Configuration Balancing for Stochastic Requests

Franziska Eberle[1(✉)], Anupam Gupta[2(✉)], Nicole Megow[3(✉)], Benjamin Moseley[2(✉)], and Rudy Zhou[2(✉)]

[1] London School of Economics and Political Science, London, UK
f.eberle@lse.ac.uk
[2] Carnegie Mellon University, Pittsburgh, PA, USA
anupamg@cs.cmu.edu, {moseleyb,rbz}@andrew.cmu.edu
[3] University of Bremen, Bremen, Germany
nicole.megow@uni-bremen.de

Abstract. The configuration balancing problem with stochastic requests generalizes well-studied resource allocation problems such as load balancing and virtual circuit routing. There are given m resources and n requests; each request has multiple possible *configurations*, each of which increases the load of each resource by some amount. The goal is to select one configuration for each request to minimize the *makespan*: the load of the most-loaded resource. In the stochastic setting, the amount by which a configuration increases the resource load is uncertain until the configuration is chosen, but we are given a probability distribution.

We develop both offline and online algorithms for configuration balancing with stochastic requests. When the requests are known offline, we give a non-adaptive policy for configuration balancing with stochastic requests that $O(\frac{\log m}{\log \log m})$-approximates the optimal adaptive policy, which matches a known lower bound for the special case of load balancing on identical machines. When requests arrive online in a list, we give a non-adaptive policy that is $O(\log m)$ competitive. Again, this result is asymptotically tight due to information-theoretic lower bounds for special cases (e.g., for load balancing on unrelated machines). Finally, we show how to leverage adaptivity in the special case of load balancing on *related* machines to obtain a constant-factor approximation offline and an $O(\log \log m)$-approximation online. A crucial technical ingredient in all of our results is a new structural characterization of the optimal adaptive policy that allows us to limit the correlations between its decisions.

Keywords: stochastic scheduling · stochastic routing · load balancing

1 Introduction

This paper considers the *configuration balancing* problem: there are m resources and n requests. Request j has q_j configurations $x_j(1), \ldots, x_j(q_j) \in \mathbb{R}^m_{\geq 0}$. We

F. Eberle—Supported by the Dutch Research Council (NWO), Netherlands Vidi grant 016.Vidi.189.087.

A. Del Pia and V. Kaibel (Eds.): IPCO 2023, LNCS 13904, pp. 127–141, 2023.
https://doi.org/10.1007/978-3-031-32726-1_10

must choose one configuration $c_j \in [q_j]$ per request, which adds $x_j(c_j)$ to the load vector on the resources. The goal is to minimize the makespan, i.e., the load of the most-loaded resource. Configuration balancing captures many natural resource allocation problems where requests compete for a finite pool of resources and the task is to find a "fair" allocation in which no resource is over-burdened. Two well-studied problems of this form arise in scheduling and routing.

(i) In *load balancing* a.k.a. *makespan minimization*, there are m (unrelated) machines and n jobs. Scheduling job j on machine i increases the load of i by $p_{ij} \geq 0$. The goal is to schedule each job on some machine to minimize the makespan, i.e., the load of the most-loaded machine.
(ii) In *virtual circuit routing* or *congestion minimization*, there is a directed graph $G = (V, E)$ on m edges with edge capacities $c_e > 0$ for $e \in E$, and n requests, each request consisting of a source-sink pair (s_j, t_j) in G and a demand $d_j \geq 0$. The goal is to route each request j from s_j to t_j via some directed path, increasing the load/congestion of each edge e on the path by d_j/c_e, while the objective is to minimize the load of the most-loaded edge.

Configuration balancing captures both problems by taking the m resources to be the m machines or edges, respectively; each configuration now corresponds to assigning a job to some machine or routing a request along some path.

Typically, job sizes or request demands are not known exactly when solving resource allocation problems in practice. This motivates the study of algorithms under uncertainty, where an algorithm must make decisions given only partial/uncertain information about the input. Uncertainty can be modeled in different ways. In exceptional cases, a *non-clairvoyant* algorithm that has *no* knowledge about the loads of requests may perform surprisingly well; an example is Graham's greedy list scheduling for load balancing on identical machines [15]. In general, a non-clairvoyant algorithm cannot perform well. Hence, we consider a stochastic model, where the unknown input follows some known distribution but the actual realization is a priori unknown. Such a model is natural when there is historical data available from which such distributions can be deduced.

In the *configuration balancing with stochastic requests* problem, we assume that each configuration c of request j is a random vector $X_j(c)$ with known distribution $\mathcal{D}_j(c)$ supported on $\mathbb{R}^m_{\geq 0}$ such that the $X_j(c)$'s are independent across different requests j. The actual realized vector of a configuration c of request j is only observed after *irrevocably* selecting this particular configuration for request j. The objective is to minimize the expected maximum load (makespan) $\mathbb{E}\Big[\max_i \sum_{j=1}^n X_{ij}(c_j)\Big]$, where c_j is the configuration chosen for request j. We assume that we have oracle access to the $\mathcal{D}_j(c)$'s; in particular we assume that in constant time, we can compute any needed statistic of the distribution $\mathcal{D}_j(c)$.

Further, we distinguish whether there is an additional dimension of uncertainty or not, namely the knowledge about the request set. In the *offline* setting, the set of requests and the distributions of the configurations of each request are known up-front, and they can be selected and assigned to the resources irre-

vocably in any order. In the *online* setting, requests are not known in advance and they are revealed one-by-one (online-list model). The algorithm learns the stochastic information on configurations of a request upon its arrival, and must select one of them without knowledge of future arrivals. After a configuration is chosen irrevocably, the next request arrives.

