Robust and Probabilistic Failure-Aware Placement

Madhukar Korupolu
Google Research
mkar@google.com

Rajmohan Rajaraman*
Northeastern University
rraj@ccs.neu.edu

ABSTRACT

Motivated by the growing complexity and heterogeneity of modern data centers, and the prevalence of commodity component failures, this paper studies the *failure-aware placement problem* of placing tasks of a parallel job on machines in the data center with the goal of increasing availability. We consider two models of failures: adversarial and probabilistic. In the adversarial model, each node has a weight (higher weight implying higher reliability) and the adversary can remove any subset of nodes of total weight at most a given bound W and our goal is to find a placement that incurs the least disruption against such an adversary. In the probabilistic model, each node has a probability of failure and we need to find a placement that maximizes the probability that at least K out of N tasks survive at any time.

For adversarial failures, we first show that (i) the problems are in Σ_2, the second level of the polynomial hierarchy, (ii) a basic variant, that we call RobustFAP, is co-NP-hard, and (iii) an all-or-nothing version of RobustFAP is Σ_2-complete. We then give a PTAS for RobustFAP, a key ingredient of which is a solution that we design for a fractional version of RobustFAP. We then study fractional RobustFAP over hierarchies, denoted HierRobustFAP, and introduce a notion of *hierarchical max-min fairness* and a novel *Generalized Spreading* algorithm which is simultaneously optimal for all W. These generalize the classical notion of *max-min fairness* to work with nodes of differing capacities, differing reliability weights and hierarchical structures. Using randomized rounding, we extend this to give an algorithm for integral HierRobustFAP.

For the probabilistic version, we first give an algorithm that achieves an additive ε approximation in the failure probability for the single level version, called ProbFAP, while giving up a $(1+\varepsilon)$ multiplicative factor in the number of failures. We then extend the result to the hierarchical version, HierProbFAP, achieving an ε additive approximation in failure probability while giving up an $(L+\varepsilon)$ multiplicative factor in the number of failures, where L is the number of levels in the hierarchy.

*Supported in part by NSF grants CNS-1217981, CCF-1422715, and CCF-1535929, and a Google Research Award.

SPAA '16, July 11 - 13, 2016, Pacific Grove, CA, USA

ISBN 978-1-4503-4210-0/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2935764.2935802

1. Introduction

Modern datacenters are becoming increasingly complex and heterogeneous [3, 6, 11, 36]. Machines, racks, network switches, storage drives and other hardware equipment of different generations and capabilities often co-exist in the data center – leading to heterogeneity both in capacities as well as failure characteristics. The use of commodity hardware and the large scale of these data centers makes failures a common occurence and software systems need to plan and design for them [3, 38, 41].

In fact, the book on warehouse scale computing [3] devotes a whole chapter to failure and fault tolerance. Machines in the data center are often arranged in hierarchies [2, 6, 39] – with racks and top-of-rack (ToR) switches which are then inter-connected via network and power hierarchies. The components of these hierarchies are also prone to failure – due to hardware defects, planned and unplanned outages.

Several recent systems papers address the question of failures and the need to improve reliability in the presence of failures. [16, 34, 37] analyze failures in large disk drive and distributed installations. [17, 38, 42, 44] analyze failures of machines and network elements in large data centers, including Microsoft data centers over the course of a year. [6, 39, 40] study correlated failures in data centers due to failure of common nodes such as network and power domains. Decreased availability due to failures causes service downtimes and also hurts reputation for cloud service providers.

In recent work, [6, 39] addressed the problem of placing tasks of a parallel job (such as MapReduce [10], Spark [45]) among machines in a hierarchical data center so as to increase availability. However the treatment is informal and the proposed heuristics are evaluated experimentally without theoretical analysis.

Given this increasingly important context, in this paper, we initiate the formal definition of *failure-aware placement problem* and study its theoretical complexity and approximation algorithms. This models the common scenario in modern data centers where a job (such as MapReduce [10], Spark [45]) arrives with multiple identical tasks that need to be placed on machines in the data center. These can be either *batch jobs* that perform computations over large data sets and terminate when finished or *service jobs* that run indefinitely serving user requests [8, 36, 41]. The tasks of the job can be executed in parallel among machines in the

data center. Both the machines and the hardware domains that connect them (racks, switches, busducts, power nodes) are prone to failure. Jobs have availability requirements – such as a minimum number of tasks that need to be up and running at any time – to make quick progress and also to meet service level agreements. When a job arrives, the goal of the *failure aware placement system* is to allocate its tasks among machines so as to improve availability.

While we present our results in the context of tasks and machines, the concepts are general and apply to storage and other contexts as well. From a storage point of view, failure rates and disk drive characteristics have been studied in [34, 16, 37]. Error correcting codes are often used to store data among nodes in such a way that as long as k of n chunks are available, the original data can be reconstructed [18, 25, 35]. However, the related problem of how to place the n chunks among the available nodes so as to maximize the probability of k chunks surviving has received little attention [27]. A good failure-aware placement algorithm can increase data availability and thereby reduce the amount of redundant storage required.

1.1 Problem Setup

We formulate two classes of problems for failure-aware placement: *robust (adversarial)* and *probabilistic*. The first class of problems focuses on *robustness* by associating each domain with a weight representing its reliability (higher weight implies higher reliability), and allowing an adversary to remove (i.e., cause failures in) a subset of domains with total weight bounded by a given failure budget W. The goal is to determine a placement of N tasks among machines such that the maximum number of tasks that fail, taken over all adversary's choices of failing domains, is minimized. The second class of failure-aware problems considers probabilistic failures by associating a failure probability with each domain. The goal here is to determine a placement of N tasks among machines such that the probability that more than a given number F of tasks fail is minimized.

We study several variants of failure-aware placement under the above two classes. In addition to differences in the nature of failures (adversarial or probabilistic), these variants differ in the organization of the failure domains (single-level or hierarchical), the kind of adversary (fractional or integral), and the kind of placement (fractional or integral). In all of these variants, we focus on the case where all the tasks have the same size. Many of the modern data parallel batch processing systems (such as MapReduce [10], Spark [45]) have jobs with multiple (tens to thousands of) tasks of the same size running on different machines [36, 41]. They are launched using a config file which specifies the resource requirements for each task and the number of tasks (e.g., see [41], Section 2.3). User-facing jobs also often have multiple instances of the same size running on different machines, and they use a load balancer in front to distribute load evenly among them.

1.2 Overview of results and techniques

We now formally define the specific problems we study, and summarize our results and techniques. In each formulation, we have a set $\mathcal{F}$ of failure domains which includes machines and internal nodes in case of a hierarchy, and a job with N identical tasks that just arrived and needs to be placed. We have a capacity C_i for each domain, representing the number of tasks of the job that can be placed in the domain. We define a placement as a map $P : \mathcal{F} \to \mathbb{N}$ with $0 \le P(i) \le C_i$ for each domain i, $0 \le i \le |\mathcal{F}|$. For a subset D of disjoint domains, we define $P(D) = \sum_{i \in D} P(i)$.

Adversarial failures. We first consider the robust variants with adversarial failures.

Definition 1. *(*Placement against adversarial failures*) Given a nonnegative integer W, and for each i in $\mathcal{F}$ an integer weight w_i, the goal of* RobustFAP *is to place N tasks subject to capacity constraints so as to minimize the maximum number of tasks placed on any subset of domains with total weight at most W.*

For a given set S of domains, let $w(S)$ denote the sum of the weights of the domains in S. Formally, the goal of RobustFAP is to find a placement P that optimizes $\min_P \max_{S:w(S)\le W} \sum_{i \in S} P(i)$.

Hardness of RobustFAP **and related problems**. As formulated, it is unclear whether RobustFAP is in NP because the inner maximization is an NP-complete integer knapsack problem. Since the decision version of RobustFAP is of the form $\exists P \forall S$, where P refers to a placement, and S to the choice of an adversary, the problem is in Σ_2, the second level of the polynomial hierarchy [32, Chapter 17].

Theorem 1. *The decision version of* RobustFAP *is co-NP-hard. (Section 2)*

To better understand the complexity of RobustFAP, we consider a variant which we call AllOrNoneRobustFAP, which is identical to RobustFAP, except that we require that the placement P be *whole*: for any domain i, $P(i) = 0$ or $P(i) = C_i$; i.e., we either use the entire capacity of a domain or do not use it at all. AllOrNoneRobustFAP belongs to the class of bilevel knapsack problems, which have gathered interest in recent work [7, 14, 15].

Theorem 2. AllOrNoneRobustFAP *is Σ_2-complete, and NP-hard if input weights are unary. (Section 2)*

Algorithms for RobustFAP **and related variants**. As a step toward an approximation algorithm for RobustFAP, we first consider a fractional variant.

