# Online Demand Scheduling with Failovers

**Konstantina Mellou**
Microsoft Research, Redmond, WA, USA

**Marco Molinaro**
Microsoft Research, Redmond, WA, USA
PUC-Rio de Janeiro, Brazil

**Rudy Zhou**[1]
Carnegie Mellon University, Pittsburgh, PA, USA

## Abstract

Motivated by cloud computing applications, we study the problem of how to optimally deploy new hardware subject to both power and robustness constraints. To model the situation observed in large-scale data centers, we introduce the *Online Demand Scheduling with Failover* problem. There are $m$ identical devices with capacity constraints. Demands come one-by-one and, to be robust against a device failure, need to be assigned to a pair of devices. When a device fails (in a failover scenario), each demand assigned to it is rerouted to its paired device (which may now run at increased capacity). The goal is to assign demands to the devices to maximize the total utilization subject to both the normal capacity constraints as well as these novel failover constraints. These latter constraints introduce new decision tradeoffs not present in classic assignment problems such as the Multiple Knapsack problem and AdWords.

In the worst-case model, we design a deterministic $\approx \frac{1}{2}$-competitive algorithm, and show this is essentially tight. To circumvent this constant-factor loss, which represents substantial capital losses for big cloud providers, we consider the stochastic arrival model, where all demands come i.i.d. from an unknown distribution. In this model we design an algorithm that achieves sub-linear additive regret (i.e. as $\mathsf{OPT}$ or $m$ increases, the multiplicative competitive ratio goes to 1). This requires a combination of different techniques, including a configuration LP with a non-trivial post-processing step and an online monotone matching procedure introduced by Rhee and Talagrand.

**2012 ACM Subject Classification** Theory of computation → Online algorithms

**Keywords and phrases** online algorithms, approximation algorithms, resource allocation

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2023.92

**Category** Track A: Algorithms, Complexity and Games

**Related Version** *Full Version*: `https://arxiv.org/abs/2209.00710`

**Funding** *Marco Molinaro*: Supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES, Brasil) - Finance Code 001, by Bolsa de Produtividade em Pesquisa #312751/2021-4 from CNPq, FAPERJ grant "Jovem Cientista do Nosso Estado".

**Acknowledgements** We thank the anonymous reviewers for their valuable suggestions. We also thank Alok Gautam Kumbhare and Ishai Menache for useful discussions.

## 1 Introduction

A critical challenge faced by cloud providers is how to deploy new hardware to satisfy the increasing demand for cloud resources, and the main bottleneck in this process is power. Data centers consist of power devices with limited capacity and each demand for hardware (e.g.

[1] Work performed as intern at Microsoft Research, Redmond.







rack of servers) has a power requirement. The goal is to assign demands to power devices to fulfill their requirements while using the available power in the data centers efficiently. This allows cloud providers to maximize return on investment on existing data centers before incurring large capital expenses for new data centers to accommodate additional demand.

An important consideration that sets this demand assignment process apart from other applications is *reliability*. Cloud users are promised a high availability of service which mandates that cloud capacity can only be unavailable for very short durations (between a few minutes and a few hours per year). As a result, assigning each demand to a single power device leads to an unacceptable level of risk; if that device fails, the capacity for the demand becomes unavailable, leading to potentially millions of dollars in costs for the provider and jeopardizing the cloud business model that is highly dependent on users' trust. To this end, power *redundancy* is built into the assignment process.

We consider a specific model of redundancy used by large cloud providers [19]. In this model, each demand gets assigned to two power devices. In normal operations (no device failure), the demand obtains half of its required power from each device. If one of the devices fails, the remaining device must provide the full power amount to the demand (see Figure 1 for an example). In these failover scenarios, the remaining devices may run at an increased capacity temporarily to accommodate their increased load. The provider uses this time to take ad-hoc corrective actions, for instance, shut down certain workloads and reduce the power of others in order to bring the power utilization of each device back within its normal limits. As in [19] we consider a single device failure at a time.

This architecture is favored in practice because it provides strong reliability guarantees with a small increase in overhead and complexity. One could consider more complex architectures, where demands could be assigned with a power split other than half-half to each device or to more than two devices, but this comes at an increased cost in hardware and operational complexity. Further, a common goal of large cloud providers is to provide statistical guarantees for high service availability, e.g., 99.99% availability for certain cloud resources or services; cloud operators have determined that accounting for a single device failure with the described architecture provides such target guarantees.

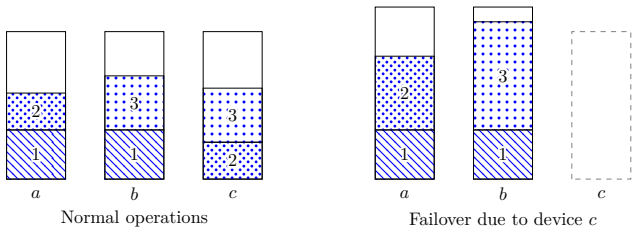
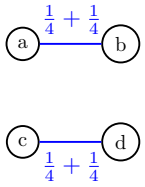
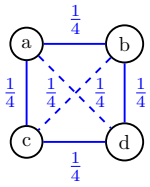



**Figure 1** In normal operations (left), each demand (denoted with a different pattern) is assigned to two devices and gets half of its required power from each device. In the failover scenario where device $c$ has failed (right), the demands that were assigned to $c$ now get their full power from the remaining devices that may run at increased capacity.

We introduce the *Online Demand Scheduling with Failover* problem (FAILOVER) to model this issue of assigning demands to power devices with redundancy. Formally, in this problem there are $m$ identical devices (or machines) and $n$ demands. Each device has two capacities: a nominal capacity that is normalized to 1 and a failover capacity $B \geq 1$. Each demand $j$ has some size $s_j \geq 0$, which for convenience is defined as its per-device power requirement (so the total power requirement of the demand is $2s_j$). The demands arrive online one-by-one and there is no knowledge about future demands. The goal is to irrevocably assign the arriving

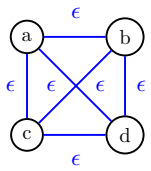



**(a)** Bad assignment.

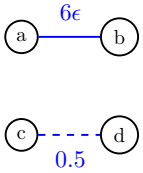



**(b)** Good assignment.

**Figure 2** Illustration of Example 1.




**(a)** Bad assignment.




**(b)** Good assignment.

**Figure 3** Illustration of Example 2.

demands to pairs of devices (or edges, where we consider each device as a node) satisfying:

1. (Nominal Constraints) For every device $u$, its total load has to be at most 1, namely $L_u := \sum_{v \neq u} L_{uv} \leq 1$, where we define $L_{uv} = \sum_{j \to uv} s_j$ to be the total load on edge $uv$ (i.e., all demands assigned to the pair of devices $uv$).
2. (Failover Constraints) For every device $u$, we have $L_u + \max_{v \neq u} L_{uv} \leq B$ (i.e., if a device $v \neq u$ fails, all demands assigned to $uv$ have to be supplied solely by $u$, which sees its load increased by the amount $L_{uv}$ that was formerly supplied to them by device $v$; the increased load has to fit the failover capacity $B$).

We assume that each demand size $s_j$ fits on a pair of devices by itself, so $s_j \in [0, \min(1, B/2)]$. We are not allowed to reject demands, so the algorithm assigns arriving demands to the available devices until a demand cannot be scheduled, in which case the algorithm terminates. Our objective is to maximize the total size of all assigned demands (i.e., the utilization). We compare the algorithm against the optimal offline strategy that knows the demand sequence in advance (but still subject to the same no-rejection requirement). We use $\mathsf{OPT}$ to denote the total utilization of this optimal offline strategy.

This problem has similarities with several classical packing problems. For example, in the Multiple Knapsack problem (and related problems such as Generalized Assignment [18], AdWords [14], etc.) we are given a set of items each with a weight and size, and the goal is to select a subset of the items to pack in capacitated bins in order to maximize the total weight. However, one fundamental difference in our setting, besides the need to assign each demand to a pair of devices instead of a single device, is the failover constraint. Unlike in previously studied resource allocation problems, here the capacity constraints are not just determined by the total demand incident to a node, but rather they depend also on how the demands are arranged across its edges. See the next example.

▶ **Example 1.** Consider an instance with 4 power devices $a$, $b$, $c$, $d$ with failover capacity $B = 1$. There are 6 demands of size $\frac{1}{4}$ that arrive sequentially; suppose 4 demands have arrived so far. One possible assignment has placed 2 demands on each of the pairs $ab$ and $cd$ (see Figure 2a), while a different one may place each of the 4 demands on a different pair (see Figure 2b, in solid lines). While in the second option all remaining demands can be placed (dashed lines in Figure 2b), the first option cannot accommodate more demands due to the Failover capacity. To see this, suppose we assign another demand to device $a$, say. If device $b$ fails, then the total load on $a$ will become at least $\frac{5}{4}$ violating its Failover capacity.

The above example suggests that due to the Failover constraints we should "spread out" the demands by not putting too many demands on one edge, because if one of its endpoints fails then this edge can have a large contribution to the Failover constraint of the other endpoint. However, there is a danger in spreading out the demands too much and not leaving enough devices free.