In general, we allow an algorithm to base the next decision on knowledge about the realized vectors of all previously selected request configurations. We call such policies *adaptive*. Conversely, a *non-adaptive* policy is one that fixes the particular configuration chosen for a request without using any knowledge of the realized configuration vectors.

The goal of this paper is to investigate the power of adaptive and non-adaptive policies for online and offline configuration balancing with stochastic requests. We quantify the performance of an algorithm by bounding the worst-case ratio of the achieved expected makespan and the minimal expected makespan achieved by an optimal offline adaptive policy. We say that an algorithm ALG *α-approximates* an algorithm ALG' if, for any input instance, the expected makespan of ALG is at most a factor α larger than the expected makespan of ALG'; we refer to α also as *approximation ratio*. For online algorithms, the term *competitive ratio* refers to their approximation ratio.

1.1 Our Results

Main Result. As our first main result, we present non-adaptive algorithms for offline and online configuration balancing with stochastic requests.

Theorem 1. *For configuration balancing with stochastic requests there is a randomized* offline *algorithm that computes a non-adaptive policy that is a $\Theta\left(\frac{\log m}{\log \log m}\right)$-approximation and an efficient deterministic* online *algorithm that is a $\Theta(\log m)$-approximation when comparing to the optimal offline adaptive policy. Both algorithms run in polynomial time in the number of resources and the total number of configurations over all requests.*

The offline analysis relies on a linear programming (LP) relaxation of configuration balancing, which has a known integrality gap of $\Theta\left(\frac{\log m}{\log \log m}\right)$, even for virtual circuit routing [25], implying that the analysis is tight. In the online setting, our analysis employs a potential function to greedily determine which configuration to choose for each request. In particular, we generalize the idea by [3] to the setting of configuration balancing with stochastic requests and match a known lower bound for online deterministic load balancing on unrelated machines by [5].

If the configurations are not given explicitly as part of the input or the number of configurations is large, then efficiently solving the problem requires us to be able to optimize over configurations in polynomial time.

Applications. These results would hold for both load balancing on unrelated machines and virtual circuit routing if we could guarantee that either the con-

figurations are given explicitly or the respective subproblems can be solved efficiently. We can ensure this in both cases.

For stochastic load balancing on unrelated machines, the resources are the m machines, and each job has m possible configurations – one corresponding to assigning that job to each machine. Thus, we can efficiently represent all configurations. Further, here the LP relaxation of configuration balancing used in Theorem 1 is equivalent to the LP relaxation of the generalized assignment problem (GAP) solved in [33], which gives a deterministic rounding algorithm. Hence, Theorem 1 implies the following theorem. We omit the proof in this extended abstract; see [14] for proof.

Theorem 2. *There exist efficient deterministic algorithms that compute a non-adaptive policy for load balancing on unrelated machines with stochastic jobs that achieve an* $\Theta\big(\frac{\log m}{\log\log m}\big)$*-approximation* offline *and an* $\Theta(\log m)$*-approximation* online *when comparing to the optimal offline adaptive policy.*

These results are asymptotically tight due to the lower bound of $\Omega\big(\frac{\log m}{\log\log m}\big)$ on the adaptivity gap [17] and the lower bound of $\Omega(\log m)$ on the competitive ratio of any deterministic online algorithm, even for deterministic requests [5]. This implies that the adaptivity gap for stochastic load balancing is $\Theta\big(\frac{\log m}{\log\log m}\big)$.

For virtual circuit routing, the resources are the m edges and each request has a configuration for each possible routing path. Thus, efficiently solving the subproblems requires more work as the configurations are only given *implicitly* and there can be exponentially many. For the offline setting, since the LP relaxation has (possibly) exponentially many variables, we design an efficient separation oracle for the dual LP in order to efficiently solve the primal. For the online setting, we carefully select a subset of polynomially many configurations that contain the configuration chosen by the greedy algorithm, even when presented with all configurations. Thus, Theorem 1 implies that stochastic requests are not harder to approximate than deterministic requests. We omit the proof in this extended abstract; see [14] for proof.

Theorem 3. *For routing with stochastic requests, there exist an efficient randomized* offline *algorithm computing a non-adaptive policy that is a* $\Theta\big(\frac{\log m}{\log\log m}\big)$*-approximation and an efficient deterministic* online *algorithm that computes an* $\Theta(\log m)$*-approximation when comparing to the optimal offline adaptive policy.*

Adaptive Policies for Related Machines. When each request j has m configurations and configuration $c \in [m]$ can be written as $X_j(c) = \frac{X_j}{s_i}\, e_c$, where $e_c \in \mathbb{R}^m$ is the cth standard unit vector, the problem is also known as *load balancing on related machines*. We say that X_j is the size of request (or job) j and s_i is the speed of resource (or machine) i. In this special case, we show how to leverage adaptivity to overcome the $\Omega\big(\frac{\log m}{\log\log m}\big)$ lower bound on the adaptivity gap. Interestingly, our adaptive algorithms begin with a similar *non-adaptive* assignment of jobs to machines, but we deviate from the assignment adaptively to obtain our improved algorithms.