Definition 2 (Fractional RobustFAP). *The problem* FracRobustFAP *is identical to* RobustFAP*, except that the actions of both the placement algorithm and the adversary can be fractional. Formally, we seek a fractional placement $P : \mathcal{F} \to \mathbb{R}$ optimizing*
$\min_{P:P(\mathcal{F})\ge N} \max_{x \in [0,1]^n : x \cdot w \le W} x \cdot P$.

Note that the notion of robustness is different in the RobustFAP and FracRobustFAP problems. In RobustFAP, an adversary can take down any subset of domains with total weight bounded by W. In FracRobustFAP, an adversary can take down any fraction (say $x(i)$) of any domain i, as a result of which the number of failed tasks in the domain is $x(i)$ times the number of tasks allocated to i; the sole constraint is that the total fractional weight of the failed domains, $\sum_i x(i)w_i$, is bounded by W. We show that the classic notion of weighted max-min fair allocation [19, 28, 33] is simultaneously optimal for all weights W for FracRobustFAP.

THEOREM 3. *The weighted max-min fair allocation is simultaneously optimal for all W for* FRACROBUSTFAP. *(Section 3)*

Our main result here is a PTAS for ROBUSTFAP that combines the algorithm for FRACROBUSTFAP with a careful enumeration of placements according to how allocations are made to domains. We also present a PTAS for ALLORNONEROBUSTFAP, which places it alongside the bilevel knapsack problem DNEG [7] in the class of Σ_2-complete problems that surprisingly admit polynomial-time approximation schemes.

THEOREM 4. *There exist PTASes for* ROBUSTFAP *and* ALLORNONEROBUSTFAP. *(Section 3)*

Failure domain hierarchies. Thus far, we have assumed a flat organization of the failure domains. We next generalize the problem to failure domain hierarchies, in which we have a failure domain tree $\mathcal{T}$, with machines at the leaves and internal nodes of the tree representing failure domains. Each node v has a weight w_v representing its reliability, and a capacity C_v representing the maximum number of tasks that can placed in the subtree rooted at that node. Failure of an internal node v causes the whole subtree rooted at v and the tasks placed in that subtree to be unavailable.

DEFINITION 3. *(*FRACTIONAL PLACEMENT AGAINST ADVERSARIAL FAILURES IN HIERARCHIES*) The goal of* HIERFRACROBUSTFAP *is to compute a fractional placement of N tasks among machines in a hierarchy such that the capacity constraints are satisfied and the maximum number of tasks that can fail due to a fractional adversary with weight bound W is minimized.*

Our second main result is a new placement algorithm for HIERFRACROBUSTFAP, which computes a placement that is *simultaneously optimal* for all weight budget bounds W. Our algorithm, which we call *Generalized Spreading*, introduces the concept of *hierarchical max-min fairness* that applies to nodes of different capacities, different reliability weights and a hierarchical structure, and could be useful in other domains.

THEOREM 5. *The Generalized Spreading algorithm is optimal for all W for* HIERFRACROBUSTFAP. *(Section 4)*

We then consider the integral problem HIERROBUSTFAP, and show in Section 4, that a randomized rounding of the Generalized Spreading algorithm solution yields a solution for HIERROBUSTFAP with performance guarantees against oblivious adversaries.

Probabilistic failures in domain hierarchies. The next class of failure-aware placement problems we study is with probabilistic failures: where we model failures by having, for each domain, a probability of failure for that domain.

DEFINITION 4. *(*PLACEMENT UNDER PROBABILISTIC FAILURES*) In* PROBFAP*, we are given a set of N tasks, and a set of domains $\mathcal{F}$, and for each domain i, a probability p_i of the domain failing. The goal is to find a placement of tasks to machines subject to capacity constraints so as to minimize the probability that the number of tasks that fail exceeds a given failure bound of F.*

The probability that the number of tasks failing in a given placement exceeds F is given by

$$Pr_P(F) = \sum_{D \subseteq \mathcal{F}: P(D) \geq F} \prod_{i \in D} p_i \prod_{i \in \mathcal{F}-D} (1 - p_i).$$

where D is the set of domains that fail and the summation is over all possible subsets D of $\mathcal{F}$. The goal of PROBFAP then is to find a placement P that minimizes $Pr_P(F)$.

We then extend the problem formulation to hierarchies of domains. In HIERPROBFAP, we have a failure domain tree $\mathcal{T}$, each node v of which has a probability p_v of failing, and a capacity C_v representing the maximum number of tasks that can placed in the subtree rooted at that node. If a domain v fails, then the entire subtree under it is unavailable, implying that all the tasks placed in the domain fail. Given a bound F on the number of failed tasks, the goal of HIERPROBFAP is to allocate N tasks to the machines in the hierarchy subject to capacity constraints so as to minimize the probability that the number of failed tasks exceeds F.

Our main result here is a bicriteria approximation algorithm for PROBFAP and HIERPROBFAP. Our solution approach builds on the stochastic knapsack framework of Li and Yuan [24] to handle dependencies among knapsack items and hierarchical constraints on the allocations.

THEOREM 6. *There exists a polynomial-time algorithm for* HIERPROBFAP*, which computes, for any given constant $\varepsilon > 0$, a placement of N tasks such that the probability that the number of failed tasks is at most $(L+\varepsilon)F$ is at least $OPT - \varepsilon$ where OPT is the maximum achievable probability for the given failure bound F, and L is the number of levels in the hierarchy. (Section 5)*

The remainder of the paper is organized as follows. Section 1.3 reviews related work. Section 2 contains hardness results. Section 3 presents PTASes for ROBUSTFAP and ALLORNONEROBUSTFAP. Section 4 presents results for HIERFRACROBUSTFAP and HIERROBUSTFAP. Section 5 presents results for PROBFAP and HIERPROBFAP. Section 6 closes with open problems stemming from our work. Due to space constraints, we have omitted many of the proofs in this paper. We refer the reader to the full paper for missing proofs and details [21].

1.3 Related work

To the best of our knowledge, our specific formulations for failure-aware placement are new. We now review related results in the theory, operations research, and systems communities, and highlight the differences between our work and past work.

Max-min fairness. The notion of max-min fairness and its generalizations, including weighted max-min fairness and generalized processor sharing, have been used in networking [19, 28, 33]. The basic idea is to allocate resources among multiple players such that any attempt to further increase the allocation of one player necessarily results in the decrease in allocation of some other player with an equal or smaller allocation. Max-min fairness and its variants are used in process and network schedulers to allow multiple packet flows to share bandwidth fairly [12]. We show in Section 3.1 that the optimal placement for FRACROBUSTFAP is a weighted max-min fair one. For hierarchies the problem is more involved due to the corelations among the different levels of the hierarchy. To address this, we introduce the concept of hierarhical max-min fairness in the *Generalized Spreading* algorithm of Section 4.

Bilevel knapsack. The ALLORNONEROBUSTFAP problem has similarities to the bilevel knapsack problems which model decision makers at multiple levels [13, 43]. A bilevel knapsack problem has two players: a leader and a follower.

First, a leader fixes the values of their variables with the aim of optimizing their objective. Next, the follower reacts by setting their variables with the aim of optimizing their objective, under the constraints imposed by the leader. The variants studied earlier differ in the objective functions and variables under control of the leader and follower [14, 15, 26]. Dempe and Richter [14] study a variant in which the leader controls the capacity of the knapsack while the follower decides which items are packed in the knapsack. In the variant labeled DNeg [15], both players have a knapsack of their own, and the follower can only choose from items that the leader did not pick. Our AllOrNoneRobustFAP problem has similarity to DNeg [15], however the particular combination of constraints and objective functions are different and as a result the complexity results for one do not directly translate to the other. Our Σ_2-completeness and PTAS for AllOrNoneRobustFAP build on ideas from [7]. The RobustFAP problem has a different structure and uses additional ideas.

Stochastic knapsack and chance-constrained optimization. The placement problem under probabilistic failures resembles the category of chance-constrained optimization [29, 30], which involves constraints and objectives determined by probability of certain events. One consequence of having such chance constraints is that they define a non-convex set, making it difficult to apply general optimization techniques [29]. This challenge applies to ProbFAP and HierProbFAP as well. A number of algorithms have been developed for chance-constrained optimization [1, 4, 20, 30, 31], however, these do not yield any provable approximations for our problems of interest. Our solution approach for ProbFAP and HierProbFAP builds on the stochastic optimization framework of Li and Yuan [24] which uses a Poisson approximation technique [22]. Other recent work on stochastic knapsack and related problems includes [5, 23].

Finally, the recent work of [9] studies a distributed storage problem, which is a stochastic knapsack problem in which the unknown variables that are being determined are real-valued. Using structural results from the study of linear threshold functions, they give a unicriterion additive PTAS for their storage problem. In contrast, we focus on the discrete version of the related placement problem, and obtain a bicriteria additive PTAS for constant-depth network hierarchies with probabilistic failures.

2. Hardness of placement under adversarial failures

In this section, we establish the hardness of RobustFAP and AllOrNoneRobustFAP: Theorems 1 and 2.