▶ **Example 2.** Consider again the same 4 power devices $a$, $b$, $c$, $d$ with failover capacity $B = 1$. Now, there are 7 demands; the first 6 have a small size $\varepsilon > 0$ and the last demand has size 0.5. Assume the first 6 demands have arrived. A first option is to assign one demand of size $\epsilon$ per device pair (see Figure 3a). In this case, the remaining demand of size 0.5 cannot be placed, as the Failover capacities would be exceeded. The second option groups the first 6 demands on a single edge (see Figure 3b); in this case, all demands can be fulfilled by assigning the last demand on a disjoint edge (dashed edge of Figure 3b).

Taking these two examples together, there is a delicate balance between spreading demands out across edges to minimize their impact in failover scenarios and leaving enough devices open for future demands, as to not prematurely end up with an unassignable demand.

## 1.1 Our results

We first consider the FAILOVER problem in the worst-case and design a deterministic algorithm with competitive ratio $\approx \frac{1}{2}$. Since no deterministic algorithm can be better than $\frac{1}{2}$-competitive (see the full version of the paper for upper bound), this result is almost best possible.

▶ **Theorem 3.** *There is a deterministic poly-time online algorithm for* FAILOVER *in the worst-case model with competitive ratio* $\frac{1}{2} - O(\frac{1}{m^{1/3}})$,[2] *where* $m$ *is the number of devices.*

A $\frac{1}{2}$-competitive solution may, roughly speaking, underutilize by a factor of $\frac{1}{2}$ the available power; in the context of big cloud providers, this inefficiency translates to substantial capital expenses due to the extra data centers required to accommodate the demands. Since such losses are unavoidable in the worst-case model, we consider the FAILOVER problem in the *stochastic arrival model*. Here the demand sizes are drawn i.i.d. from an unknown distribution $\mu$ supported on $[0, \min(1, B/2)]$.

We show that in this stochastic model it is possible to obtain *sublinear additive regret*.

▶ **Theorem 4.** *For the* FAILOVER *problem in the stochastic arrival model, there is a poly-time algorithm that achieves utilization* $\mathsf{OPT} - O(\mathsf{OPT}^{5/6} \log \mathsf{OPT})$ *with probability* $1 - O(\frac{1}{m})$.

We remark that since $\mathsf{OPT}$ grows like $\Theta(\xi)$, where $\xi := \min\{n, m\}$ (see Lemma 9), this guarantee implies the multiplicative approximation $(1 - O(\log \xi \,/\, \xi^{1/6})) \cdot \mathsf{OPT}$. So as the number of demands and devices grow, the competitive ratio of this algorithm goes to 1.

As a subroutine of this algorithm, we need to solve the natural *offline minimization* variant of demand scheduling with failover: Given a collection of demands, minimize the number of devices needed to assign all demands satisfying the Nominal and Failover constraints. We also design an (offline) algorithm with sublinear additive regret for this problem (Section 4).

## 1.2 Technical Overview

We illustrate the main technical challenges in the FAILOVER problem in both the worst-case and stochastic models, as well as in the offline minimization subproblem needed for the latter.

**Online Worst-Case (Section 2).** The examples from Figure 2 and 3 show that the main difficulty is dealing with the trade-off between spreading out the demands, which allows for a better use of the failover budgets, and co-locating demands on fewer edges, keeping some edges free for future big demands.

[2] Throughout the paper we use $O(x)$ to mean "$\leq cst \cdot x$" for some constant $cst$ independent of $x$.

To effectively strike this balance and get near optimal guarantees, the main idea is to group demands based on their sizes using intervals $I_k$ and schedule each group separately on cliques of size $k$. That is, we will "open" a set of $k$ unused devices and assign the demands in $I_k$ only to the edges between these devices (opening new $k$-cliques as needed). Interestingly, we assign at most one demand per edge of the clique (other than for tiny demands, which are handled separately). This means the algorithm tries to co-locate demands in controlled *regions*, which allows for the right use of the failover budgets.

**Online Stochastic Arrivals (Section 3).** First, note that because demands are i.i.d. from a distribution with bounded support, the total utilization of the first $\ell$ demands grows as $\ell \cdot \mathbb{E}_{S\sim\mu}S$. Thus, it suffices to show that our algorithm "survives" for as many demand arrivals as possible without needing to reject one due to lack of space. Our approach is to try and assign prefixes of arrivals to the (approximately) minimum number of devices possible. This ensures that if our algorithm fails due to needing more than $m$ devices to feasibly assign another demand, then OPT will fail shorty after.

Our algorithm is based on a learn-and-pack framework, where we use knowledge of the first $\ell$ arrivals to compute a good template assignment for the next $\ell$ arrivals. To compute this template, we need a subroutine that (approximately) solves the offline minimization subproblem mentioned above. Concretely, we run the subroutine on the realized sizes of the first $\ell$ arrivals, which gives a possible assignment of these demands into, say $m'$ unused devices. We use the "slots" of this possible assignment as a template to assign the future $\ell$ demands by employing the *online monotone matching* process of Rhee-Talgrand [16]: For each future arrival, we assign it to a (carefully-chosen) open slot in the template that has a larger size – if we cannot find such an open slot, then we assign this demand to its own disjoint edge (using 2 more devices).

It is known that this matching process leaves $o(\ell)$ unmatched demands with high probability. Further, our offline minimization subroutine has sublinear additive regret, that is, it uses only $o(\ell)$ more devices than the optimal offline assignment. Since these losses are sublinear in the prefix size, it seems that by repeating this process together with doubling the prefix size we should obtain a final sublinear regret guarantee.

But there is still a major issue: This strategy uses *disjoint* sets of devices to fulfill the first $\ell$ demands and the next $\ell$ demands (for each doubling $\ell$). But this is possibly very wasteful: even using the optimal assignment for each of these $\ell$ demands separately may require many more devices (up to double) compared to reusing the leftover space from the first batch of $\ell$ demands for the next batch (i.e. assigning the batches to a common set of devices). Wasting a constant fraction of devices would lead to the unwanted constant-competitive loss. To overcome this, we show that $M_\ell$, the minimum number of devices to assign $\ell$ i.i.d. demands, is approximately linear in $\ell$ (Theorem 8), e.g. $M_\ell + M_\ell$ (assigning batches separately) is approximately $M_{2\ell}$ (assigning them together). This is a non-trivial task (another Rhee-Talagrand paper [15] is entirely devoted to doing this for the simpler Bin Packing problem). Perhaps surprisingly, our proof relies on our algorithm for the offline device minimization problem, which is LP-based. The crucial property is that the optimal LP value doubles if we duplicate the items on its input, which (with additional probabilistic arguments) translates into the additivity of $M_\ell$.

**Offline Minimization (Section 4).** Our algorithm for offline minimization of the number of devices needed to fulfill a set of demands is based on a configuration LP inspired by the classic Gomory-Gilmore LP for the Bin Packing problem. Consider a fixed assignment of

demands to some number of devices. We want to interpret each device as a configuration, which captures the arrangement of demands on this device's edges. Our LP will minimize the number of configurations needed in order to assign all demands.

There is a tension between two issues in this approach. First, the Failover constraint depends not only on the subset of demands on this device's edges, but also how they are arranged within these edges (because the most-loaded edge contributes to the Failover constraint). This suggests that a configuration should not only specify a subset of demands, but also have enough information about the edge assignment to control the most-loaded edge. Second, each demand must be assigned to a pair of devices rather than a single device, so our configurations are not "independent" of each other. Thus, we need to "match" configurations to ensure that a collection of configurations can be realized in an edge assignment. In summary, our configurations should be expressive enough to capture the Failover constraints, but also simple enough so that we can actually realize them in an actual assignment.

Our solution to this is to define a configuration to be a subcollection, say $C$, of demands satisfying $\sum_{s\in C} s \leq 1$ (the Nominal constraint) and $\sum_{s\in C} s + \max_{s\in C} s \leq B$ (a relaxed Failover constraint). Note that this notion of configuration does not capture the arrangement of the demands $C$ across a device's edges – we assume the best case that every demand is on its own edge to minimize their impact in failover scenarios. It is not clear that there even exists a near-optimal assignment that assigns at most one demand per edge, let alone that we can obtain one from the LP solution. However, our LP post-processing procedure will show that – by opening slightly more devices – we can match configurations of this form to realize them in a near-optimal assignment.

## 1.3 Related work

Despite a vast literature on assignment-type problems, none of the ones considered addresses the main issue of redundancy, modeled in the Failover problem. Arguably the Coupled Placement [11] problem is the closest to Failover. Given a bipartite graph with capacities at the nodes and a set of jobs, the goal is to assign a subset of the jobs to the edges of the graph to maximize the total value (each assigned job gives a value that also depends on its assigned edge), while respecting the capacity of the nodes (each assigned job consumes capacity from its edge's endpoints). [11] gives a $\frac{1}{15}$-approximation to the offline version of this problem (see also [1]). While this problem involves the allocation of jobs to a pair of nodes (albeit on a bipartite graph) and has the additional difficulty that the value and consumption of a job depends on which pair of nodes it is assigned, it does not have any Failover type constraints, a crucial component of our problem.

As already mentioned, several classic assignment problems are related to ours, such as the Multiple Knapsack [4], Generalized Assignment (GAP) [18], and AdWords problem [14, 7]. The latter is the closest to our problem: there are $m$ bins (i.e. advertisers) of different capacities, and jobs (i.e. keyword searches) that come one-by-one and need to be assigned to the bins; each assignment consumes some of the bin's capacity and incurs an equal amount of value (i.e. bid). The goal is maximize total value subject to bin capacities. Despite the similarities, this problem does not consider critical aspects of our problem, namely the need to assign a job/demand to a pair of bins/devices and the Failover constraints.