Theorem 4. *For load balancing on related machines with stochastic jobs, there exist efficient deterministic algorithms that compute an adaptive* offline $O(1)$*-approximation and an adaptive* online $O(\log \log m)$*-approximation when comparing to the optimal offline adaptive policy.*

It remains an interesting open question whether the online setting admits an $O(1)$-competitive algorithm.

1.2 Technical Overview

We illustrate the main idea behind our non-adaptive policies, which compare to the optimal offline adaptive policy. Throughout this paper, we let Opt denote the optimal adaptive policy as well as its makespan. As in many other stochastic optimization problems, our goal is to give a good deterministic proxy for the makespan of a policy. Then, our algorithm will optimize over this deterministic proxy to obtain a good solution. First, we observe that if all configurations were bounded with respect to $\mathbb{E}[\text{Opt}]$ in every entry, then selecting configurations such that each resource has expected load $O(\mathbb{E}[\text{Opt}])$ gives the desired $O\big(\frac{\log m}{\log \log m}\big)$-approximation by standard concentration inequalities for independent sums with bounded increments. Thus, in this case the expected load on each resource is a good proxy. However, in general, we have no upper bound on $X_{ij}(c)$, so we cannot argue as above. We turn these unbounded random variables (RVs) into bounded ones in a standard way by splitting each request into *truncated* and *exceptional* parts.

Definition 1 (Truncated and Exceptional Parts). *Fix $\tau \geq 0$ as threshold. For a RV X, its truncated part (w.r.t. threshold τ) is $X^T := X \cdot \mathbb{1}_{X<\tau}$ and its exceptional part is $X^E := X \cdot \mathbb{1}_{X\geq\tau}$. Note that $X = X^T + X^E$.*

It is immediate that the truncated parts $X^T_{ij}(c)$ are bounded in $[0, \tau]$. Taking $\tau = O(\mathbb{E}[\text{Opt}])$, we can control their contribution to the makespan using concentration. It remains to find a good proxy for the contribution of exceptional parts to the makespan. This is one of the main technical challenges of our work as we aim to compare against the optimal adaptive policy: adaptive policies have much better control over the exceptional parts than non-adaptive ones.

Concretely, let c_j be the configuration chosen by some fixed policy for request j. Note that c_j itself can be a random variable in $\{1, \ldots, q_j\}$. We want to control the quantity $\mathbb{E}\big[\max_i \sum_{j=1}^n X^E_{ij}(c_j)\big]$. Since we have no reasonable bound on the $X^E_{ij}(c_j)$'s, for non-adaptive policies, we can only upper bound the expected maximum by the following sum

$$\mathbb{E}\Big[\max_{1\leq i\leq m} \sum_{j=1}^{n} X^E_{ij}(c_j)\Big] \leq \sum_{j=1}^{n} \mathbb{E}\Big[\max_{1\leq i\leq m} X^E_{ij}(c_j)\Big]. \tag{1}$$

We call the right hand side *total (expected) exceptional load*. The above inequality is tight up to constants for non-adaptive policies, so it seems like the total expected exceptional load is a good proxy to use for our algorithm. However, it is far from tight for adaptive policies as the following example shows.

Example 1. Recall that in load balancing on related machines, each request j has m configurations and configuration $c \in [m]$ has the special form of $X_j(c) = \frac{X_j}{s_i} e_c$, where X_j is the processing time of job j and s_i is the speed of machine i. We assume that there is one fast machine with speed $s_1 = 1$ and $m-1$ slow machines with speed $s_2 = \ldots = s_m = \frac{1}{\tau m}$, where $\tau > 0$ is the truncation threshold. There are m jobs: a stochastic one with processing time $X_j \sim \tau \cdot \mathrm{Ber}\left(\frac{1}{\tau}\right)$ and $m-1$ deterministic jobs with processing time $X_j \equiv \frac{1}{m}$. The optimal adaptive policy first schedules the stochastic job on the fast machine. If its realized size is 0, then it schedules all deterministic jobs on the fast machine as well. Otherwise the realized size is τ and it schedules one deterministic job on each slow machine, implying $\mathbb{E}[\textsc{Opt}] = \left(1 - \frac{1}{\tau}\right)\left(\frac{m-1}{m}\right) + \frac{1}{\tau} \cdot \tau = \Theta(1)$. However, the total expected exceptional load (w.r.t. τ) is $\sum_{i,j} \mathbb{E}\left[X_{ij}^E \cdot \mathbb{1}_{j \to i}\right] = \frac{1}{\tau}(m\tau) = m$, where $j \to i$ denotes that job j is assigned to machine i, i.e., configuration i is chosen for j.

In the example, the optimal adaptive policy accrues a lot of exceptional load, but this does not have a large effect on the makespan. Concretely, (1) can be loose by a $\Omega(m)$-factor for adaptive policies. Thus, it seems that the total exceptional load is a bad proxy in terms of lower-bounding Opt. However, we show that, by comparing our algorithm to a *near-optimal* adaptive policy rather than the optimal one, the total exceptional load becomes a good proxy in the following sense. This is the main technical contribution of our work, and it underlies all of our algorithmic techniques.

Theorem 5. *For configuration balancing with stochastic requests, there exists an adaptive policy with expected maximum load and total expected exceptional load at most* $2 \cdot \mathbb{E}[\textsc{Opt}]$ *with respect to any truncation threshold* $\tau \geq 2 \cdot \mathbb{E}[\textsc{Opt}]$. *Further, any configuration* c *selected by this policy satisfies* $\mathbb{E}\left[\max_i X_i(c)\right] \leq \tau$.