2.1 Hardness of RobustFAP

Recall that, in the decision version of RobustFAP, our goal is to decide if there exists a placement such that for any subset S of domains with weight at most W, the total number of tasks allocated to the domains in S is at most F.

Proof of Theorem 1: Our co-NP-hardness proof is by a reduction from the NP-hard partition problem. Let I be an instance of the partition problem consisting of a set S of n integers $\{a_i : 1 \leq i \leq n\}$ such that $\sum_i a_i$ equals an even number $2W$. The goal of the partition problem is to determine if there exists a subset T of S such that the sum of the elements in T equals W.

Given I, we construct the following instance I' of RobustFAP. We have one domain for each element of S, with the capacity C_i and the reliability weight $w(i)$ of the ith domain both equal to a_i, for $1 \leq i \leq n$. We set (i) N, number of tasks we need to place, to $2W$, (ii) the adversary's weight budget to W, and (ii) F, the maximum number of allowed failures, to $W-1$.

We now show that I' has a placement with maximum failures $W-1$ if and only if I cannot be partitioned into two sets of equal size. First note that there is exactly one feasible placement of N tasks: place a_i tasks in domain i. Also for each subset X of domains, the number of tasks allocated as well as the weight are both equal to $\sum_{i \in X} a_i$. Therefore, a placement with maximum failures $W-1$ exists if and only if I cannot be partitioned into two sets of equal size thus establishing the co-NP-hardness.

2.2 Hardness of AllOrNoneRobustFAP

We now establish Theorem 2, the hardness of AllOrNoneRobustFAP. Clearly, AllOrNoneRobustFAP is in Σ_2. We show that it is Σ_2-hard via a reduction from the Σ_2-complete problem SubsetSumInterval. Our proof is similar to the proof given in [7], that establishes this result for DNEG, a different variant of bilevel knapsack which is incomparable to AllOrNoneRobustFAP. We refer the reader to [21] for the proof.

3. Polynomial-time approximation schemes for RobustFAP and AllOrNoneRobustFAP

We first consider the fractional variant FracRobustFAP and show in Section 3.1 that it can be solved optimally using weighted max-min fair allocations. We then present PTASes for RobustFAP and AllOrNoneRobustFAP, using the solution to FracRobustFAP for the former.

3.1 FracRobustFAP: Weighted max-min fair allocations

We begin with FracRobustFAP, as defined in Definition 2. Consider the *weighted max-min fair allocation*, a fractional placement that satisfies the following *fairness property*: if a domain i gets less than its "fair share" compared to another domain j, it is only because i is full. More formally, for any i and j, if $P(i)/w_i < P(j)/w_j$, then $P(i) = C_i$. Such a placement can be computed by the following algorithm.

1. Set $N' = N$, and for each domain i, $C'_i = C_i$.
2. Repeat the following until $N' = 0$:
   1. Let $D = \{i : C'_i > 0\}$ and $w(D) = \sum_{i \in D} w_i$
   2. For each domain i in D: place $a_i = \min\{C'_i, N'w_i/w(D)\}$ tasks on i and set $C'_i = C'_i - a_i$
   3. Set $N' = N' - \sum_i a_i$

Note that the above allocation is a fractional placement. We show that the weighted max-min fair allocation is in fact simultaneously optimal for all weight bounds W. This result is somewhat implicit in previous fair allocation works [28, 33]. In the full paper [21], we include detailed proofs for the sake of completeness, and to place the results in our failure-aware placement context.

LEMMA 1. *For any* FRACROBUSTFAP *instance with weight bound W, there exists an optimal allocation that satisfies max-min fairness.*

PROOF. For any placement P not satisfying the fairness property, we define the *low-ratio* as the $\min_i k_i/w_i$ over all i such that $k_i < C_i$, yet there exists a j for which $k_j/w_j > k_i/w_i$. Suppose, for the sake of contradiction, that there does not exist any optimal allocation satisfying the fairness property. Let P be an optimal allocation for the weight bound W, with the highest low-ratio.

Let i denote the arg-min for the low-ratio minimization, and let j be such that $k_i/w_i < k_j/w_j$ and yet $k_i < C_i$. Consider an alternative placement P' that is identical to P except that $P'(i) = P(i) + \varepsilon$ and $P'(j) = P(j) - \varepsilon$, for a sufficiently small $\varepsilon > 0$ satisfying $(P(i) + \varepsilon)/w_i < (P(j) - \varepsilon)/w_j$. Let s' be any knapsack of weight W and value v' built from P'. Consider the knapsack s that is identical to s' except that $s_i = s'_i - \varepsilon$ and $s_j = s'_j + \delta w_i(P(j) - \varepsilon)/(w_j(P(i)+\varepsilon))$, where $\delta = \min(s'_i, w_j(P(i)+\varepsilon)/(w_i(P(j)-\varepsilon))$. By construction, s has the same weight as s'. By our assumption on ε, $w_i(P(j)-\varepsilon)/(w_j(P(i)+\varepsilon))$ is at least δ, so the value of s is at least that of s'. Finally, either $\delta = s'_i$ or $\delta \geq \varepsilon$, and $\delta w_i(P(j)-\varepsilon)/(w_j(P(i)+\varepsilon)) \leq \varepsilon$, which together imply that s is a knapsack that can be built from P. Thus, for every knapsack of weight W and value v in P', we have a knapsack of weight W and value at least v in P, implying that P' is a placement that is also optimal and has a higher low-ratio than P. This contradicts our assumption that P had the higher low-ratio, completing the proof. □

LEMMA 2. *For any* FRACROBUSTFAP *instance, there is a unique weighted max-min fair allocation.*

Lemmas 1 and 2 establish the following Theorem.

THEOREM 7. *For a single level of failure domains with weight vector $(w_1, \ldots, w_m)$ and capacity vector $(C_1, \ldots, C_m)$, the weighted max-min fair allocation is simultaneously optimal for all weight bounds W for* FRACROBUSTFAP. □

3.2 ROBUSTFAP and ALLORNONEROBUSTFAP

For ROBUSTFAP one can easily construct instances to show that no (integral) placement can be simultaneously optimal for all weight bounds [21]. Given the "universal" optimality of the weighted max-min fair allocation for FRACROBUSTFAP, it is natural to explore its integer variants for ROBUSTFAP. One integer variant of ROBUSTFAP that we consider is the *rounded fair* placement: (i) compute the weighted max-min fair allocation P_f; (ii) let D be the set of domains that have a non-integer assignment, and $k < |D|$ equal to the integer $\sum_{i \in D}(P_f(i) - \lfloor P_f(i) \rfloor)$; (iii) set new placement P'_f as follows: $P'_f(i) = \lceil P_f(i) \rceil$ for i belonging to an arbitrary k-size subset of D, and $P'_f(i) = \lfloor P_f(i) \rfloor$ for all other i. We immediately observe that $P'_f(i) < P_f(i) + 1$ for all i. Though the rounded fair placement does not directly yield a PTAS, it plays a central role in our PTAS construction below.

We now describe our PTAS for ROBUSTFAP. Recall that the goal of ROBUSTFAP is to determine a placement P that minimizes the maximum of $P(T)$, over all subsets T of $\mathcal{F}$ with total weight at most W, such that (i) $P(\mathcal{F}) = N$, and (ii) for each $i \in \mathcal{F}$, $P(i) \leq C_i$. Consider the decision version of ROBUSTFAP: does there exist a placement such that for any subset T of domains whose weight is at most W, the sum of tasks assigned to domains in T is at most F? We give a poly-time algorithm that either determines that there does not exist any such placement, or returns a placement such that for any subset T of $\mathcal{F}$ whose weight is at most W, the sum of tasks placed in T is at most $(1 + \varepsilon)F$. We then obtain a PTAS for ROBUSTFAP via a binary search over F, invoking the above decision algorithm at each step.

Grouping of capacities and allocations. We constrain the placements to only allocate for each domain i a number of tasks that is either at most $\lceil 1/\varepsilon \rceil$, or the floor of an integer power of $(1 + \varepsilon)$, or C_i. We say that an allocation to a particular domain is *small* if it is at most $\lceil 1/\varepsilon \rceil$, *large* if it is not small and at least εF, and *medium* otherwise. We say that a placement P is monotonic if for any two domains i and j, if $w_i < w_j$ and $P(j) < C_j$, then $P(i) \leq P(j)$.

LEMMA 3. *There exists a monotonic placement, optimal for* ROBUSTFAP.

PROOF. For any placement P, we define the number of *inversions* of P as the number of pairs (i, j) such that $w_i < w_j$, $P(j) < C_j$, and $P(i) > P(j)$. The number of inversions lies between 0 and $n(n-1)/2$, where n is the number of domains. A monotonic placement has no inversions. Let P be an optimal placement that has the least number of inversions. If this number is 0, we are done. Otherwise, we will derive a new optimal placement that has fewer inversions, thus contradicting the assumption and establishing the lemma.