There is also a large literature on survivable network design problems, where failures in the network are explicitly considered [6], but the nature of the problems is quite different from our assignment problem, as the focus there is typically on routing flows.

Finally, a problem related to our device minimization problem, and from which we borrow some tools and techniques, is Bin Packing. Here jobs of different sizes need to be assigned to a minimum number of bins of size 1. Results are known in both offline [9] and online settings [2, 16]. In the online stochastic setting, [16] obtains an additive $+O(\sqrt{\mathsf{OPT}}\cdot\log^{3/4}\mathsf{OPT})$ sublinear approximation (see [5, 8, 13] for improvements under different assumptions).

## 2 FAILOVER Problem in the Online Worst-Case Model

In this section we consider the FAILOVER in the online worst-case model. We design an algorithm that achieves competitive ratio $\approx \frac{1}{2}$ in this setting (restated from the introduction).

▶ **Theorem 3.** *There is a deterministic poly-time online algorithm for* FAILOVER *in the worst-case model with competitive ratio* $\frac{1}{2} - O(\frac{1}{m^{1/3}})$,[3] *where* $m$ *is the number of devices.*

Recall that in the full version of the paper, we also show the almost matching upper bound of $\frac{1}{2}$ on such competitive ratio, and design another algorithm whose competitive ratio approaches 1 as the size of the largest demand goes to 0. To convey the main ideas more clearly, here we focus only on Theorem 3.

### 2.1 Algorithm

As suggested in the technical overview, our algorithm will group demands by size, and assign each group of demands to sub-cliques of an appropriate size. To make this precise, set in hindsight $L := m^{1/3}$ and for $k = 2, \dots, L-1$ define the interval

$$I_k := \left( \min\left\{\frac{1}{k}, \frac{B}{k+1}\right\}, \ \min\left\{\frac{1}{k-1}, \frac{B}{k}\right\}\right].$$

(Notice there is no $k = 1$, because the upper limit of $I_2$ is the max size of a demand.) This definition ensures that it is feasible to assign one demand of such size to each edge of a $k$-clique, as we argue in the next subsection. Also define the interval of small sizes

$$I_{\geq L} := \left[0, \min\left\{\frac{1}{L-1}, \frac{B}{L}\right\}\right].$$

The algorithm is then the following:

**Algorithm 1** **FailoverWostCase.**

1: When a demand arrives, determine the interval $I_k$ (or $I_{\geq L}$) that it belongs to based on its size.

2: If it belongs to an interval $I_k$ with $k \in \{2, \dots, L-1\}$, assign the demand to any "empty" edge (i.e. that has not received any demands) of a $k$-clique opened for $I_k$. If no such edge exists, then open a new $k$-clique for $I_k$.

3: Otherwise it belongs to $I_{\geq L}$, so assign it to an edge of one of its $L$-cliques using *first-fit* (so here we **can assign multiple demands to the same edge**) making sure that the total load on each edge is at most $\min\{\frac{1}{L-1}, \frac{B}{L}\}$. By first-fit we mean that the edges of the $I_{\geq L}$ cliques are arbitrarily ordered and the demand is assigned to the first possible edge. Open a new $L$-clique for $I_{\geq L}$ if need be.

4: If the demand cannot fit in the appropriate clique and it is not possible to open a new clique (i.e. there are not enough unused machines to form a clique of the desired size), then stop.

[3] Throughout the paper we use $O(x)$ to mean "$\leq cst \cdot x$" for some constant $cst$ independent of $x$.

## 2.2 Analysis

We first quickly verify that the assignment done by the algorithm is feasible, i.e. satisfies the Nominal and Failover constraints. Consider a node/machine $u$ on an $I_k$ clique opened by the algorithm (for machines in an $I_{\geq L}$ clique the argument is analogous). For the Nominal capacity constraint: Every demand assigned to $u$ is actually assigned to one of the $k-1$ edges in this clique incident on $u$; each such edge receives at most 1 demand from $I_k$ (and no other demands), so using the upper limit of this interval we see that $u$ receives total size at most $(k-1)\cdot\min\{\frac{1}{k-1}, \frac{B}{k}\} \leq 1$, so within its Nominal capacity. For the Failover capacity: in a failover scenario one of these $(k-1)$ demands has "both ends" assigned to $u$, so the total size it receives is now $k\cdot\min\{\frac{1}{k-1}, \frac{B}{k}\} \leq B$, so within the Failover capacity. Hence the algorithm produces a feasible assignment.

Now we show that the value obtained by the algorithm is at least $\left(\frac{1}{2} - O(\frac{1}{m^{1/3}})\right)\mathsf{OPT}$. The idea is to show that for (essentially) each clique opened by the algorithm, we get on average value at least $\approx \frac{1}{4}$ per vertex. Since each node has Nominal capacity 1 and each demand must be scheduled on two nodes, $\mathsf{OPT}$ can only get at most $\frac{1}{2}$ value from each node on average, showing that our algorithm is a $\approx \frac{1}{2}$-approximation. However, there are two exceptions where we may get less than $\approx \frac{1}{4}$ per vertex on average. The first is the last clique for each $I_k$, which may not be "fully used" (but by setting $L$ appropriately there are not too many nodes involved in this loss). More importantly, the second exception is the "big items" $I_2$, which may not allow us to get average value $\frac{1}{4}$ per node (e.g. when the failover is $B=1$, a demand of size $\frac{1}{3}+\varepsilon$ falls in the group $I_2$ and is put by itself on an edge, giving value $\frac{1}{6}+\frac{\varepsilon}{2} \ll \frac{1}{4}$ per node used). However, in this case we show that we can obtain a stronger upper bound for these demands for $\mathsf{OPT}$.

We now make this precise. Assume throughout that the algorithm has stopped before the end of the input (else it scheduled everything, so it is $\mathsf{OPT}$). We account for the value obtained on each type of clique separately.

**Cliques for $I_{\geq L}$.** We will use two observations: (i) when the algorithm opens a new $I_{\geq L}$ clique, every edge of the previous $I_{\geq L}$ cliques has some demand assigned to it, and (ii) across all $I_{\geq L}$ cliques, out of all edges with some demand assigned to them, at most one can have total size assigned to it less than $\alpha := \frac{1}{2}\min\left\{\frac{1}{L-1}, \frac{B}{L}\right\}$ (i.e. half of its "capacity").

Both observations stem from the first-fit strategy used to assign these demands. In particular, the algorithm will only open a new clique when a demand in $I_{\geq L}$ does not fit in the edges of the existing cliques, implying that all of these edges already have some demand assigned; this shows the first statement. For the second statement, by contradiction assume that at some point there are at least two edges on $I_{\geq L}$ cliques with total load less than $\alpha$. Then the first demand that was assigned to the last such edge has size less than $\alpha$. But this means that it could have been assigned to an earlier edge with load less than $\alpha$, contradicting the first-fit procedure.

Let $c_{\geq L}$ be the total number of $I_{\geq L}$ cliques that the algorithm opened, and $m_{\geq L} := c_{\geq L}\cdot L$ the number of nodes associated with those cliques. Combining the above two observations, at the end of the execution either: (i) every edge of the first $c_{\geq L}-1$ of these cliques has load at least $\alpha$ or; (ii) all but one edge in the first $c_{\geq L}-1$ cliques has load at least $\alpha$ and some edge of the last $c_{\geq L}$-th (e.g., the one that "opened" it) has load at least $\alpha$. In both cases, the total size of demands assigned by the algorithm to the edges of these cliques is at least

$$(c_{\geq L}-1)\cdot\binom{L}{2}\cdot\alpha = (c_{\geq L}-1)\cdot\frac{L}{4}\cdot\min\left\{1, \frac{(L-1)B}{L}\right\} \geq m_{\geq L}\cdot\frac{1}{4}\left(1-\frac{1}{L}\right) - O(L), \qquad (1)$$

yielding roughly average value $\frac{1}{4}$ from each node of these cliques, as claimed.

**Cliques for $I_k$, for $k \geq 3$.** Consider any clique for $I_k$ except the last one to be opened. All edges of this clique have some demand from $I_k$ assigned to it; given the lower limit for this interval, this means that the algorithm has assigned to each such clique total size at least

$$\binom{k}{2} \cdot \min\left\{\frac{1}{k}, \frac{B}{k+1}\right\} = \frac{k}{2} \cdot \min\left\{\frac{k-1}{k}, \frac{B(k-1)}{k+1}\right\}.$$

Since $k \geq 3$ and $B \geq 1$, the right-hand side is at least $\frac{k}{4}$. Letting again $c_k$ denote the number of cliques for $I_k$ that the algorithm opens and $m_k$ the corresponding number of nodes/machines, we can count the total value of all but the last $I_k$ clique and we see that the algorithm has assigned to them total size at least $(c_k - 1) \cdot \frac{k}{4} = m_k \cdot \frac{1}{4} - O(k)$.

**Cliques for $I_2$.** (Recall that there is no $k = 1$, so this is the last case to consider.) Given the lower limit of the interval $I_2$, each $I_2$ clique (which being a 2-clique is just an edge) has a demand of size at least $\min\{\frac{1}{2}, \frac{B}{3}\}$ assigned to it. So the algorithm assigns total size at least $m_2 \cdot \min\{\frac{1}{4}, \frac{B}{6}\}$ to these $I_2$ cliques, where $m_2$ is the number of nodes in these cliques.