The proof of the above relies on carefully modifying the "decision tree" representing the optimal adaptive policy; see [14] for proof. In light of Theorem 5, the deterministic proxies we consider are the expected truncated load on each resource and the total expected exceptional load. All of our algorithms then proceed by ensuring that both quantities are bounded with respect to $\mathbb{E}[\textsc{Opt}]$. In the offline case, we round a natural assignment-type linear program (LP), and in the online case, we use a potential-function argument. All of these algorithms actually output non-adaptive policies. For the special case of related-machines load balancing, we also compute a non-adaptive assignment but instead of following it exactly, we deviate using adaptivity and give improved solutions.

1.3 Related Work

While stochastic optimization problems have long been studied [6,11], approximation algorithms for them are more recent [13,29]. By now, multi-stage stochastic problems (where uncertain information is revealed in stages) are well-understood [9,19,34]. In contrast, more dynamic models, where the exact value of an unknown parameter becomes known at times depending on the algorithms

decisions (serving a request) still remain poorly understood. Some exceptions come from stochastic knapsack [8,12,16,28] as well as stochastic scheduling and routing which we discuss below.

Scheduling. For load balancing with deterministic sizes, a 2-approximation in the most general unrelated-machines offline setting [26] is known. For identical machines ($p_{ij} = p_j$ for all jobs j), the greedy algorithm (called *list scheduling*) is a $\left(2 - \frac{1}{m}\right)$-approximation algorithm [15]. This guarantee holds even when the jobs arrive online and *nothing* is known about job sizes. This implies a $\left(2 - \frac{1}{m}\right)$-approximate *adaptive* policy for stochastic load balancing on identical machines.

Apart from this, prior work on stochastic scheduling has focused on approximating the optimal *non-adaptive* policy. There are non-adaptive $O(1)$-approximations known for identical machines [24], unrelated machines [17], the ℓ_q-norm objective [30], and monotone, symmetric norms [21].

In contrast, our work focuses on approximating the stronger optimal *adaptive* policy. The *adaptivity gap* (the ratio between the expected makespan of the optimal adaptive and non-adaptive policies) can be $\Omega\left(\frac{\log m}{\log \log m}\right)$ even for the simplest case of identical machines [17]. Thus, previous work on approximating the optimal non-adaptive policy does not immediately give any non-trivial approximation guarantees for our setting. The only previous work on adaptive stochastic policies for load-balancing (beyond the highly-adaptive list scheduling) is by [32]. They propose scheduling policies whose degree of adaptivity can be controlled by parameters and show an approximation factor of $O(\log \log m)$ for scheduling on identical machines.

Online load balancing with deterministic jobs is also well studied [4]. On identical machines, the aforementioned list scheduling algorithm [15] is $\left(2 - \frac{1}{m}\right)$-competitive. For unrelated machines, there is a deterministic $O(\log m)$-competitive algorithm [3] and this is best possible [5]. When the machines are uniformly related, [7] design an $O(1)$-competitive algorithm for minimizing the makespan. [22,23] study the multi-dimensional generalization to vector scheduling under the makespan and the ℓ_q-norm objective.

To the best of our knowledge, configuration balancing has not been explicitly defined before. The techniques of [3] give an $O(\log m)$-competitive algorithm for deterministic requests. It is also studied for packing integer programs [1,2,18].

Routing. For oblivious routing with stochastic demands, [20] give an algorithm which is an $O(\log^2 n)$-approximation with high probability. Here, "oblivious" refers to the requirement that the chosen path between a source-sink pair must not depend on the current congestion of the network. In particular, after specifying a set of paths for each possible source-sink pair, a demand matrix is drawn from an a-priori known distribution and each demand needs to be routed along one of the predefined paths. The obliviousness requirement is very different from our setting and makes the two models essentially incomparable.

When $d_j = 1$ for each source-sink pair, there is an $O\left(\frac{\log m}{\log \log m}\right)$-approximation algorithm by [31], which is best possible, unless NP $\subseteq$ ZPTIME$(n^{\log \log n})$ [10].

In the online setting, when the source-sink pairs arrive online in a list and have to be routed before the next pair arrives, [3] give a lower bound of $\Omega(\log n)$ on the competitive ratio of any deterministic online algorithm in directed graphs, where n is the number of vertices. They also give a matching upper bound. For more details on online routing we refer to the survey [27].

2 Configuration Balancing with Stochastic Requests

In this section, we prove our main results for the most general problem we consider: configuration balancing. We give an $O\big(\frac{\log m}{\log \log m}\big)$-approximation offline and an $O(\log m)$-approximation online; both algorithms are non-adaptive. Before describing the algorithms, we give our main structural theorem that enables all of our results. Roughly, we show that instead of comparing to the optimal adaptive policy, by losing only a constant factor in the approximation ratio, we can compare to a near-optimal policy that behaves like a non-adaptive one (w.r.t. the proxy objectives we consider, namely, the total expected exceptional load).

2.1 Structural Theorem

In this section, we show that there exists a near-optimal policy as guaranteed by Theorem 5. To this end, we modify the optimal policy by "restarting" whenever an exceptional request is encountered. Additionally, we ensure that this modified policy never selects a configuration c for a request j with $\mathbb{E}\big[\max_i X_{ij}(c)\big] > \tau$.