Let (i, j) be any inversion in P. We define a new placement P' which is identical to P except that $P'(j)$ equals $\min\{C_j, P(i)\}$, and $P'(i) = P(i) - (P'(j) - P(j))$. By construction, all capacity constraints are maintained. It is not hard to see, that the number of inversions of P' is less than that of P. We now show that P' is also optimal. Let T' denote the set of domains selected by the adversary against P'. Clearly, $w_{T'} \leq W$. If either T' has both i and j or has neither i nor j, then the adversary can select exactly the same set T' against P and achieve the same value. If T' has i but not j, then the adversary can select exactly the same set T' against P and achieve a value of $F' - P'(i) + P(i) > F'$. Finally, if T' has j but not i, then the adversary can select the set $T' - \{j\} \cup \{i\}$, with weight at most $W + w_i - w_j \leq W$, against P and achieve a value of $F' - P'(j) + P(i) \geq F'$. Thus, the value achieved by the adversary against P is at least that against P', thus completing the proof. □

Controlling large allocations. Since a large allocation is at least εF and at most F, it follows from our grouping of allocations that there are at most $g(\varepsilon) = \log_{1+\varepsilon}(1/\varepsilon) = O(\ln(1/\varepsilon)/\varepsilon)$ values to choose from for a large allocation. Since we are restricting to monotonic placements, there are at most $n^{g(\varepsilon)}$ different ways in which large allocations can be assigned to the domains, where n is the number of domains. Each way yields a partial placement: it specifies the exact allocation for all domains that have large allocations, while the small and medium allocations are undetermined. Given one such partial monotonic solution J, the adversary selects a subset of at most $\lceil 1/\varepsilon \rceil$ domains with large allocations, subject to the weight constraint. Thus, for a given J, the set $K(J)$ of choices for the adversary has size at most $n^{1/\varepsilon+1}$.

Enumerating small allocations. Restricting to monotonic placements allows us to bound the number of ways in

which we can make small allocations. Suppose we sort the domains in nondecreasing order of weights: i.e., $w_1 \leq \ldots \leq w_n$. Then, the small allocations are completely characterized by $t = \lceil 1/\varepsilon \rceil + 1$ indices $0 = i_0 \leq i_1 \leq \ldots \leq i_{t-1} \leq n$ such that every domain in the interval (i_{j-1}, i_j) has exactly $\min\{C_i, j\}$ tasks. Clearly, the number of ways we can determine these small allocations is $n^{O(1/\varepsilon)}$. In the remainder, we fix one choice S of the small allocations.

Determining medium allocations. Thus far, we have fixed a partial placement that assigns large allocations to some domains, and small allocations to others, and the adversary has selected a subset of the domains with large allocations. It remains to assign medium allocations. Let N' denote the number of tasks that remains to be assigned, and let $\mathcal{F}'$ denote the set of domains for which an allocation has not been determined. Note that the weight of each domain in $\mathcal{F}'$ is at most that of each domain that has already been assigned a large allocation. We need to consider the value achieved by an adversary that selects domains with medium and small allocations. Since all these domains have at most εF tasks, an adversary can achieve a value within an εF additive amount of the optimal by ordering the domains in increasing order of their density, and selecting a prefix of this sequence until no more domains can be added. We refer to this adversary as the *prefix adversary*.

Since we are restricting to medium allocations, we first place $\lceil 1/\varepsilon \rceil$ tasks in each domain in $\mathcal{F}'$, and set the capacity of each domain i in $\mathcal{F}'$ to be $\min\{\varepsilon F, C_i\}$. If the number of tasks already placed exceeds N', then the current partial assignment is not part of an $(1+\varepsilon)$-optimal solution. Otherwise, we assign the remaining tasks using the rounded fair placement, which assigns to each domain in $\mathcal{F}'$ at most one more task than the weighted max-min fair allocation. Since the number of tasks in any medium allocation is at least $\lceil 1/\varepsilon \rceil$, this additional task per domain with a medium allocation can cause only an extra ε fraction task failures, as compared with the weighted max-min fair allocation, which is optimal against all fractional adversaries. As we argued above, the prefix adversary is within an additive εF of an optimal fractional adversary. Consequently, for medium allocations, the value achieved by the rounded fair placement is within an additive εF of optimal against any adversary.

Putting it together. We enumerate over partial monotonic placements J in which the large allocations are fixed, an adversarial choice from the set $K(J)$ that determines which domains with large allocations to fail, a small allocation S, and run the rounded fair placement procedure for medium allocations. For each of these choices, we calculate the number of tasks failed by a weight W-bounded adversary, and determine if there exists a solution with at most $(1+\varepsilon)F$ failures. Given the polynomial size of J, $K(J)$, and the number of small allocations, we obtain a PTAS for RobustFAP.

PTAS for AllOrNoneRobustFAP. Recall that in AllOrNoneRobustFAP we have a set $\mathcal{F}$ of failure domains, a weight w_i and capacity C_i for each domain i, and an integer N. The goal of AllOrNoneRobustFAP is to determine a subset S of domains whose capacity is at least N such that the maximum, over any subset T of S whose weight is at most W, of the sum of capacities of domains in T is minimized. As in Section 3.2, we give a polynomial-time approximate algorithm for the decision problem and then give a PTAS for the optimization problem by doing a binary search. Due to space constraints, we defer the algorithm description and proofs to the full paper [21].

4. Hierarchical RobustFAP

We now present the *Generalized Spreading* algorithm for HierFracRobustFAP in Sections 4.1 and 4.2. We then use the Generalized Spreading algorithm to develop a randomized algorithm for HierRobustFAP in Section 4.3.

4.1 HierFracRobustFAP: Hierarchical max-min fairness

Recall that in HierFracRobustFAP we have a failure domain tree, with machines at the leaves and internal nodes representing failure domains. We now develop a hierarchical notion of max-min fairness. Given a fractional placement P and node v, let $P(v)$ denote the number of tasks placed by P in the subtree rooted at v. Given a placement P, for any leaf u, and any subtree T containing u, let $p_u(T, P)$ denote

$$\max_{\text{ancestor } v \text{ of } u \text{ in } T} P(v)/w_v.$$

We refer to $p_u(T, P)$ as the *vulnerability* of u in T. Note that this definition generalizes the analogous ratio used for defining fairness in the single-level case in Section 3.1. We say that a placement P is *hierarchical max-min fair* if for any pair of leaf nodes i and j, and any subtree T containing i and j, if $p_i(T, P) < p_j(T, P)$, then $P(i) = C_i$. By definition, any hierarchical max-min fair placement, when restricted to any subtree, is also hierarchical max-min fair.

In the following lemmas, we show that a hierarchical max-min fair optimal solution exists for every weight bound, and establish the existence of a unique hierarchical max-min fair placement for a given number of tasks; this solution is hence simultaneously optimal for all weight bounds. We defer the proofs of the lemmas to Section 4.4 after we present a polynomial-time algorithm to compute the hierarchical max-min fair placement in Section 4.2.

Lemma 4. *For any instance of* HierFracRobustFAP *and weight bound W, there exists a hierarchical max-min fair optimal solution.*

Lemma 5. *For any instance of* HierFracRobustFAP*, there exists a unique hierarchical max-min fair allocation.*

4.2 Generalized spreading for HierFracRobustFAP

We now present an algorithm for computing the hierarchical max-min fair allocation for HierFracRobustFAP. Our algorithm incrementally allocates tasks to the leaf nodes, while always maintaining the fairness property. *The intuition is best captured by thinking of directing fluid (tasks) into each of the leaf nodes in such way that the next infinitesimal unit goes to that node which has the least vulnerability.* Since vulnerability increases with the assignment of tasks, that node will soon have more vulnerability than some of the other nodes, in which case the next infinitesimal unit goes to another node. Repeating this process will yield the desired allocation. How do we convert the above intuition into a finite efficient procedure? The key challenge in implementing this algorithm is that the rate of increase for different leaf nodes is different, depending on their vulnerability, and not even constant for any given node.

An iteration of Generalized Spreading. Let P be a given current hierarchical max-min fair allocation, and we would like to add more tasks. We proceed in a bottom-up manner, and compute the following for each node j: (i) vector v_j giving the relative rates for the leaves in the subtree rooted at j by which we can increment their allocation (subject to capacity constraints); (ii) a scalar upper bound $u_j > 0$ indicating the (fractional) number of tasks that we can allocate using the relative rates v_j; and (iii) $maxr_j(x)$, a linear function over a scalar variable x yielding the maximum vulnerability in the subtree rooted at j. We show how to compute the above three quantities for a node j, given the same for the children of j.

Computation at the leaves. For any leaf l, we set v_l (which is simply a scalar) to be 1, u_l to be the remaining capacity of l, and $maxr_l(x)$ to be $(P(l) + x)/w_l$.