**Total value of Alg.** Since we assumed that the algorithm stops at some point, it means that it could not open more cliques. This means that all but at most $L-1$ nodes belong to one such clique (the worst case is that it tried to open an $L$-clique but could not), so $m_{\geq L} + \sum_{k=3}^{L-1} m_k + m_2 \geq m - L$. Then adding the above estimates for the values obtained on each type of clique, we see that the algorithm gets total value at least

$$\begin{aligned}\mathsf{Alg} &\geq \frac{1}{4}\left(1 - \frac{1}{L}\right) \cdot \left(m - m_2 - L\right) - O(L^2) + m_2 \cdot \min\left\{\frac{1}{4}, \frac{B}{6}\right\} \\ &= \frac{1}{4} \cdot (m - m_2) + m_2 \cdot \min\left\{\frac{1}{4}, \frac{B}{6}\right\} - O(m^{2/3})\end{aligned}$$

where the last inequality uses the fact that $L = m^{1/3}$.

If the minimum in the last line is $\frac{1}{4}$, then we obtain $\mathsf{Alg} \geq \left(\frac{1}{2} - O(\frac{1}{m^{1/3}})\right)\mathsf{OPT}$ as desired (recall $\mathsf{OPT} \leq \frac{m}{2}$ since each machine has Nominal capacity 1 and each demand is assigned to two machines). **So assume this is not the case, namely $B < \frac{3}{2}$.** Under this assumption

$$\mathsf{Alg} \overset{with\ ass.}{\geq} \frac{1}{4} \cdot (m - m_2) + \frac{B}{6} \cdot m_2 - O(m^{2/3}) \tag{2}$$

**Value of OPT.** We analyze $\mathsf{OPT}$ again under the assumption $B < \frac{3}{2}$. The Failover constraints also ensure that in order to accommodate the demand from $I_2$ in case of failure, any node that receives a demand from $I_2$ can have total size assigned to it a most $B - \min\{\frac{1}{2}, \frac{B}{3}\} \overset{with\ ass.}{=} \frac{2B}{3}$, due to the assumption $B < \frac{3}{2}$. For all other nodes, $\mathsf{OPT}$ can assign at most size 1 per node due to the Nominal capacity constraint. Let $m_2^{\mathsf{OPT}}$ be the number of nodes where $\mathsf{OPT}$ schedules a demand from $I_2$. Again, since the size of each demand is counted towards the Nominal capacity of two nodes, the total size scheduled by $\mathsf{OPT}$ is

$$\mathsf{OPT} \leq \frac{1}{2}\left(m_2^{\mathsf{OPT}} \cdot \frac{2B}{3} + (m - m_2^{\mathsf{OPT}}) \cdot 1\right) = \frac{1}{2} \cdot (m - m_2^{\mathsf{OPT}}) + \frac{B}{3} \cdot m_2^{\mathsf{OPT}} \tag{3}$$

Notice that since every demand in $I_2$ has size $> \min\{\frac{1}{2}, \frac{B}{3}\} \geq \frac{1}{3}$, the Failover constraints ensure that in $\mathsf{OPT}$ (as well as in our algorithm) the demands from $I_2$ that are scheduled form a matching, i.e. no 2 such demands can share a node/machine. So $m_2^{\mathsf{OPT}}$ (resp. $m_2$) is

just twice the number of $I_2$ demands scheduled by OPT (resp. our algorithm). Moreover, both Alg and OPT schedule a prefix of the instance. Since OPT gets at least as much value as Alg, it means that it scheduled a prefix that is at least as long; in particular it schedules at least as many $I_2$ demands as our algorithm. Together these observations imply that that $m_2^{\mathsf{OPT}} \geq m_2$. Then given inequalities (2) and (3), under the assumption $B < \frac{3}{2}$ we obtain that $\mathsf{Alg} \geq \left(\frac{1}{2} - O(\frac{1}{m^{1/3}})\right)\mathsf{OPT}$ as desired. This concludes the proof of Theorem 3.

## 3 Sublinear Additive Regret in the Stochastic Model

We now consider FAILOVER in the online stochastic model, where, instead of being adversarial, the size $S_t$ of each demand now comes independently from an unknown distribution $\mu$ over $[0, \min\{1, \frac{B}{2}\}]$. Again, at time $t$ the algorithm observes the size $S_t$ of the current demand and irrevocably assigns it to two of the $m$ machines. We still use $\mathsf{OPT} = \mathsf{OPT}(S_1, \ldots, S_n)$ to denote the value of (sum of the sizes scheduled by) the optimal strategy, which is now a random quantity. Our main result is algorithm FailoverStochastic that achieves a sublinear additive loss compared to OPT in this model (restated from the introduction for convenience).

▶ **Theorem 4.** *For the* FAILOVER *problem in the stochastic arrival model, there is a poly-time algorithm that achieves utilization* $\mathsf{OPT} - O(\mathsf{OPT}^{5/6} \log \mathsf{OPT})$ *with probability* $1 - O(\frac{1}{m})$.

The algorithm relies on a learn-and-pack approach that uses previously seen items to compute a *template* for packing the next items. This process is performed in rounds. Each round starts by assigning the first demand of the round on a pair of (empty) machines. Then, we iteratively create a template based on the first $n_k := 2^k$ items of the round, which we use to schedule the next $n_k$ items. When the number of machines needed for the template (along with some slack) exceeds the number of available machines, the current round terminates and the next round begins. The next round maintains no knowledge of the previous demands; it only takes as input the number of empty machines $\tilde{m}$ which it is allowed to use.

Before describing the algorithm in more detail, an important question that arises is how to use the templates to schedule the future demands. A crucial component in this process are *monotone matchings*, which only match two values if the second is at least as big as the first.

▶ **Definition 5** (Monotone matching). *Given two sequences* $x_1, \ldots, x_n \in \mathbb{R}$ *and* $y_1, \ldots, y_n \in \mathbb{R}$, *a* monotone matching $\pi$ *from the* $x_t$*'s to the* $y_t$*'s is an injective function from a subset* $I \in \{1, \ldots, n\}$ *to* $\{1, \ldots, n\}$ *such that* $x_i \leq y_{\pi(i)}$ *for all* $i \in I$. *We say that* $x_i$ *is* matched *to* $y_{\pi(i)}$ *if* $i \in I$, *and* $x_i$ *is* unmatched *otherwise.*

Monotone matchings will allow us to match future demands ($x_i$'s) to the demands that are part of a template ($y_{\pi(i)}$'s) and put the former in the place of the latter (since $x_t \leq y_{\pi(i)}$). A surprising result of Rhee and Talagrand [16] is that if the two sequences are sampled i.i.d. from the same distribution, then almost all items can be matched, and moreover such a matching can be found online (see the paper for a more general result where the sequences may come form different distributions).

▶ **Theorem 6** (Monotone Matching Theorem [16]). *Suppose the random variables* $A_1, \ldots, A_n$ *and* $B_1, \ldots, B_n$ *are all sampled independently from a distribution* $\mu$. *Then there is a constant cst such that with probability at least* $1 - e^{-cst \cdot \log^{3/2} n}$ *there is a monotone matching* $\pi$ *of the* $A_i$*'s to the* $B_i$*'s where at most* $cst \cdot \sqrt{n} \log^{3/4} n$ *of the* $A_i$*'s are unmatched. Moreover, this matching can be computed even if the sequence* $A_1, \ldots, A_n$ *is revealed online.*

## 3.1 Algorithm

We are now ready to present the details of the FailoverStochastic algorithm.

**FailoverStochastic.** The algorithm just repeatedly calls the procedure OneRound below, passing to it the number of machines that are still available/unopened (e.g. initially it calls OneRound$(m)$); it does this for $\frac{\log m}{\log 4/3}$ rounds.

**OneRound($\tilde{m}$).** This procedure receives as input the number $\tilde{m}$ of machines that it is allowed to open. It is convenient to rename the demands and use $Y_t$ to denote the $t$-th demand seen by OneRound (which are still sampled i.i.d. from $\mu$). Similar to the work of Rhee and Talagrand [16], this algorithm works in phases: As mentioned earlier, each phase $k$ sees the previous $n_k = 2^k$ items and creates a template based on them, which will then be used to schedule the next $n_k$ items. To create this template, we define the offline problem OffMinFailover of minimizing the number of machines that are required to schedule these $n_k$ items. To solve this problem, we design an approximation algorithm OffMinFailoverAlg achieving a sublinear approximation guarantee (more on this soon). Specifically, let $\overline{\mathsf{OPT}}_{mach}(x_1, \ldots, x_n)$ be the number of machines that OffMinFailoverAlg (with $\varepsilon = 1/n_k^{1/6}$) uses to schedule the demands $x_1, \ldots, x_n$. OneRound is then as follows:

**Algorithm 2** OneRound**:** Given a number of available machines $\tilde{m}$.

1: Assign the first demand $Y_1$ to an empty edge by itself, opening 2 machines.

2: For phases $k = 0, 1, 2, \ldots$

**(a)** See the first $n_k$ items $Y_1, \ldots, Y_{n_k}$. Run the algorithm OffMinFailoverAlg from Section 4 (with $\varepsilon = 1/n_k^{1/6}$) to find a solution for them that uses $\overline{\mathsf{OPT}}_{mach}(Y_1, \ldots, Y_{n_k})$ machines; let $templ(t)$ denote the pair of machines that $Y_t$ is assigned to. This solution is our template.