We let J denote the set of requests. For any subset $J' \subseteq J$, we let $\textsc{Opt}(J')$ denote the optimal adaptive policy (and its maximum load) on the set of requests J'. Note that $\textsc{Opt}(\emptyset) = 0$. Our (existential) algorithm to construct such a policy will begin by running the optimal policy $\textsc{Opt}(J)$ on all requests. However, once an exceptional request is encountered or the next decision will choose a configuration with too large expected maximum, we cancel $\textsc{Opt}(J)$ and instead recurse on all remaining requests, ignoring all previously-accrued loads; see Algorithm 1. The idea of our analysis is that we recurse with small probability; see [14].

Theorem 5. *For configuration balancing with stochastic requests, there exists an adaptive policy with expected maximum load and total expected exceptional load at most* $2 \cdot \mathbb{E}[\textsc{Opt}]$ *with respect to any truncation threshold* $\tau \geq 2 \cdot \mathbb{E}[\textsc{Opt}]$. *Further, any configuration* c *selected by this policy satisfies* $\mathbb{E}\big[\max_i X_i(c)\big] \leq \tau$.

Having this near-optimal policy at hand, the upshot is that we can bound our subsequent algorithms with respect to the following LP relaxation ($\mathsf{LP_C}$) for configuration balancing with stochastic requests. The variable y_{cj} denotes selecting configuration c for request j. We take our threshold between the truncated and exceptional parts to be τ. Using the natural setting of the y-variables as the probabilities of the policy from Theorem 5, it is straight-forward to show that

Algorithm 1: Policy $S(J)$

```
R ← J                                                    // remaining requests
if R = ∅ then
    return empty policy                                  // finish
while R ≠ ∅ do
    j ← first / next request considered by Opt(J)
    c_j ← configuration chosen for request j by Opt(J)
1   if E[max_i X_ij(c_j)] > τ then                       // maximum too large
        break
    else
        choose c_j for request j                         // S(J) follows Opt(J)
        R ← R \ {j}                                      // update remaining requests
2       if max_i X_ij(c_j) ≥ τ then                      // exceptional configuration observed
            break
run S(R)                                                 // recurse with remaining requests
```

the following LP relaxation has a feasible solution, formalized in Lemma 1.

$$\begin{aligned}
\textstyle\sum_{c=1}^{q_j} y_{cj} &= 1 && \forall\, j \in [n] \\
\textstyle\sum_{j=1}^{n}\sum_{c=1}^{q_j} \mathbb{E}[X^T_{ij}(c)] \cdot y_{cj} &\le \tau && \forall\, i \in [m] \\
\textstyle\sum_{j=1}^{n}\sum_{c=1}^{q_j} \mathbb{E}[\max_i X^E_{ij}(c)] \cdot y_{cj} &\le \tau && \\
y_{cj} &= 0 && \forall\, j \in [n], \forall\, c \in [q_j] : \mathbb{E}[\max_i X_{ij}(c)] > \tau \\
y_{cj} &\ge 0 && \forall\, j \in [n], \forall\, c \in [q_j]
\end{aligned} \tag{$\mathsf{LP_C}$}$$

Lemma 1. ($\mathsf{LP_C}$) *has a feasible solution for any* $\tau \ge 2 \cdot \mathbb{E}[\textsc{Opt}]$.

2.2 Offline Setting

Our offline algorithm is the natural randomized rounding of ($\mathsf{LP_C}$). For the truncated parts, the following inequality bounds their contribution to the makespan.

Lemma 2. *Let* $S_1, \dots, S_m$ *be sums of independent RVs bounded in* $[0, \tau]$ *for some* $\tau > 0$ *such that* $\mathbb{E}[S_i] \le \tau$ *for all* $i \in [m]$. *Then,* $\mathbb{E}[\max_i S_i] = O\big(\frac{\log m}{\log\log m}\big)\tau$.

To bound the contribution of the exceptional parts, we use (1), i.e., the total expected exceptional load. Using binary search for the correct choice of τ and re-scaling the instance down by the current value of τ, it suffices to give an efficient algorithm that either

- outputs a non-adaptive policy with expected makespan $O\big(\frac{\log m}{\log\log m}\big)$, or
- certifies that $\mathbb{E}[\textsc{Opt}] > 1$.

This is because for $\tau \in \big(\mathbb{E}[\textsc{Opt}], 2 \cdot \mathbb{E}[\textsc{Opt}]\big]$, the re-scaling guarantees $\mathbb{E}[\textsc{Opt}] \in [\frac{1}{2}, 1)$ on the scaled instance, in which case the algorithm achieves expected makespan $O\big(\frac{\log m}{\log\log m}\big) = O\big(\frac{\log m}{\log\log m}\big) \cdot \mathbb{E}[\textsc{Opt}]$.

Algorithm 2: Offline Configuration Balancing with Stochastic Requests

try to solve $(\mathsf{LP_C})$ with $\tau = 2$
if $(\mathsf{LP_C})$ *is feasible* **then**
  let y^* be the outputted feasible solution
  for *each request* j **do**
    independently sample $c \in [q_j]$ with probability y^*_{cj}
    choose sampled c as c_j
else
  return "$\mathbb{E}[\textsc{Opt}] > 1$"

To that end, we use the natural independent randomized rounding of $(\mathsf{LP_C})$. That is, if $(\mathsf{LP_C})$ has a feasible solution y^*, for request j, we choose configuration c as configuration c_j independently with probability y^*_{cj}; see Algorithm 2.