Recursive computation of v_j, u_j, and $maxr_j$. Fix an internal node j and let H_j denote the set of children of j. We first define function $r_i(x)$ to be the ratio $(P(j)+x)/w_j$. Fix a child i in H_j. If $r_j(0) > maxr_i(0)$, then this implies that the vulnerability at j exceeds that inside the subtree rooted at i. So we set $r_i^*(x) = r_j(x)$ and u_i^* to be the minimum of u_i and the smallest x_i such that $r_j(x_i) \leq maxr_i(x_i)$ (the point at which the ratios switch); otherwise, we set $r_i^*(x) = maxr_i(x)$ and u_i^* to be u_i.

Let i be an arbitrary child of j that is not full. Consider equating $r_i^*(x_i)$ with $r_l^*(x_l)$ for all non-full children l in H_j. By the fairness of the current placement and the definition of fairness from Section 4.1, if i and l are both not full, then $r_i^*(0) = r_l^*(0)$. Since these equations are linear, we can obtain a solution to the set of equations such that x_l for each l is a scalar multiple of x_i. We now set v_j to be the vector obtained by the putting together $x_l v_l$, over all children l of j, scaled so that the sum of the values equals 1. We next set u_j to be the minimum of the remaining capacity at j and the minimum, over all l, of $u_l^*/v_j(l)$, where $v_j(l)$ is the sum, over all leaf nodes m_l in the subtree rooted at l, of $(v_j)_{m_l}$. Finally, we set $maxr_j(x) = r_i^*(x)$.

Updating the placement. Using the above computation in a bottom-up manner, we compute for the root node t: v_t, u_t, and $maxr_t(x)$. We set P to be the new allocation given by the vector sum $P + v_t \min\{u_t, N - |P|\}$, where we use $|P|$ here to denote the number of tasks allocated in P.

We repeat the above process until the required number of tasks have been allocated. By design, the algorithm maintains the fairness property. In each iteration, either a node reaches capacity or a new node achieves the maximum vulnerability bound. This leads to a polynomial bound on the number of iterations. Combining this with Lemmas 5 and 4 yields the theorem.

THEOREM 8. *The generalized spreading algorithm computes in polynomial time a fractional placement that is simultaneously optimal for all W for* HIERFRACROBUSTFAP.

4.3 Randomized algorithm for HIERROBUSTFAP

Though the generalized spreading algorithm is inherently "continuous" it suggests natural algorithms for HIERROBUSTFAP. One discrete version of generalized spreading for HIERROBUSTFAP is to execute the same high-level incremental procedure, but allocating units of tasks in each iteration, as opposed to fractions. An analysis of this algorithm is an open problem. We now present a basic randomized rounding of generalized spreading that offers performance guarantees against oblivious adversaries.

Suppose P is the fractional placement returned by generalized spreading. We number the machines in the hierarchy in order (say left to right in appearance in the domain tree) so that all the machines belonging to a domain appear contiguously. With machine i, we associate the interval $I(i) = [\sum_{j=1}^{i-1} P(j), \sum_{j=1}^{i} P(j))$. We select a number ρ uniformly at random from $[0, 1]$. Consider the set $K = \{k + \rho : k \in \mathbf{Z}\}$. In our solution, the number of tasks we place in machine i equals the size of the intersection $I(i) \cap K$.

We now analyze the guarantee provided by the above randomized algorithm. For any machine i, the expected number of tasks placed on i equals the number of tasks $P(i)$ since it exactly equals the length of the interval $I(i)$. By linearity of expectation, the expected number of tasks placed on any set S of domains is $P(S)$. Therefore, for any weight bound W, if the optimal solution to HIERFRACROBUSTFAP achieves an objective of A_W^*, then for any subset S of domains of weight at most W, the expected number of tasks placed by the randomized algorithm on the domains in S is at most A_W^*. Furthermore, for any domain, the number of tasks placed on the domain is within one of that placed by the optimal fractional solution. This yields the following lemma.

LEMMA 6. *For any weight bound W, let A_W^* denote the objective value achieved by the optimal solution to* HIERFRACROBUSTFAP. *There exists a randomized algorithm for computing a placement for* HIERROBUSTFAP *such that for any W and any subset S of domains of total weight W, the expected number of tasks placed in S is at most A_W^*.*

4.4 Proofs for Lemmas 4 and 5

We start with characterizing an optimal adversary. The adversary's maximization problem for a given placement P is given in Equation 1 below, where $P(u)$ denotes the number of tasks placed by P in the subtree rooted at u.

$$\begin{aligned} A(P) &= \max \sum_u x_u P(u) \qquad (1)\\ \sum_u x_u w_u &\leq W \\ \sum_{u \in anc(v)} x_u &\leq 1 \text{ for every machine } v \\ x_u &\leq 1 \text{ for every domain } u \\ x_u &\geq 0 \text{ for every domain } u \end{aligned}$$

The following two lemmas provide insight into a solution to the adversary's maximization problem.

LEMMA 7. *Given any placement P, there exists an optimal solution x for the adversary in which there exists at most one node v such that x_v is not 0 and the sum of x_u, over all ancestors u of v, is in $(0, 1)$.*

PROOF. Let x be an optimal extreme point solution to the LP (an optimal vertex of the associated polytope). For any u, if x_u is 0 we remove x_u from the LP and u from the hierarchy; for any x_u that is 1, we add $P(u)$ as a constant into the LP objective and constraints, and remove the subtree rooted under u. We thus have an LP with number of variables equal to the number of nodes in the resulting forest,

and the number of constraints equal to one more than twice the number of nodes. Since the number of tight constraints in an extreme point solution equals the number of variables and none of the range constraints are tight, it follows that all but at most one of the sum constraints are tight, thus completing the proof of the desired claim. □

LEMMA 8. *Given any placement P, there exists an optimal solution x for the adversary in which there exists at most one node v such that x_v is in $(0,1)$.*

PROOF. By Lemma 7, there exists an optimal solution x for the adversary in which there is at most one node v such that x_v is not 0 and the sum of x_u, over all ancestors u of v, is in $(0,1)$. We refer to such as node v, if it exists, as an *outlier*. Let S be the set of nodes such that for each s in S, $x_s \in (0,1)$. Fix an s in S. If there exists an ancestor s' of s such that $x_{s'}$ is non-zero, then s' is an outlier; otherwise, s is an outlier. Given that there is at most one outlier, if $|S| > 1$, then there exists a unique node t in S such that (a) t is an outlier and an ancestor for all nodes in S, (b) no node in $S-t$ is an ancestor of any other node in $S-t$, and (c) for every s in $S-t$, x_s equals $1-x_t$.

We now argue that x can be transformed into another optimal solution x' for the adversary which satisfies the claim of the corollary. We consider three cases. In the first case $w_t < w(S-t)$, where $w(S-t)$ is the sum of the weights of the nodes in $S-t$. In this case, we set x'_t to be $x_t+\varepsilon$ and x'_s to be $x_s-\varepsilon'$, where $\varepsilon > \varepsilon' > 0$ is a real made sufficiently small. The weight constraint is maintained by the adversary by making ε' and ε sufficiently small, while the objective values increases, yielding a contradiction. In the second case, we have $w_t > w(S-t)$. Let $\varepsilon_t = \delta/w_t$ and $\varepsilon_s = \delta/w(S-t)$. If $P(t)/w_t \geq P(S-t)/w(S-t)$, then we set $x'_t = x_t+\varepsilon_t$ and $x'_s = x_s-\varepsilon_s$ for all $s \in S-t$. Otherwise, we set $x'_t = x_t-\varepsilon_t$ and $x'_s = x_s+\varepsilon_s$ for all $s \in S-t$. By the setting of ε_s and ε_t, the weight constraint for the adversary is satisfied. And in either of the two subcases, the objective value of the adversary improves, yielding a contradiction.

We are finally left with the case $w_t = w(S-t)$. In this case, we set $x_t = 1$ and $x_s = 0$ for all $s \in S-t$. Clearly, the weight constraint is satisfied, and the objective value of the adversary remains the same, establishing the desired claim of the corollary. □

LEMMA 9. *Given any placement P and any weight bound W, in every optimal adversarial solution x, for any two leaf nodes i and j and any subtree T containing i and j, if $p_i(T,P) > p_j(T,P)$, then $y_i(T,x) \geq y_j(T,x)$.*

PROOF. The proof is by contradiction. Let i and j be two leaves and T be a subtree containing them such that $p_i(T,P) > p_j(T,P)$ and $y_i(T,x) < y_j(T,x)$. Since $y_i(T,x) < y_j(T,x)$, there must exist an ancestor r of j in T such that $x_r > 0$. Let q denote the ancestor of i that is the argmax of $\max_{\text{ancestor } l \text{ of } i \text{ in } T} P(l)/w_l$. Then, we have $P(q)/w_q = p_i(T,P) > p_j(T,P) \geq P(r)/w_r$. We also have $x_q \leq y_i(T,x) < y_j(T,x) \leq 1$. Consider an alternative adversarial solution x' that is identical to x except that $x'_r = x_r - \varepsilon$, $x'_q = x_q + w_r/w_q$, for $\varepsilon \leq \min\{1-x_q, x_r\}$. The difference between the total value of the knapsack x' and that of x equals $\varepsilon w_r k_q/w_q - \varepsilon k_r$, which is positive since $k_q/w_q > k_r/w_r$. This contradicts the assumption that x is optimal, thus completing the proof of the lemma. □

Proof of Lemma 4: Let P be an optimal (fractional) placement for the weight bound W. Suppose, for the sake of contradiction, that there does not exist any optimal hierarchical max-min fair allocation. We consider any linear order $<$ of all of the subtrees of the hierarchy in which $T_1 < T_2$ whenever T_1 is a subtree of T_2. For any allocation P, let $\text{muf}(P)$ denote the minimum subtree, according to this linear order, such that P is not hierarchical max-min fair within that subtree. By our assumption, such a subtree exists. Let $\text{lvs}(P)$ denote the number of leaves i in $\text{muf}(P)$ such that $p_i(\text{muf}(P),P)$ equals $\max_{j \in \text{muf}(P)} p_j(\text{muf}(P),P)$.