**(b)** STOP if

$$\#\{\text{already open machines}\} + \underbrace{\overline{\mathsf{OPT}}_{mach}(Y_1, \ldots, Y_{n_k})}_{\text{machines from template}} + \underbrace{cst_1 \cdot \sqrt{n_k} \log^{3/4} n_k}_{\text{predicted unmatched demand}} + 2m^{5/6} > \tilde{m}.$$

**(c)** Else, open a clique of $\overline{\mathsf{OPT}}_{mach}(Y_1, \ldots, Y_{n_k})$ machines. Upon the arrival of each of the next $n_k$ demands $Y_{n_k+1}, \ldots, Y_{2n_k}$, assign them to machines based on the template. More precisely, find the Rhee-Talagrand monotone matching $\pi$ guaranteed by Theorem 6 from the new to the old demands (as the new ones arrive online). Schedule each matched new demand $Y_t$ to the pair of machines that $Y_{\pi(t)}$ occupied in the template, namely the machine pair $templ(\pi(t))$. For each unmatched new demand, schedule it on an edge by itself (opening two more machines for each). If at any point the execution tries to open more than $\tilde{m}$ machines, declare FAIL.

## 3.2 Analysis

We next discuss the main ideas for the analysis of the algorithm FailoverStochastic, leading to the proof of Theorem 4. We assume throughout that $m$ is at least a sufficiently large constant, else the success probability $1 - O(\frac{1}{m})$ trivially holds. Due to space constraints, we mostly state and discuss at a high-level the main components of the proof and show how they imply Theorem 4, deferring details to the full version of the paper.

First, we need to develop two important and complex components. Let $\mathsf{OPT}_{mach}(J)$ denote the minimum number of devices needed to assign all demands from a set of demands $J$, satisfying the Nominal and Failover constraints.

**First component.** The first component is the aforementioned algorithm OffMinFailoverAlg that is called within OneRound. It relies on a novel configuration LP, ($\mathsf{LP}_{mach}$), and a post-processing algorithm to realize a rounded LP solution as a feasible assignment. It has the following guarantee:

▶ **Theorem 7.** *There exists a poly-time algorithm,* OffMinFailoverAlg*, that given $\varepsilon \in (0,1)$, finds a solution for* OffMinFailover *with at most $\big(1+O(\varepsilon)\big)\mathsf{LP}_{mach} + O(\frac{1}{\varepsilon^5}) \le \big(1+O(\varepsilon)\big)\mathsf{OPT}_{mach} + O(\frac{1}{\varepsilon^5})$ machines.*

Choosing $\varepsilon$ appropriately, we can create a template using at most $\mathbb{E}\,\mathsf{OPT}_{mach}(Y_1,\ldots,Y_{n_k}) + o(n_k)$ devices in expectation for the next $n_k$ arrivals. This result is proved in Section 4.

**Second component.** Recall from the technical overview that a worrisome aspect of FailoverStochastic is that each call to OneRound does not re-use machines from previous rounds. To show that this is not too wasteful, we prove that $\mathbb{E}\,\mathsf{OPT}_{mach}(X_1,\ldots,X_T)$ is approximately linear in $T$. We do so by giving a quantitative convergence theorem of $\mathbb{E}\,\mathsf{OPT}_{mach}(X_1,\ldots,X_T)$ to $T\cdot c(\mu)$, where $c(\mu)$ is a constant that characterizes the "average number of devices needed per demand." Furthermore, we use the bounded-differences inequality [3] to show that the number of machines $\mathsf{OPT}_{mach}(X_1,\ldots,X_T)$ is concentrated around this mean. That is, in the full version of the paper we show the following:

▶ **Theorem 8.** *Let $\mu$ be a distribution supported on $[0,\min\{1,\frac{B}{2}\}]$. Then there exists a scalar $c(\mu)$ such that for every $T\in\mathbb{N}$ and $\lambda>0$, we have*

$$\mathsf{OPT}_{mach}(X_1,\ldots,X_T) \;\in\; T\cdot c(\mu) \pm O(T^{5/6}) \pm \lambda\sqrt{T}$$

*with probability at least $1-2e^{-\frac{\lambda^2}{2}}$, where $X_1,\ldots,X_T$ are i.i.d. samples from $\mu$.*

Thus splitting the first $2n_k$ demands into two rounds of $n_k$ demands each costs us only an extra $o(n_k)$ devices.

With those two results in hand, the core of the analysis is that OneRound gets good value density, i.e., the ratio of value over number of machines $m$. We use $\mathbb{E}S_0$ to denote the expected value of the size of a demand (which is the same as $\mathbb{E}S_t$ for any $t$).

Specifically, according to Theorem 8, there is a scalar $c(\mu)$ such that OPT is able to fit roughly $\frac{1}{c(\mu)}$ demands per machine. Each such demand gives value roughly $\mathbb{E}S_0$; so the intuition is that the best possible density value/machine should be around $\frac{\mathbb{E}S_0}{c(\mu)}$. We first make this formal in the next lemma.

▶ **Lemma 9.** *With probability at least $1-\frac{2}{m^2}$ we have*

$$\mathsf{OPT} \le m\cdot\frac{\mathbb{E}S_0}{c(\mu)} + O(m^{5/6}) \quad \textit{and} \quad \mathsf{OPT} \ge \min\left\{ n\cdot\mathbb{E}S_0 - \sqrt{n\log m}\;,\; m\cdot\frac{\mathbb{E}S_0}{c(\mu)} - O(m^{5/6}) \right\}.$$

Crucially, the next lemma says that OneRound almost achieves this density.

▶ **Lemma 10.** *Let Open be the number of machines opened by* OneRound$(\tilde{m})$ *(which is a random variable). Then with probability at least $1-\frac{1}{m^2}$, the total value of the demands scheduled by* OneRound$(\tilde{m})$ *is at least*

$$\textit{value of } \mathsf{OneRound}(\tilde{m}) \ge \frac{\mathbb{E}S_0}{c(\mu)}\cdot \mathit{Open} - O(m^{5/6}).$$

Given this lemma, we see that the total value of the FailoverStochastic algorithm (which repeatedly calls OneRound) is approximately $\frac{\mathbb{E}S_0}{c(\mu)}$ times the total machines opened during the execution. By showing that the number of machines FailoverStochastic opens is $\approx m$, we then almost match the upper bound on OPT from Lemma 9.

▶ **Lemma 11.** *There is a constant $cst_5$ such that with probability $1-O(\frac{1}{m})$, FailoverStochastic opens at least $m - 5cst_5 \cdot m^{5/6}$ machines.*

These lemmas quickly lead to the proof of Theorem 4.

**Proof of Theorem 4.** Let $L := \frac{\log m}{\log 4/3}$ denote the number of calls to OneRound that FailoverStochastic makes, and let $val_i$ and $Open_i$ be the value obtained and number of machines opened by the $i$-th call. Employing Lemma 10 on these $L$ calls, we have that with probability at least $1 - \frac{L}{m^2}$ the total value of FailoverStochastic is

$$\text{algo value} \;=\; val_1 + \ldots + val_L \;\geq\; \frac{\mathbb{E}S_0}{c(\mu)} \cdot \sum_{i \leq L} Open_i \;-\; O(m^{5/6} \log m).$$

Moreover, from Lemma 11, with probability at least $1 - O(\frac{1}{m})$ the total number of machines open $\sum_{i \leq L} Open_i$ is at least $m - 5cst_5 \cdot m^{5/6}$, in which case we get

$$\text{algo value} \;\geq\; m \cdot \frac{\mathbb{E}S_0}{c(\mu)} \;-\; O(m^{5/6} \log m). \tag{4}$$

Furthermore, from Lemma 9 we have that $\mathsf{OPT} \leq m \cdot \frac{\mathbb{E}S_0}{c(\mu)} + O(m^{5/6})$ with probability at least $1 - \frac{2}{m^2}$. So by taking a union bound and combining this with the above lower bound on the algorithm's value, we get that with probability $1 - O(\frac{1}{m})$

$$\text{algo value} \;\geq\; \mathsf{OPT} \;-\; O(m^{5/6} \log m).$$

Since (4) also implies that $\mathsf{OPT} \geq \Omega(m)$, the previous bound is at least $\mathsf{OPT} - O(\mathsf{OPT}^{5/6} \log \mathsf{OPT})$. This concludes the proof of Theorem 4. ◀

We conclude this section by proving the lower bound on the value density of OneRound from Lemma 10. We defer the proofs of Lemma 9 and 11 to the full version of the paper.

### 3.2.1 Proof of Lemma 10

First, we control in high-probability the number of phases that OneRound($\tilde{m}$) executes before stopping or failing; this is important to avoid dependencies on the total number of demands $n$ in the instance, which can be arbitrarily bigger than the scale of the effective instance.