If the configurations are given explicitly as part of the input, then $(\mathsf{LP_C})$ can be solved in polynomial time and, thus, Algorithm 2 runs in polynomial time. Hence, the $O\big(\frac{\log m}{\log\log m}\big)$-approximate non-adaptive policy for configuration balancing with stochastic requests (Theorem 1) follows from the next lemma.

Lemma 3. *If* $(\mathsf{LP_C})$ *can be solved in polynomial time, Algorithm 2 is a polynomial-time randomized algorithm that either outputs a non-adaptive policy with expected makespan* $O\big(\frac{\log m}{\log\log m}\big)$*, or certifies correctly that* $\mathbb{E}[\textsc{Opt}] > 1$.

2.3 Online Setting

We now consider online configuration balancing where n stochastic requests arrive online one-by-one, and for each request, one configuration has to be irrevocably selected before the next request appears. We present a non-adaptive online algorithm that achieves a competitive ratio of $O(\log m)$, which is best possible due to the lower bound of $\Omega(\log m)$ [5].

By a standard guess-and-double scheme, we may assume that we have a good guess of $\mathbb{E}[\textsc{Opt}]$. We omit the proof, which is analogous to its virtual-circuit-routing counterpart in [3].

Lemma 4. *Given an instance of online configuration balancing with stochastic requests, suppose there exists an online algorithm that, given parameter* $\lambda > 0$*, never creates an expected makespan more than* $\alpha \cdot \lambda$*, possibly terminating before handling all requests. Further, if the algorithm terminates prematurely, then it certifies that* $\mathbb{E}[\textsc{Opt}] > \lambda$*. Then, there exists an* $O(\alpha)$*-competitive algorithm for online configuration balancing with stochastic requests. Further, the resulting algorithm preserves non-adaptivity.*

We will build on the same technical tools as in the offline case. In particular, we wish to compute a non-adaptive assignment online with small expected truncated load on each resource and small total expected exceptional load. To achieve this, we generalize the greedy potential function approach of [3]. Our two new ingredients are to treat the exceptional parts of a request's configuration as a

resource requirement for an additional, artificial resource and to compare the potential of our solution directly with a *fractional* solution to ($\mathsf{LP_C}$).

Now we describe our potential function, which is based on an exponential/soft-max function. Let λ denote the current guess of the optimum as required by Lemma 4. We take $\tau = 2\lambda$ as our truncation threshold. Given load vector $L \in \mathbb{R}^{m+1}$, our potential function is

$$\phi(L) = \sum_{i=0}^{m} (3/2)^{L_i/\tau}.$$

For $i \in [m]$, we ensure the ith entry of L is the *expected* truncated load on resource i and use the 0th entry as a virtual resource that is the total expected exceptional load. For any request j, let L_j be the expected load vector after handling the first j requests, with L_{ij} denoting its ith entry. Let $L_{i0} := 0$ for all i. Upon arrival of request j, our algorithm tries to choose the configuration $c_j \in [q_j]$ that minimizes the increase in potential; see Algorithm 3.

Algorithm 3: Online Configuration Balancing with Stochastic Requests

$\ell \leftarrow \log_{3/2}(2m+2)$
$\lambda \leftarrow$ current guess of $\mathbb{E}[\textsc{Opt}]$
$\tau \leftarrow 2\lambda$ truncation threshold
upon *arrival of request* j **do**
1 $\quad c_j \leftarrow \arg\min_{c \in [q_j]} \Big((3/2)^{(L_{0j-1} + \mathbb{E}[\max_{i\in[m]} X^E_{ij}(c)])/\tau} + \sum_{i=1}^{m} (3/2)^{(L_{ij-1} + \mathbb{E}[X^T_{ij}(c)])/\tau} \Big) - \phi(L_{j-1})$
$\quad$ **if** $L_{ij-1} + \mathbb{E}[X^T_{ij}(c_j)] \le \ell\tau$ *for all* $i \in [m]$ **and** $L_{0j-1} + \mathbb{E}[\max_{i\in[m]} X^E_{ij}(c_j)] \le \ell\tau$ **then**
$\quad\quad$ choose c_j for j
$\quad\quad$ $L_{ij} \leftarrow L_{ij-1} + \mathbb{E}[X^T_{ij}(c_j)]$ for all $i \in [m]$
$\quad\quad$ $L_{0j} \leftarrow L_{0j-1} + \mathbb{E}[\max_{i\in[m]} X^E_{ij}(c_j)]$
$\quad$ **else**
$\quad\quad$ return "$\mathbb{E}[\textsc{Opt}] > \lambda$"

To analyze this algorithm, we compare its makespan with a solution to ($\mathsf{LP_C}$). This LP has an integrality gap of $\Omega\big(\frac{\log m}{\log\log m}\big)$, which follows immediately from the path assignment LP for virtual circuit routing [25]. Hence, a straightforward analysis of Algorithm 3 comparing to a rounded solution to ($\mathsf{LP_C}$) gives an assignment with expected truncated load per machine and total expected exceptional load $O\big(\log m \cdot \frac{\log m}{\log\log m}\big) \cdot \mathbb{E}[\textsc{Opt}]$. To get a tight competitive ratio of $O(\log m)$, we avoid the integrality gap by comparing to a *fractional* solution to ($\mathsf{LP_C}$), and we use a slightly different inequality than Lemma 2 for the regime where the mean of the sums is larger than the increments by at most a $O(\log m)$-factor.