Let P denote an optimal allocation with lexicographically minimum with respect to the triple

$$(\text{muf}(P), \max_{j \in \text{muf}(P)} p_j(\text{muf}(P),P), \text{lvs}(P)).$$

Let T equal $\text{muf}(P)$. We thus have two leaf nodes i and j of T such that j has the maximum vulnerability in T and $p_i(T,P) < p_j(T,P)$ and yet, $P(i) < C_i$. We argue now by contradiction that i and j belong to two different proper subtrees of T. If i and j belong to the same proper subtree, say T', of T, then since P is hierarchical max-min fair within T', $p_i(T',P) \geq p_j(T',P)$, which also implies that $p_i(T,P) \geq p_j(T,P)$ leading to a contradiction.

Let i and j belong to proper subtrees rooted at say u and v, respectively; we refer to these trees as T_u and T_v, respectively. By our assumption the allocations within both T_u and T_v are hierarchical max-min fair. Consider an alternative allocation P' that is identical to k except that an $\varepsilon > 0$ amount of tasks are added to the tree rooted at u, and an amount decreased from the tree rooted at v. We do this following the algorithm defined above; this is feasible since the allocation P, restricted to u (or v), is hierarchical max-min fair and hence produced by the algorithm (following our assumption about T being $\text{muf}(P)$). The amount is chosen sufficiently small such that $(P(u)+\varepsilon)/w_u < (P(v)+\varepsilon)/w_v$, and $P'(l) \leq C_l$ for all l in the tree rooted at u, and $P'(l) \geq 0$ for all l in the tree rooted at v. Clearly, P' is a valid allocation. Furthermore, since j has maximum vulnerability in P, P' has either a smaller maximum vulnerability than P or P' has fewer leaf nodes with the maximum vulnerability value. In either case, P' is lexicographically smaller with respect to the metric given above. We now derive a contradiction by arguing that P' is also an optimal allocation.

Let s' be an optimal knapsack of weight W and value v' built from P'. Note that j has the largest vulnerability in the tree rooted at v and i has the largest vulnerability in the tree rooted at u. Since j has higher vulnerability than u, we obtain from Lemma 9 that s' will select at least as much fraction of j and the other leaves in T_v with the highest vulnerability than of i and the other leaves in T_u with the highest vulnerability. Since the amount of the former is less in P' than in P by ε, while that of the latter is more in P' than in P by ε, we obtain that the value of the knapsack s' against allocation P is at least that of s' against P'. Thus, for any weight W, an optimal weight-W knapsack against P has at least as much value as an optimal weight-W knapsack against P'. Hence, P' is optimal and lexicographically smaller than P, contradicting our assumption about P. This completes the proof of the lemma.

For any leaf u and subtree T containing u, and knapsack solution x to the adversarial maximization problem, let

$y_u(T,x)$ denote the sum, over all ancestors v of u in T, of x_v.

Proof of Lemma 5: The proof is by induction on the height of the hierarchy. For the induction basis (hierarchies of height 1), the claim follows from the single-level case. For the induction step, consider any hierarchy of height h. For a given number N, let, if possible, there be two distinct hierarchical max-min fair solutions P and P'. Since the claim is true for hierarchies of height smaller than h, it follows that there exist two children u and v of the root such that $P(u) > P'(u)$ while $P(v) < P'(v)$. So there must exist an incomplete leaf node, say i, under P in the tree rooted at v, and an incomplete leaf node, say j, under P' in the tree rooted at u. We thus have $p_i(v,P) \geq p_j(u,P')$ and $p_i(v,P') \leq p_j(u,P)$. But since $P(v) < P(v)$ and $P(u) > P'(u)$, and the unique hierarchical max-min fair allocations for the two hierarchies rooted at v and u, respectively, have the property that the vulnerability of an incomplete leaf node only increases with the number of tasks, we have $p_i(v,P') > p_i(v,P)$ and $p_j(u,P) > p_j(u,P')$. These inequalities together yield a contradiction, completing the induction step and the proof of the lemma.

5. Probabilistic failures: Bicriteria approximations

We now present bicriteria polynomial-time approximations for ProbFAP and HierProbFAP. In particular, our algorithm for ProbFAP achieves an ε additive approximation in the failure probability, while giving up a $(1+\varepsilon)$ multiplicative factor in the number of failures. We next extend the result to hierarchies by giving an algorithm for HierProbFAP, which achieves an ε additive approximation in failure probability, while giving up a $(L+\varepsilon)$ multiplicative factor in the number of failures, for L-level hierarchies, where L is a constant.

5.1 Single-level case: ProbFAP

Our solution approach here builds on the stochastic optimization framework of Li and Yuan [24]. For each domain i and integer b in $[0, C_i]$, we introduce an item (i,b) with associated random variable $X(i,b)$, capturing the fraction of failed tasks if domain i fails, that has the following distribution:

$$X(i,b) = \begin{cases} 0 & \text{with probability } 1-p_i \\ \frac{b}{N} & \text{with probability } p_i \end{cases}$$

Define U as the universe of all items; i.e., $U = \{(i,b) : i \in \mathcal{F}, 0 \leq b \leq C_i\}$. We define a subset S of U as feasible if (a) for each $i \in \mathcal{F}$, there is exactly one item of the form (i,b) in S; and (b) $\sum_{(i,b)\in S} b = N$.

For a feasible solution S, define $X(S) = \sum_{(i,b)\in S} X(i,b)$. ProbFAP is then exactly the *threshold probability maximization problem* where we aim to maximize $\Pr[X(S) \leq F/N]$. Li and Yuan show that a PTAS for an associated multi-dimensional exact version of the problem implies an approximation scheme for the threshold probability maximization problem. Their framework, however, assumes that the random variables $X()$ are mutually independent. This is not true for our problem since while $X(i,b)$ is independent of $X(j,b')$ for any $i \neq j$, b, b', the variables $X(i,b)$ and $X(i,b')$ are not independent of each other. Fortunately, in our problem any feasible set of items consists of at most one item of the form (i,b) for any given i; so, the random variables associated with the items in any feasible set are all mutually independent. We use this fact to extend the Li-Yuan framework for the single-level case.

We now present the mechanism of classifying items as light or heavy, discretizing the size distributions of light items, enumerating all possible heavy item sets, and signatures of light item sets.

Item classification. Call an item (i,b) *light* if $E[X(i,b)] \leq \varepsilon^{10}$, and *heavy* otherwise.

Discretization. For item (i,b), we define variable $\tilde{X}(i,b)$ that takes values in $\{s_j = j\varepsilon^5 : 0 \leq j \leq \lfloor 1/\varepsilon^5 \rfloor\}$. We map $X(i,b)$ to $\tilde{X}(i,b)$ as follows. If $X(i,b) > \varepsilon^4$, then $\tilde{X}(i,b) = \lfloor X(i,b)/\varepsilon^5 \rfloor \varepsilon^5$. For $X(i,b) \leq \varepsilon^4$, we find a d such that $\varepsilon^4 \cdot \Pr[X(i,b) \geq d | X(i,b) \leq \varepsilon^4] = E[X(i,b) | X(i,b) \leq \varepsilon^4]$, and set $\tilde{X}(i,b) = 0$ for $0 \leq X(i,b) < d$ and $\tilde{X}(i,b) = \varepsilon^4$ for $d \leq X(i,b) \leq \varepsilon^4$. It is easy to show that for any feasible set of items, the distribution of the sum of the discretized variables for the set is close to that for the original variables.

Lemma 10 ([24, Lemma 2.2]). *For any set S of items such that $E[X(S)] \leq 3/\varepsilon$, for any k, we have*

$$|\Pr[X(S) \leq k] - \Pr[\tilde{X}(S) \leq k]| = O(\varepsilon).$$

Enumeration of heavy item sets. We enumerate all possible heavy item sets H with $E[X(H)] < 3/\varepsilon$. Each heavy item set is of constant cardinality, hence the number of such sets is polynomial. In the remainder, we can assume a fixed heavy item set H.