▷ Claim 12. With probability $1 - \frac{1}{m^3}$, the number of phases within OneRound is at most

$$\bar{k} := \log\left(\frac{\tilde{m}}{c(\mu)} + O(\tilde{m}^{5/6}) + 3\log^{\frac{3}{2}} m\right). \tag{5}$$

Proof. Recall that the demand sizes $Y_1, Y_2, \ldots$ that OneRound sees are still i.i.d. samples from the original distribution $\mu$. Using Theorem 8, it is not hard to show that with probability at least $1 - \frac{1}{m^3}$ OneRound can schedule at most $\frac{\tilde{m}}{c(\mu)} + O(\tilde{m}^{5/6}) + 3\log^{\frac{3}{2}} m$ many of these demands (for intuition, Theorem 8 indicates that even OPT requires more than $\tilde{m}$ machines to schedule these many demands). Since this quantity is exactly $n_{\bar{k}}$, OneRound cannot complete phase $\bar{k}$ (there are $2n_{\bar{k}}$ demands by the end of it) and the claim holds. ◁

Next, we need to bound how many machines are opened by OneRound, which in particular affects the probability of it failing. For a phase $k$, let $M_k := \overline{\mathsf{OPT}}_{mach}(Y_1, \ldots, Y_{n_k})$ denote the number of machines in the template solution, and let $U_k$ be the number of additional machines that had to be open to accommodate the unmatched demands among $Y_{n_k+1}, \ldots, Y_{2n_k}$, namely twice the number of unmatched items. Notice that these quantities are well defined even for phases that the algorithm did not execute. The quantity $M_k + U_k$ is then the number machines that the algorithm OneRound opens in phase $k$ (if it executes it). We have the following bounds for the number of machines open, at least for a phase $k$ where the number of items $n_k$ is sufficiently large (but still sublinear in $m$).

▷ Claim 13. Let $k_0 := (\frac{2}{cst_2} \log m)^{2/3}$ for a sufficiently small constant $cst_2$. Then there is a constant $cst_1$ such that:

1. For $k \geq k_0$, we have $M_k \in n_k \cdot c(\mu) \pm cst_1 \cdot n_k^{5/6}$ with probability $\geq 1 - \frac{1}{m^3}$
2. For $k \geq k_0$, we have $U_k \leq cst_1 \cdot \sqrt{n_k} \log^{3/4} n_k$ with probability $\geq 1 - \frac{1}{m^3}$
3. $n_{k_0} \leq m^{5/6}$.

Proof. Consider a phase $k \geq k_0$. Since the demand sizes $Y_1, \ldots, Y_{2n_k}$ seen in this phase are i.i.d. samples from the original distribution $\mu$, we can bound the minimum number of machines $\mathsf{OPT}_{mach}(Y_1, \ldots, Y_{n_k})$ (using Theorem 8 with $\lambda = n_k^{1/3}$) as

$$\mathsf{OPT}_{mach}(Y_1, \ldots, Y_{n_k}) \in n_k \cdot c(\mu) \pm O(n_k^{5/6})$$

with probability at least $1 - 2e^{-\frac{n_k^{2/3}}{2}}$. Moreover, employing the guarantee of the algorithm OffMinFailoverAlg used to build the template (Theorem 7 with $\varepsilon = 1/n_k^{1/6}$), we get that $M_k$ is in the range $n_k \cdot c(\mu) \pm cst_1 \cdot n_k^{5/6}$ with probability at least $1 - 2\exp(-\frac{n_k^{2/3}}{2})$ for some constant $cst_1$. But since $n_k = 2^k \geq 2^{k_0}$, a quick calculation shows that this probability is at least $1 - \frac{1}{m^3}$, proving the first item of the claim.

To control $U_k$, we can use the Monotone Matching Theorem (Theorem 6) with the first sequence of sizes being the demands from the template, i.e., $(B_1, \ldots, B_{n_k}) = (Y_1, \ldots, Y_{n_k})$, and the second one being the demands that we attempted to match to them, namely $(A_1, \ldots, A_{n_k}) = (Y_{n_k+1}, \ldots, Y_{2n_k})$ to obtain that the number of unmatched demands is at most $cst \cdot \sqrt{n_k} \log^{3/4} n_k$ with probability at least $1 - e^{-cst \cdot \log^{3/2} n_k}$, and hence with this probability $U_k \leq 2cst \cdot \sqrt{n_k} \log^{3/4} n_k$. Again because $k \geq k_0$, we get that this probability is at least $1 - \frac{1}{m^3}$, proving Item 2 of the claim (by taking $cst_1 \geq 2cst$ we can just replace the latter by the former).

The last item $n_{k_0} \leq m^{5/6}$ of the claim can be directly verified using the fact that we assumed $m$ is at least a sufficiently large constant. ◁

Recall that OneRound only fails when the number of machines $M_k + U_k$ actually opened in a phase is bigger than it "predicted" in Line 2.b, and this prediction is exactly $M_k$ plus the upper bound $U_k$ from Claim 13 plus a slack of $2m^{5/6}$. By considering all phases, it is now easy to upper bound the probability that OneRound fails ($\bar{k}$ is defined in (5)).

▷ Claim 14. The probability that OneRound fails is at most $\frac{\bar{k}+1}{m^3}$.

Proof. Fix any phase $k$, and we claim that the probability that OneRound fails on this phase is at most $\frac{1}{m^3}$. If OneRound fails on phase $k$, then it did not STOP in Line 2.b, so

$$\#\,[\text{machines open before phase } k] + \overline{\mathsf{OPT}}_{mach}(Y_1, \ldots, Y_{n_k}) + cst_1 \cdot \sqrt{n_k} \log^{3/4} n_k + 2m^{5/6} \leq \tilde{m},$$

but it ran out of machines during phase $k$, namely

$$\#\,[\text{machines open before phase } k] + (M_k + U_k) > \tilde{m}.$$

Since $M_k = \overline{\mathsf{OPT}_{mach}}(Y_1, \ldots, Y_{n_k})$, these observations imply that $U_k > cst_1 \cdot \sqrt{n_k}\, \log^{3/4} n_k + 2m^{5/6}$. This is impossible if $n_k \leq m^{5/6}$, because the number of machines $U_k$ opened for the unmatched demands is at most twice the number $n_k$ of demands considered for the matching. So we must have $n_k > m^{5/6}$ (and so from Claim 13 $k \geq k_0$) and at least $U_k > cst_1 \cdot \sqrt{n_k}\, \log^{3/4} n_k$; but again by Claim 13 the latter happens with probability at most $\frac{1}{m^3}$. Thus, the probability that OneRound fails on phase $k$ is at most $\frac{1}{m^3}$.

Moreover, by Claim 12, with probability at least $1 - \frac{1}{m^3}$ OneRound has at most $\bar{k}$ phases. Then taking a union bound, we see that the event that OneRound has at most $\bar{k}$ phases and in all of them it does not fail holds with probability at least $1 - \frac{\bar{k}+1}{m^3}$; in particular, with at least this much probability the algorithm does not fail in its execution, proving the claim. ◁

We now finally lower bound the value that OneRound gets. Let $\tau$ be the (random) index of the last phase attempted by OneRound, namely where Line 2.c is executed. As long as it does not fail on the last phase $\tau$ (which by the previous claim happens with probability at least $1 - \frac{\bar{k}+1}{m^3}$) OneRound gets the value of all items up until this phase, that is

$$\text{value of OneRound} \geq Y_1 + \ldots + Y_{2n_\tau} \geq Y_1 + \ldots + Y_{2n_{\min\{\tau,\bar{k}\}}}. \tag{6}$$

Recall that the $Y_i$'s are independent and each has mean $\mathbb{E}S_0$. Then employing the Chernoff bound (Theorem 2.8 of [3]), for any fixed $t \leq n_{\bar{k}}$ we have that

$$Y_1 + \ldots + Y_t \geq t \cdot \mathbb{E}S_0 - \sqrt{n_{\bar{k}} \log(m^3 \cdot n_{\bar{k}})} \qquad \text{with probability at least } 1 - \frac{1}{m^3 \cdot n_{\bar{k}}}.$$

Then taking a union bound over (6), the previous displayed inequality for all $t \leq n_{\bar{k}}$, and over the event that OneRound has at most $\bar{k}$ phases (which holds with probability at least $1 - \frac{1}{m^3}$) we get that

$$\begin{aligned} \text{value of OneRound} &\geq 2n_{\min\{\tau,\bar{k}\}} \cdot \mathbb{E}S_0 - \sqrt{n_{\bar{k}} \log(m^3 \cdot n_{\bar{k}})} \\ &= 2n_\tau \cdot \mathbb{E}S_0 - \sqrt{n_{\bar{k}} \log(m^3 \cdot n_{\bar{k}})} \\ &\geq 2n_\tau \cdot \mathbb{E}S_0 - O(m^{5/6}) \qquad \text{with probability} \geq 1 - \frac{\bar{k}+3}{m^3}. \end{aligned} \tag{7}$$

To conclude the proof of Lemma 10 we just need to relate this quantity to the number of machines opened by OneRound. Let $Open_\ell$ be the number of machines opened until (including) phase $\ell$, and recall that $Open$ is the number of machines opened over all phases. Since the number of machines opened on phase $k$ is $M_k + U_k$ (plus two machines for the first demand $Y_1$), we have

$$Open_\ell = 2 + (M_1 + U_1) + \ldots + (M_\ell + U_\ell) \tag{8}$$

To upper bound the right-hand side, for the phases $k < k_0$ we just use the fact that $M_k + U_k \leq 2n_k + 2n_k = 4n_k$, since both in the template and for the unmatched demands we never open more than 2 machines per demand considered (and $n_k$ demands are considered in each part). For each phase $k = k_0, \ldots, \bar{k}$ we can use Claim 13 to upper bound $M_k + U_k$ with probability at least $1 - \frac{2}{m^3}$ by

$$M_k + U_k \leq n_k \cdot c(\mu) + cst_1 \cdot n_k^{5/6} + cst_1 \cdot \sqrt{n_k}\, \log^{3/4} n_k \leq n_k \cdot c(\mu) + cst_3 \cdot n_k^{5/6}$$

for some constant $cst_3$. Together these bounds give that with probability at least $1 - \frac{2\ell}{m^3}$

$$Open_\ell \leq 2 + \sum_{k<k_0} 4n_k + \sum_{k=k_0}^{\ell} \left( n_k \cdot c(\mu) + cst_3 \cdot n_k^{5/6} \right).$$