Lemma 5. *Let $S_1, \ldots, S_m$ be sums of independent RVs bounded in $[0, \tau]$ for $\tau > 0$ such that $\mathbb{E}[S_i] \le O(\log m)\tau$ for all $1 \le i \le m$. Then, $\mathbb{E}[\max_i S_i] \le O(\log m)\tau$.*

We give the guarantee for Algorithm 3, which implies the $O(\log m)$-competitive algorithm for online configuration balancing with stochastic requests.

Lemma 6. *Suppose the minimizing configuration in Line 1 can be found in polynomial time. Then Algorithm 3 runs in polynomial time; it is deterministic, non-adaptive and correctly solves the subproblem of Lemma 4 for* $\alpha = O(\log m)$.

3 Load Balancing on Related Machines

In this section, we improve on Theorem 2 in the special case of related machines, where each machine i has a speed parameter $s_i > 0$ and each job j an independent size X_j such that $X_{ij} = \frac{X_j}{s_i}$. Recall that we gave a non-adaptive $O\big(\frac{\log m}{\log\log m}\big)$-approximation for unrelated machines. However, the adaptivity gap is $\Omega\big(\frac{\log m}{\log\log m}\big)$ even for load balancing on identical machines where every machine has the same speed. Thus, to improve on Theorem 2, we need to use adaptivity.

The starting point of our improved algorithms is the same non-adaptive assignment for unrelated-machines load balancing. However, instead of non-adaptively assigning job j to the specified machine i, we adaptively assign j to the least loaded machine with similar speed to i. We formalize this idea and briefly explain the algorithms for offline and online load balancing on related machines.

Machine Smoothing. In this part, we define a notion of *smoothed machines*. We show that by losing a constant factor in the approximation ratio, we may assume that the machines are partitioned into at most $O(\log m)$ groups such that machines within a group have the same speed and the size of the groups shrinks geometrically. Thus, by "machines with similar speed to i," we mean machines in the same group.

Formally, we transform an instance $\mathcal{I}$ of load balancing on m related machines with stochastic jobs into an instance $\mathcal{I}_s$ with so-called "smoothed machines" and the same set of jobs with the following three properties:

(i) The machines are partitioned into $m' = O(\log m)$ groups such that group k consists of m_k machines with speed exactly s_k such that $s_1 < s_2 < \cdots < s_{m'}$.
(ii) For all groups $1 \le k < m'$, we have $m_k \ge \frac{3}{2} m_{k+1}$.
(iii) $\textsc{Opt}(\mathcal{I}_s) = O(\textsc{Opt}(\mathcal{I}))$.

To this end, we suitably decrease machine speeds and delete machines from the original instance $\mathcal{I}$; see [14] for the algorithm and the technical details.

Lemma 7. *There is an efficient algorithm that, given an instance* $\mathcal{I}$ *of load balancing with* m *related machines and stochastic jobs, computes an instance* $\mathcal{I}_s$ *of smoothed machines with the same set of jobs satisfying Properties (i) to (iii).*

A similar idea for machine smoothing has been employed by Im et al. [23] for deterministic load balancing on related machines. In their approach, they ensure that the *total processing power* of the machines in a group decreases geometrically rather than the number of machines.

Offline Setting. We run Algorithm 2 on the configuration balancing instance defined by the load balancing instance with smoothed machines. Given a job-to-machine assignment, we list schedule the jobs assigned to a particular group on the machines of this group. In the proof of Theorem 4, we rely on the following strong bound on the expected maximum of the truncated load; see [14].

Lemma 8. *Let* $c_1, \ldots, c_m \in \mathbb{N}_{\geq 1}$ *be constants such that* $c_i \geq \frac{3}{2} c_{i+1}$ *for all* $1 \leq i \leq m$. *Let* $S_1, \ldots, S_m$ *be sums of independent random variables bounded in* $[0, \tau]$ *such that* $\mathbb{E}[S_i] \leq c_i \tau$ *for all* $1 \leq i \leq m$. *Then,* $\mathbb{E}\big[\max_i \frac{S_i}{c_i}\big] \leq O(\tau)$.

Online Setting. We apply a similar framework as above. Note that our online configuration balancing algorithm loses a logarithmic factor in the number of resources, so to obtain a $O(\log \log m)$-approximation, we aggregate each group (in the smoothed-machines instance) as a single resource. Intuitively, this definition captures the fact that we will average all jobs assigned to a group over the machines in this group. Thus, our configuration balancing instance will have only $O(\log m)$ resources and applying Theorem 1 proves Theorem 4; see [14].

Conclusion

We considered the configuration balancing problem under uncertainty. In contrast to the (often overly optimistic) clairvoyant settings and the (often overly pessimistic) non-clairvoyant settings, we consider the stochastic setting where each request j presents a set of random vectors, and we need to (adaptively) pick one of these vectors, to minimize the *expected* maximum load over the m resources. We give logarithmic bounds for several general settings (which are existentially tight), and a much better $O(1)$ offline and $O(\log \log m)$ online bound for the related machines setting. Closing the gap for online related-machines load balancing remains an intriguing open problem. More generally, getting a better understanding of both adaptive and non-adaptive algorithms for stochastic packing and scheduling problems remains an exciting direction for research.