Signatures. For an item (i,b), we define its signature to be the vector $(\pi_{i,b}(s_1), \ldots, \pi_{i,b}(s_{z-1}))$, where

$$\pi_{i,b}(s) = \frac{\varepsilon^6}{n} \left\lfloor \frac{n}{\varepsilon^6} \cdot \Pr[\tilde{X}(i,b) = s] \right\rfloor$$

for all s in $\{s_1, s_2, \ldots, s_{z-1}\}$. We define the signature of a set of items to be the coordinate-wise sum of the signatures of the individual items in the set. A key claim [24, Lemma 2.3] shows that it is sufficient to consider signatures whose components add up to at most $3/\varepsilon$, and enumerate signatures for set of light items. The proof of this lemma relies on the Poisson approximation theorem for sums of independent random variables [22]. Since we apply the Possion approximation theorem for random variables associated with the items in any feasible set, the independence of these variables continues to hold, the same lemma holds.

Lemma 11. *Let S_1 and S_2 be two sets of light items, with the same signature, and $E[\tilde{X}(S_1)] \leq 3/\varepsilon$, $E[\tilde{X}(S_2)] \leq 3/\varepsilon$. Then, the total variation distance between $X(S_1)$ and $X(S_2)$ is $O(\varepsilon)$.*

Using Lemmas 10 and 11, we then obtain the equivalent of Theorem 1.1 of [24]: If there is a pseudopolynomial algorithm for determining whether, for a given weight bound $\mathcal{W}$, there is a feasible solution of weight $\mathcal{W}$ for a given instance of ProbFAP where individual elements have weights, then there is a polynomial-time approximation algorithm for ProbFAP that finds a feasible solution S such that $\Pr[X(S) \leq (1+\varepsilon)F/N] \geq \text{OPT} - \varepsilon$, where OPT is the maximum value achieveable for $\Pr[X(S) \leq F/N]$, for any feasible S.

Algorithm for an exact version of ProbFAP. In the exact version of ProbFAP, each element (i,b) of U is associated with a weight $\omega_{i,b}$, where the weight will represent

the item signature we have defined above. We are given a weight constraint W and we would like to determine if there is a feasible solution S such that $\sum_{(i,b)\in S} \omega_{i,b} = \mathcal{W}$. Note that $\mathcal{W}$ represents the overall signature we desire for the set of light items. We define $\Pi(X, M, j)$ to be a predicate that is true if it is possible to assign tasks to domains 1 through j so that the total weight of the assignment is exactly X. We set up a dynamic program as follows: $\Pi(0, 0, 0)$ is true; $\Pi(X, M, j)$ is true if $\Pi(X - w(i, b), M - b, j - 1)$ is true for any $0 \le b \le C_j$, and false otherwise. It can be seen that the above can be computed in time polynomial in the number of tasks N, the number of possible weight values, and the number of domains n. Furthermore, the algorithm extends to the scenario where the weights are constant-dimensional vectors, as is the case with the signatures; the exponent in the polynomial time grows with the dimension of the weight vectors. Finally, note that our algorithm as presented is polynomial in N, not the size of N. We apply the standard method of restricting b values in the items (i, b) to rounded powers of $(1 + \varepsilon)$. This causes us another factor of $(1 + \varepsilon)$ loss in the number of failed tasks, and yields the following single-level special case of Theorem 6.

THEOREM 9. *For any $\varepsilon > 0$, there exists a polynomial time algorithm that runs in time $n^{poly(1/\varepsilon)}$ and returns a feasible placement of N tasks such that*

$$\Pr[\textit{number of failed tasks } \le (1+\varepsilon)F] \ge \mathrm{OPT} - \varepsilon,$$

where OPT *is the maximum probability, over any placement of N tasks, that the number of failed tasks is at most F.*

5.2 Hierarchical case: HierProbFAP

We next consider the hierarchical version, HierProbFAP. For each node i, we define an item (i, b) for each $0 \le b \le C_i$. As before, we have the associated random variable $X(i, b)$, that takes value 0 with the probability that at least one of its ancestor domains fails, and b/N otherwise. The $X(i, b)$ variables are no longer independent, even for different i. This is because nodes share ancestor domains.

One approach to this problem is to solve multiple stochastic knapsack problems, one for each of the L levels, using the single-level algorithm of Section 5.1 and then select the best of all the solutions. This approach is unlikely to yield any useful approximation guarantees, however, since the solution for any one level may perform poorly for other levels. We present an approximation algorithm that considers the multiple level problems in an integrated manner and identifies a single solution that performs close to the optimal at each level. Let $P_\ell(E)$ denote the probability of event E, over the failure events of domains at level ℓ; that is,

$$P_\ell(E) = \Pr[E| \text{ no domain at any level } i \ne \ell \text{ fails}].$$

For subproblem at level ℓ, we enumerate a polynomial number of solutions to find all feasible solutions S_ℓ such that $P_\ell(X(S_\ell) \le (1+\varepsilon/L)F/N) \le \mathrm{OPT}_\ell - \varepsilon/L$, where OPT_ℓ is the maximum probability, over any placement of N tasks, that the number of failed tasks due to failures exclusively at level ℓ is at most F. By our enumeration of the solutions, we will identify *a single solution S* such that $P_\ell(X(S) \le (1+\varepsilon)F/N) \ge \mathrm{OPT}_\ell - \varepsilon/L$ *for each ℓ*. By combining the bound over all the levels, we will establish that S satisfies

$$\Pr[X(S) \le (L+\varepsilon)F/N] \ge \mathrm{OPT} - \varepsilon.$$

For each level, as in Section 5.1, we classify items as light or heavy based on their expected size, discretize the size distributions for light items, and enumerate all possible heavy item sets and signatures for light item sets. The item classification and discretization is the same as in Section 5.1. The heavy item sets and the dynamic program we set up to process the light item sets across all levels in an integrated manner are described below.

Enumeration of heavy item sets. We enumerate all possible heavy item sets H with $E[X(H)] < 3/\varepsilon$ at each level. Each heavy item set is of constant cardinality, hence the number of distinct heavy sets at each level is polynomial. Since the number of levels is constant, there is a polynomial number of ways to select a heavy item set at each level.

Dynamic programming. Fix a heavy item set H_ℓ for each level ℓ of the hierarchy. Fix a signature σ_ℓ for each level ℓ of the hierarchy. Recall that each signature is a vector with $O(\varepsilon^{-5})$ coordinates, and the value of each coordinate is bounded by $O(n)$.

Our goal now is to determine whether there is a selection of items so that L_ℓ is the set of light items selected at level ℓ, $H_\ell \cup L_\ell$ is feasible for each ℓ, and $Sg(L_\ell) = \sigma_\ell$ for each ℓ; here, $Sg(I)$ of a set I of light items denotes the signature of the set. We set up a dynamic program that determines if there is a feasible selection of light items satisfying the preceding condition. Let $\lambda(v)$ denote the level of node v in the hierarchy. For a given node v, positive integer i at most the number of children of v, and sequence of signature vectors $\sigma_1, \sigma_2, \ldots, \sigma_{\lambda(v)}$, we define $\Phi(v, i, \sigma_1, \ldots, \sigma_{\lambda(v)})$ to be true if there exists a feasible selection of light items for the subtree consisting of v, the first i children of v, and their descendants such that the signature of the set of light items at level ℓ, $1 \le \ell \le \lambda(v)$, equals σ_ℓ. We compute $\Phi()$ using the following dynamic program. For a leaf node v we set

$$\Phi(v, 0, \sigma) = \begin{cases} \textbf{true} & \text{if } \sigma = (0, 0, \ldots) \text{ or } \exists b : Sg(\{(v, b)\}) = \sigma \\ \textbf{false} & \text{otherwise.} \end{cases}$$

For an internal node v, let $d(v)$ denote the number of children of v, and let v_i denote the ith child of v, numbered in an arbitrary order. For internal node v, we compute that for $i > 0$ $\Phi(v, i, \sigma_1, \ldots, \sigma_{\lambda(v)})$ equals

$$\bigvee_{\sigma'_1, \ldots, \sigma'_{\lambda(v)-1}} \big(\Phi(v, i-1, \sigma_1 - \sigma'_1, \ldots, \sigma_{\lambda(v)-1} - \sigma'_{\lambda(v)-1}, \sigma_{\lambda(v)}) \wedge \Phi(v_i, d(v_i), \sigma'_1, \ldots, \sigma'_{\lambda(v)-1})\big),$$

where we note that $\lambda(v_i) = \lambda(v) - 1$. Finally, $\Phi(v, 0, \sigma_1, \ldots, \sigma_{\lambda(v)})$ is exactly the single-level problem for node v in which we determine if there exists an selection of light items for the domains v_1 through $v_{d(v)}$ such that the heavy items selected for this subset of domains, together with the light items form a feasible solution for this set of domains, and the signature of the light item set matches $\sigma_{\lambda(v)}$.