To further upper bound the first summation on the right-hand side, because of the exponential relationship $n_k = 2^k$, we have $\sum_{k<k_0} 4n_k \leq 8n_{k_0-1} \leq O(m^{5/6})$, the last inequality coming from Claim 13; for the second summation, we analogously have $\sum_{k=k_0}^{\ell} n_k \leq 2n_\ell$ and $\sum_{k=k_0}^{\ell} n_k^{5/6} \leq O(n_\ell^{5/6})$. Therefore,

$$Open_\ell \leq 2n_\ell \cdot c(\mu) + O(n_\ell^{5/6}) + O(m^{5/6}) \qquad \text{with probability at least } 1 - \frac{2\ell}{m^3}. \tag{9}$$

Finally, since by Claim 12 the number of phases $\tau$ performed by OneRound is at most $\bar{k}$ with probability at least $1 - \frac{1}{m^3}$, the total number of machines open can be upper bounded

$$Open \leq Open_{\min\{\tau,\bar{k}\}} \leq 2n_\tau \cdot c(\mu) + O(n_{\bar{k}}^{5/6}) + O(m^{5/6}) \leq 2n_\tau \cdot c(\mu) + O(m^{5/6})$$

with probability at least $1 - \frac{2\bar{k}+1}{m^3}$.

Finally, taking a union bound to combine this inequality with (7), we get that

$$\text{value of OneRound} \geq \frac{\mathbb{E}S_0}{c(\mu)} \cdot Open - O(m^{5/6})$$

with probability at least $1 - \frac{3\bar{k}+4}{m^3}$. Since $m$ is at least a sufficiently large constant, we have $m \geq 3\bar{k} + 4$, and the bound from the displayed inequality holds with probability at least $1 - \frac{1}{m^2}$. This finally concludes the proof of Lemma 10.

## 4 Offline Machine Minimization

In this section we consider the aforementioned (offline) minimization version of FAILOVER, which we call OFFMINFAILOVER: Given a failover capacity $B \geq 1$ and a collection of demands such that demand $j$ has size $s_j \in [0, \min\{1, \frac{B}{2}\}]$, we need to assign *all demands* to pairs of machines while satisfying the Nominal and Failover constraints, and the goal is to minimize the number of machines used. As before, we use $\mathsf{OPT}_{mach} = \mathsf{OPT}_{mach}(s_1, \ldots, s_n)$ to denote the cost of (i.e. number of machines in) the optimal solution.

The main result of this section (Theorem 7, restated) is an efficient algorithm with a sublinear additive regret for this problem (when $\varepsilon$ is set appropriately). We remark that a sublinear regret (compared to, say, a constant approximation) is necessary due to its use in Section 3. In fact, the algorithm compares against the stronger optimum of an LP relaxation for the problem (denoted by ($\mathsf{LP}_{mach}$), and defined below). We let $\mathsf{LP}_{mach}$ denote the optimal value of this LP.

▶ **Theorem 7.** *There exists a poly-time algorithm,* OffMinFailoverAlg*, that given $\varepsilon \in (0,1)$, finds a solution for* OFFMINFAILOVER *with at most $\big(1 + O(\varepsilon)\big)\mathsf{LP}_{mach} + O(\frac{1}{\varepsilon^5}) \leq \big(1 + O(\varepsilon)\big)\mathsf{OPT}_{mach} + O(\frac{1}{\varepsilon^5})$ machines.*

As hinted above, our algorithm is based on converting a solution of a configuration LP into a good assignment of demands to pairs of machines. But crucially, while the configuration of each machine controls the total size of demands serviced by it, it has no information how these demands are distributed over the "edges" incident to the machine, which is important for adequately handling the Failover constraints. The post-processing of the LP solution is the one in charge of creating a feasible (and low-cost) assignment from this limited control offered by the LP.

## 4.1 Configuration LP

Consider an assignment of the demands into some number of machines. We can view the collection of demands assigned to (the edges incident to) a given machine as a configuration. Precisely, we define a *configuration* $C$ to be a subset of the demands such that $\sum_{s\in C} s \leq 1$ and $\sum_{s\in C} s + \max_{s\in C} s \leq B$. Note that the first constraint is exactly the Nominal constraint, while the second is a relaxation of the Failover constraint, because the most-loaded edge incident on some machine can be larger than the single largest demand assigned to that machine. Thus, our notion of configuration does not take in to account how the demands are assigned to the respective edges incident on each machine.

To define our configuration LP, we suppose the input collection of demands is partitioned into $T$ *demand types* such that type $t$ consists of $n_t$-many demands each with size $s_t$. Thus each configuration $C$ can be represented by a number $n_t(C) \in \mathbb{N}$ of demands for each type $t$ such that $\sum_t n_t(C)\cdot s_t \leq 1$ and $\sum_t n_t(C)\cdot s_t + \max_{t|\, n_t(C)>0} s_t \leq B$. We are ready to define our configuration LP:

$$\begin{array}{lrll} \min & \sum_C x_C & & \\ s.t. & \sum_C n_t(C)\cdot x_C & \geq 2n_t & \forall t \\ & x & \geq 0 & \end{array} \qquad (\mathsf{LP}_{mach})$$

Note that the definition of $(\mathsf{LP}_{mach})$ depends on how the demands are partitioned into types. We show in the full version of the paper that the optimal value of $(\mathsf{LP}_{mach})$ does not depend on the particular type partition. Thus, throughout the analysis, we will use whichever type partition is convenient (unless a particular one is specified).

It is immediate that $(\mathsf{LP}_{mach})$ is a relaxation of OffMinFailover by taking the natural setting of the $x$-variables defined by a feasible assignment to machines: just let $x_C$ be the number of machines whose collection of demand sizes assigned to its edges are exactly those in $C$. In particular, we have that $\mathsf{LP}_{mach} \leq \mathsf{OPT}_{mach}$.

Although $(\mathsf{LP}_{mach})$ has exponentially many variables in general, we can approximately solve it via column generation similar to the standard bin packing configuration LP [10, 17] (proof in the full version of the paper).

▶ **Lemma 15.** *We can find in poly-time an extreme point solution of* $(\mathsf{LP}_{mach})$ *with objective value at most* $\mathsf{LP}_{mach} + 1$.

Further, observe that $(\mathsf{LP}_{mach})$ only has $T$ non-trivial constraints, so by the standard rank argument (see for example Lemma 2.1.3 of [12]) any extreme point solution of $(\mathsf{LP}_{mach})$ has at most $T$ non-zero variables. Thus, the next lemma follows immediately by rounding up all the fractional variable of an extreme point solution.

▶ **Lemma 16.** *Given an extreme point of* $(\mathsf{LP}_{mach})$ *with objective value Val, rounding up all fractional variables to the next largest integer gives an integral solution to* $(\mathsf{LP}_{mach})$ *with objective value at most* $Val + T$.

To summarize this section, we can efficiently obtain a collection of configurations, each corresponding to a machine, that "covers" all the demands. However, these configurations do not specify how to actually assign the demands to the edges incident on the corresponding machine. This is the goal of the next section.

## 4.2 Matching configurations

We say that a collection $\mathcal{C}$ of configurations is feasible if it comes from an integer solution for $(\mathsf{LP}_{mach})$, i.e. setting $x_C$ to be the number of times $C$ appears in $\mathcal{C}$ gives a feasible solution for $(\mathsf{LP}_{mach})$. Our goal in this section is to realize such collection by actually assigning demands to edges. The main challenge is satisfying the actual Failover constraints.

For simplicity assume $\sum_{C \in \mathcal{C}} n_t(C) = 2n_t$ for all types $t$, i.e. each demand appears on exactly 2 configurations (drop from the configurations what is extra). We can think of $\mathcal{C}$ (with, say, $N$ configurations) as a graph on $N$ nodes/machines, where node/machine $C \in \mathcal{C}$ has $n_t(C)$ "slots" for demands of type $t$. While this gives the right number of slots $2n_t$ to accommodate the demands of each type $t$, we still need to specify to which *edge* (pair of machines) each of the $n_t$ demands of type $t$ is assigned in a way that satisfies the Nominal and Failover constraints. (We can alternatively see this as a graph realization problem: each node $C$ as having a requirement $n_t(C)$ of "edges of type $t$" (which we call its $t$-*degree*) and we want to create edges of different types (i.e., assignment of demands to pairs of nodes) to satisfy these requirements while also satisfying the Nominal and Failover constraints.)

To see the challenge, consider a fixed node/configuration $C$. Regardless of how we assign demands to edges (as long as it is consistent with the slots of the configurations), the Nominal constraint of $C$ is satisfied: it will receive total size $\sum_t n_t(C) \cdot s_t = \sum_{s \in C} s$, which is at most 1 by definition of a configuration. This is not the case for the Failover constraint. This is again because the definition of configuration only gives us the relaxed version of the Failover constraint $\sum_{s \in C} s + \max_{s \in C} s \leq B$, In particular, the blue term only considers the largest demand assigned to machine $C$ instead of the most-loaded edge incident to $C$. However, these two quantities are the same *if we are able to assign at most one demand per edge*. (In the graph realization perspective, it means that it suffices to construct a *simple* graph with the desired $t$-degrees.) But it is not clear that such an assignment should even exist, let alone be found efficiently.