References

1. Agrawal, S., Devanur, N.R.: Fast algorithms for online stochastic convex programming. In: Proceedings of SODA, pp. 1405–1424 (2015)
2. Agrawal, S., Wang, Z., Ye, Y.: A dynamic near-optimal algorithm for online linear programming. Oper. Res. **62**(4), 876–890 (2014)
3. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S.A., Waarts, O.: On-line routing of virtual circuits with applications to load balancing and machine scheduling. J. ACM **44**(3), 486–504 (1997)

4. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. LNCS, vol. 1442, pp. 178–195. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0029569
5. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**(2), 221–237 (1995)
6. Beale, E.M.L.: On minimizing a convex function subject to linear inequalities. J. Roy. Stat. Soc. Ser. B. Methodol. **17**, 173–184; discussion, 194–203 (1955)
7. Berman, P., Charikar, M., Karpinski, M.: On-line load balancing for related machines. J. Algorithms **35**(1), 108–121 (2000)
8. Bhalgat, A., Goel, A., Khanna, S.: Improved approximation results for stochastic knapsack problems. In: Proceedings of SODA, pp. 1647–1665. SIAM (2011)
9. Charikar, M., Chekuri, C., Pál, M.: Sampling bounds for stochastic optimization. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX/RANDOM -2005. LNCS, vol. 3624, pp. 257–269. Springer, Heidelberg (2005). https://doi.org/10.1007/11538462_22
10. Chuzhoy, J., Guruswami, V., Khanna, S., Talwar, K.: Hardness of routing with congestion in directed graphs. In: Proceedings of STOC, pp. 165–178. ACM (2007)
11. Dantzig, G.B.: Linear programming under uncertainty. Manag. Sci. **1**, 197–206 (1955)
12. Dean, B.C., Goemans, M.X., Vondrák, J.: Approximating the stochastic knapsack problem: the benefit of adaptivity. Math. Oper. Res. **33**(4), 945–964 (2008)
13. Dye, S., Stougie, L., Tomasgard, A.: The stochastic single resource service-provision problem. Naval Res. Logist. **50**(8), 869–887 (2003)
14. Eberle, F., Gupta, A., Megow, N., Moseley, B., Zhou, R.: Configuration balancing for stochastic requests. CoRR, abs/2208.13702 (2022)
15. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM J. Appl. Math. **17**(2), 416–429 (1969)
16. Gupta, A., Krishnaswamy, R., Molinaro, M., Ravi, R.: Approximation algorithms for correlated knapsacks and non-martingale bandits. In: Ostrovsky, R. (ed.) Proceedings of FOCS, pp. 827–836. IEEE Computer Society (2011)
17. Gupta, A., Kumar, A., Nagarajan, V., Shen, X.: Stochastic load balancing on unrelated machines. Math. Oper. Res. **46**(1), 115–133 (2021)
18. Gupta, A., Molinaro, M.: How the experts algorithm can help solve LPS online. Math. Oper. Res. **41**(4), 1404–1431 (2016)
19. Gupta, A., Pál, M., Ravi, R., Sinha, A.: Sampling and cost-sharing: approximation algorithms for stochastic optimization problems. SIAM J. Comput. **40**(5), 1361–1401 (2011)
20. Hajiaghayi, M.T., Kim, J.H., Leighton, T., Räcke, H.: Oblivious routing in directed graphs with random demands. In: Proceedings of STOC, pp. 193–201. ACM (2005)
21. Ibrahimpur, S., Swamy, C.: Approximation algorithms for stochastic minimum-norm combinatorial optimization. In: Proceedings of FOCS, pp. 966–977. IEEE (2020)
22. Im, S., Kell, N., Kulkarni, J., Panigrahi, D.: Tight bounds for online vector scheduling. SIAM J. Comput. **48**(1), 93–121 (2019)
23. Im, S., Kell, N., Panigrahi, D., Shadloo, M.: Online load balancing on related machines. In: Proceedings of STOC, pp. 30–43. ACM (2018)
24. Kleinberg, J.M., Rabani, Y., Tardos, É.: Allocating bandwidth for bursty connections. SIAM J. Comput. **30**(1), 191–217 (2000)
25. Leighton, T., Rao, S., Srinivasan, A.: Multicommodity flow and circuit switching. In: HICSS (7), pp. 459–465. IEEE Computer Society (1998)

26. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. **46**, 259–271 (1990)
27. Leonardi, S.: On-line network routing. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. LNCS, vol. 1442, pp. 242–267. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0029572
28. Ma, W.: Improvements and generalizations of stochastic knapsack and Markovian bandits approximation algorithms. Math. Oper. Res. **43**(3), 789–812 (2018)
29. Möhring, R.H., Schulz, A.S., Uetz, M.: Approximation in stochastic scheduling: the power of LP-based priority policies. J. ACM **46**(6), 924–942 (1999)
30. Molinaro, M.: Stochastic p load balancing and moment problems via the l-function method. In: Proceedings of SODA, pp. 343–354. SIAM (2019)
31. Raghavan, P., Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica **7**(4), 365–374 (1987)
32. Sagnol, G., Schmidt, D., Waldschmidt, G.: Restricted adaptivity in stochastic scheduling. In: Proceedings of ESA. LIPIcs, vol. 204, pp. 79:1–79:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
33. Shmoys, D.B., Tardos, É.: An approximation algorithm for the generalized assignment problem. Math. Program. **62**, 461–474 (1993)
34. Swamy, C., Shmoys, D.B.: Sampling-based approximation algorithms for multi-stage stochastic optimization. SIAM J. Comput. **41**(4), 975–1004 (2012)