Final output. We have written the above dynamic program as a predicate; it is straightforward to extract an actual solution that satisfies the condition desired in Φ. Thus, after solving the dynamic program, for any combination of signatures σ_i at level i, for $i \le \ell$, if there exists a solution that has the signature σ_i at level i, for $i \le \ell$, we obtain one such solution. Among these, we select one solution that has the highest product $\prod_l P_\ell(X_\ell(S) \le (1+\varepsilon)F/N)$, which we can estimate using the signatures, as in the single-level subproblem.

Analysis of the algorithm. Since the number of different signature values is polynomial ($n^{O(\varepsilon^{-5})}$) and the number of levels is a constant, the total number of different inputs for which Φ is being computed is polynomial in n. This ensures that in polynomial time, we determine a feasible solution S of items such that at each level ℓ, the signature of S matches that of a globally optimal solution, say S^*. If $X_\ell(S)$ (resp., $X_\ell(S^*)$ denotes the number of failed tasks in S (resp., S^*) due to failures in level ℓ, then Lemma 11 implies that at each level ℓ the variation distance between $\tilde{X}_\ell(S)$ and $\tilde{X}_\ell(S^*)$ is $O(\varepsilon)$. Together with Lemma 10, we get

$$P_\ell(X_\ell(S) \leq (1+\varepsilon)F/N) \geq \mathrm{OPT}_\ell - O(\varepsilon) \tag{2}$$

where OPT_ℓ is the maximum probability, over any placement of N tasks, that the number of failed tasks due to failures exclusively at level ℓ is at most F. Since the failure events at each of the failure domains are independent of one another, $\prod_\ell \mathrm{OPT}_\ell$ is an upper bound on maximum probability OPT, over any placement of N, that the number of failed tasks due to failures at all levels is at most F. Here we have used the fact that for the total number of failures to be at most F, the number of failures at each level need to be at most F. We thus obtain

$$\begin{aligned}
\Pr[X(S) \leq (L+\varepsilon)F/N] &\geq \prod_\ell P_\ell(X_\ell(S) \leq (1+\varepsilon/L)F/N) \\
&\geq \prod_\ell (\mathrm{OPT}_\ell - O(\varepsilon/L)) \\
&\geq \mathrm{OPT} - O(\varepsilon),
\end{aligned}$$

where in the second step, we invoke Equation 2, with ε replaced by ε/L. This gives us Theorem 6, restated below.

THEOREM 10 (RESTATEMENT OF THEOREM 6). *For any $\varepsilon > 0$, there exists a polynomial time algorithm for* HIERPROBFAP *over an L-level hierarchy that runs in time $n^{poly(L,1/\varepsilon)}$ and returns a feasible placement such that*

$$\Pr[\textit{number of failed tasks } \leq (L+\varepsilon)F] \geq \mathrm{OPT} - O(\varepsilon),$$

where OPT *is the maximum probability, over any placement of N tasks, that the number of failed tasks is at most F.*

6. Conclusions and open problems

In this paper, we formulate and initiate the study of *failure-aware placement* for hierarchical data centers under adversarial and probabilistic failures. For both classes, we give hardness results and approximation algorithms for multiple variants: based on *generalized spreading* for HIERROBUSTFAP and Poisson approximation for HIERPROBFAP.

One natural question is to improve the approximation guarantees and design more practical implementations for some of the variants. In particular, the approximation factor in the bicriteria PTAS for HIERPROBFAP grows linearly in the number of levels. While this is reasonable for current datacenters (which typically have three to four levels of hierarchy), it would be interesting to see if the approximation factor can be made sublinear or even independent of number of levels. Similarly, our PTASes incur high running times (e.g., for ROBUSTFAP, a standard implementation would take time $O(n^{(3+ln(1/\varepsilon))/\varepsilon})$ time), and it is important to design practical implementations of PTASes that trade off approximation guarantees for simplicity.

On a related note, our study here has focused on identical tasks of the same size. It will be interesting to extend this to an on-line sequence of jobs of possibly different sizes. A further extension would be to consider dynamic task migrations to improve availability as jobs finish and leave.

Acknowledgments

We would like to thank Gerhard Woeginger for providing insights into the results of [7], and sharing a full version of their paper.

7. References

[1] S. Agrawal, Y. Ding, A. Saberi, and Y. Ye. Price of correlations in stochastic optimization. *Operations Research*, 60:243–248, February 2012.

[2] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[3] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4:1–108, 2009.

[4] D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52, 2004.

[5] A. Bhalgat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *SODA*, 2011.

[6] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica. Surviving failures in bandwidth-constrained datacenters. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012.

[7] Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J. Woeginger. A complexity and approximability study of the bilevel knapsack problem. In *Integer Programming and Combinatorial Optimization*, pages 98–109. 2013.

[8] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[9] C. Daskalakis, A. De, I. Diakonikolas, A. Moitra, and R. Servedio. A polynomial-time approximation scheme for fault-tolerant distributed storage. In *SODA*, 2014.

[10] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM*, 51:107–113, 2008.

[11] Christina Delimitrou and Christos Kozyrakis. QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon. In *ACM Transactions on Computer Systems (TOCS)*, 2013.

[12] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *SIGCOMM Comput. Commun. Rev.*, 19(4):1–12, August 1989.

[13] S. Dempe. *Foundations of bilevel programming*. Kluwer Academic Publishers, 2002.

[14] S. Dempe and K. Richter. Bilevel programming with knapsack constraint. *Central European Journal of Operations Research*, 2000.

[15] S. DeNegre. *Interdiction and discrete bilevel linear programming*. PhD thesis, Lehigh University, 2011.

[16] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V. A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, 2010.

[17] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, 2011.

[18] M. Gribaudo, M. Iacono, and D. Manini. Improving reliability and performances in large scale distributed applications with erasure codes and replication. *Future Generation Computer Systems*, 2015.

[19] S. Keshav. *An Engineering Approach to Computer Networking: ATM Networks, the Internet, and the Telephone Network*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[20] O. Klopfenstein and D. Nace. A robust approach to the chance-constrained knapsack problem. *Operations Research Letters*, 36, 2008.

[21] M. Korupolu and R. Rajaraman. Robust and probabilistic failure-aware placement, 2016.

[22] L. Le Cam. An approximation theorem for the Poisson binomial distribution. *Pacific Journal of Mathematics*, 10:1181–1197, 1960.

[23] J. Li and A. Deshpande. Maximizing expected utility for stochastic combinatorial optimization problems. In *FOCS*, 2011.

[24] J. Li and W. Yuan. Stochastic combinatorial optimization via Poisson approximation. In *STOC*, 2013.

[25] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[26] R. Mansi, C. Alves, J. de Carvalho, and S. Hanafi. An exact algorithm for bilevel 0-1 knapsack problems. *Mathematical Problems in Engineering*, 2012.

[27] K. A. Mills, R. Chandrasekaran, and N. Mittal. Algorithms for replica placement in high-availability storage. In *ArXiv preprint arXiv:1503.02654*, 2015.

[28] J. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35, 1987.

[29] A. Nemirovski and A. Shapiro. Convex approximations of chance constrained programs. *SIAM Journal of Optimization*, 17, 2006.

[30] E. Nikolova. Approximation algorithms for offline risk-averse combinatorial optimization. In *APPROX*, 2010.

[31] B. K. Pagnoncelli, S. Ahmed, and A. Shapiro. Sample average approximation method for chance-constrained programming: Theory and applications. *Journal of Optimization theory and Applications*, 142, 2009.

[32] Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[33] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1, 1993.

[34] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *FAST*, pages 17–23, 2007.

[35] James S Plank et al. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Softw., Pract. Exper.*, 27:995–1012, 1997.

[36] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozych. Heterogeneity and dynamicity at scale: Google trace analysis. In *ACM Symposium on Cloud Computing*, 2012.

[37] B. Schroeder, E. Pinheiro, and W. D. Weber. Dram errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204, 2009.

[38] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, volume 78, page 012022. IOP Publishing, 2007.

[39] M. Sedaghat, E. Wadbro, J. Wilkes, S. De Luna, O. Seleznjev, and E. Elmroth. Die-hard: Reliable scheduling to survive correlated failures in cloud data centers. In *Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016.

[40] D. Tang and R. K. Iyer. Analysis and modeling of correlated failures in multicomputer systems. In *IEEE Transactions on Computers*, volume 41, pages 567–577, 1992.

[41] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.

[42] K. V. Vishwanath and N. Nagappan. Characterizing cloud computing hardware reliability. In *ACM Symposium on Cloud Computing, SOCC*, 2010.

[43] H. von Stackelberg. *The theory of market economy*. Oxford University Press, 1952.

[44] P. Yalagandula, S. Nath, H. Yu, P. B. Gibbons, and Seshan S. Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems. In *Proceedings of the Workshop on Real, Large Distributed Systems (WORLDS âĂŹ04)*, 2004.

[45] M. Zaharia, M. Chowdhury, M . J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010.