The main result of this section is that – by opening slightly more machines – we can find such an assignment that realizes any given collection of configurations satisfying both the Nominal and Failover constraints.

▶ **Theorem 17.** *Consider an instance of* OffMinFailover *with $T$ demand types. Given a collection $\mathcal{C}$ of $N$ configurations that is feasible for* ($\mathsf{LP}_{mach}$)*, we can find in poly-time a feasible solution for* OffMinFailover *that uses at most $N + O(DT)$ machines, where $D$ is the maximum number of demands in any configuration in $\mathcal{C}$.*

For that, we will need the following subroutine to assign some demands outside of their respective configurations. This result easily follows by opening disjoint edges as needed, and assigning demands arbitrarily to an already-opened edge is possible.

▶ **Lemma 18.** *There is a poly-time algorithm for* OffMinFailover *that uses at most $8 \cdot S + 2$ machines, where $S$ is the sum of the size of the demands in the instance.*

The algorithm guaranteed by Theorem 17 is the following. In order to simplify the notation, as before we assume without loss of generality that $\mathcal{C}$ has $\sum_{C \in \mathcal{C}} n_t(C) = 2n_t$ for all types $t$.

**Proof of Theorem 17.** It is clear that MatchConfigs runs in polynomial time, and assigns all demands to edges. Further, this assignment satisfies both the Nominal and Failover constraints, because we assign at most one demand per edge in Step 4 (see discussion in the beginning of this section), and Step 5 guarantees a feasible assignment for the remaining demands.

It remains to show that it opens $N + O(DT)$ machines. In particular, by Lemma 18 it suffices to show that the total size of all unassigned demands that reach Step 5 is $O(DT)$. When considering demand type $t$, there are two possibilities:

**Algorithm 3** MatchConfigs: Given a collection $\mathcal{C}$ of $N$ configurations.

1: Open $N$ machines – one corresponding to each configuration in $\mathcal{C}$.

2: Consider demand types in arbitrary order $t = 1, \dots, T$.

3: When considering demand type $t$, partition the collection $\mathcal{C}$ into two collections $\mathcal{L}_t$ and $\mathcal{R}_t$ such that their total $t$-degrees $\sum_{C\in\mathcal{L}_t} n_t(C)$ and $\sum_{C\in\mathcal{R}_t} n_t(C)$ differ by at most $D_t$, where $D_t$ is the maximum number of type $t$ demands in any configuration. (This can be achieved, e.g., by initializing $\mathcal{L}_t, \mathcal{R}_t = \emptyset$, and adding configurations one-by-one to the set with minimum total $t$-degree.)

4: Given this partition, as long as there exists a configuration $C \in \mathcal{L}_t$ that is currently assigned less than $n_t(C)$ demands of type $t$, we pick such a configuration and assign a demand of type $t$ to an arbitrary edge $(C, C')$ (for $C' \in \mathcal{R}_t$) that has not yet been assigned a demand of any type and such that $C'$ is currently assigned less than $n_t(C')$ demands of type $t$. If no such edge exists, then we stop and move on to the next demand type.

5: Once we are done considering all demand types, assign all the currently unassigned demands to new machines using Lemma 18.

**Case 1:** Step 4 assigns $n_t(C)$ type $t$ demands to each $C \in \mathcal{L}_t$. In this case it assigns $\sum_{C\in\mathcal{L}_t} n_t(C)$ type $t$ demands to edges between $\mathcal{L}_t$ and $\mathcal{R}_t$, while the total number of type $t$ demands is

$$n_t = \frac{1}{2}\left(\sum_{C\in\mathcal{L}_t} n_t(C) + \sum_{C\in\mathcal{R}_t} n_t(C)\right) \leq \sum_{C\in\mathcal{L}_t} n_t(C) + \frac{D_t}{2}, \tag{10}$$

where the inequality uses the fact that the $t$-degree of $\mathcal{R}_t$ is at most that of $\mathcal{L}_t$ plus $D_t$. Thus, at most $\frac{D_t}{2}$ demands of type $t$ remain unassigned and reach Step 5. The total size of these demands it at most $\frac{1}{2}$, since $D_t$ demands of type $t$ are in a valid configuration. Hence the total size of the unassigned demands of all types is at most $\frac{T}{2} \leq O(DT)$.

**Case 2:** Step 4 fails to assign $n_t(\bar{C})$ to a configuration $\bar{C} \in \mathcal{L}_t$. In this case, for each $C' \in \mathcal{R}_t$, either the edge $(\bar{C}, C')$ is already assigned some demand (call such $C'$ *blocked*) or $C'$ has already been assigned $n_t(C')$ demands of type $t$. But there are at most $D$ blocked $C'$'s, since the configuration $\bar{C}$ has at most $D$ slots to receive demands. Thus the total number of type-$t$ demands assigned is at least

$$\sum_{C'\in\mathcal{R}_t\setminus\text{blocked}} n_t(C') \geq \sum_{C'\in\mathcal{R}_t} n_t(C') - D\cdot\max_{C'\in\text{blocked}} n_t(C') \geq \sum_{C'\in\mathcal{R}_t} n_t(C') - D\cdot D_t.$$

Moreover, exchanging the roles of $\mathcal{L}_t$ and $\mathcal{R}_t$ in the argument from (10) we get that $\sum_{C'\in\mathcal{R}_t} n_t(C') \geq n_t - \frac{D_t}{2}$, and thus at least $n_t - D\cdot D_t - \frac{D_t}{2}$ demands of type $t$ are assigned by Step 4. Thus at most $O(D\cdot D_t)$ demands (hence total size $O(D)$) of this type remain unassigned and reach Step 5. This a total size of $O(DT)$, over all demand types, that reach the latter step, as desired. ◀

We summarize the main results of this section and the previous with the next theorem: By approximately solving ($\mathsf{LP}_{mach}$) (Lemma 15), rounding the solution (Lemma 16), and using the above algorithm to obtain an assignment of demands to edges (Theorem 17), we obtain the following.

▶ **Theorem 19.** *Consider an instance of* OffMinFailover *that has most $T$ demands types and where each configuration has at most $D$ demands. Then there is a poly-time algorithm that finds a feasible solution that uses at most $\mathsf{LP}_{mach} + O(DT)$ machines.*

To obtain our main result, Theorem 7, we need to modify the input instance to make $D$ and $T$ small enough. In the full version of this paper – by losing a multiplicative $(1 + O(\epsilon))$-factor – we show how to ensure that $D, T = poly(\frac{1}{\epsilon})$ by rounding demand sizes and handling the small demands separately. This concludes the proof of Theorem 7.

## References


1. Sara Ahmadian and Zachary Friggstad. Further approximations for demand matching: Matroid constraints and minor-closed graphs. In *44th International Colloquium on Automata, Languages, and Programming, ICALP*, 2017. doi:10.4230/LIPIcs.ICALP.2017.55.
2. János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A New and Improved Algorithm for Online Bin Packing. In *26th Annual European Symposium on Algorithms (ESA)*, 2018. doi:10.4230/LIPIcs.ESA.2018.5.
3. Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.
4. Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005. doi:10.1137/S0097539700382820.
5. Janos Csirik, David S Johnson, Claire Kenyon, James B Orlin, Peter W Shor, and Richard R Weber. On the sum-of-squares algorithm for bin packing. *Journal of the ACM (JACM)*, 53(1):1–65, 2006.
6. Anupam Gupta and Jochen Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011. doi:10.1016/j.sorms.2010.06.001.
7. Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Mathematics of Operations Research*, 41(4):1404–1431, 2016. doi:10.1287/moor.2016.0782.
8. Varun Gupta and Ana Radovanović. Interior-point-based online stochastic bin packing. *Operations Research*, 68(5):1474–1492, 2020.
9. Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, 2017.
10. Narendra Karmarkar and Richard M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd Annual Symposium on Foundations of Computer Science (SFCS 1982)*, pages 312–320, 1982. doi:10.1109/SFCS.1982.61.
11. Madhukar Korupolu, Adam Meyerson, Rajmohan Rajaraman, and Brian Tagiku. Coupled and k-sided placements: Generalizing generalized assignment. *Math. Program.*, 154(1–2):493–514, December 2015. doi:10.1007/s10107-015-0930-1.
12. Lap-Chi Lau, R. Ravi, and Mohit Singh. *Iterative Methods in Combinatorial Optimization*. Cambridge University Press, USA, 1st edition, 2011.
13. Shang Liu and Xiaocheng Li. Online bin packing with known T. *arXiv preprint arXiv:2112.03200*, 2021.
14. Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22–es, October 2007. doi:10.1145/1284320.1284321.
15. Wansoo T. Rhee and Michel Talagrand. Optimal bin packing with items of random sizes II. *SIAM Journal on Computing*, 18(1):139–151, 1989. doi:10.1137/0218009.
16. Wansoo T Rhee and Michel Talagrand. On-line bin packing of items of random sizes, II. *SIAM Journal on Computing*, 22(6):1251–1256, 1993.
17. Thomas Rothvoß. The entropy rounding method in approximation algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, 2012.
18. David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical programming*, 62(1):461–474, 1993.
19. Chaojie Zhang, Alok Gautam Kumbhare, Ioannis Manousakis, Deli Zhang, Pulkit A Misra, Rod Assis, Kyle Woolcock, Nithish Mahalingam, Brijesh Warrier, David Gauthier, et al. Flex: High-availability datacenters with zero reserved power. In *ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.