Overcommitment in Cloud Services: Bin Packing with Chance Constraints

Maxime C. Cohen,[a] Philipp W. Keller,[b] Vahab Mirrokni,[c] Morteza Zadimoghaddam[c]

[a] NYU Stern School of Business, New York, New York 10012; [b] Facebook, Menlo Park, California 94043; [c] Google Research, New York, New York 10011

Contact: maxime.cohen@stern.nyu.edu, [iD] http://orcid.org/0000-0002-2474-3875 (MCC); pkeller@fb.com (PWK); mirrokni@google.com (VM); zadim@google.com (MZ)

Abstract. This paper considers a traditional problem of resource allocation: scheduling jobs on machines. One such recent application is cloud computing; jobs arrive in an online fashion with capacity requirements and need to be immediately scheduled on physical machines in data centers. It is often observed that the requested capacities are not fully utilized, hence offering an opportunity to employ an *overcommitment policy*, that is, selling resources beyond capacity. Setting the right overcommitment level can yield a significant cost reduction for the cloud provider while only inducing a very low risk of violating capacity constraints. We introduce and study a model that quantifies the value of overcommitment by modeling the problem as bin packing with chance constraints. We then propose an alternative formulation that transforms each chance constraint to a submodular function. We show that our model captures the risk pooling effect and can guide scheduling and overcommitment decisions. We also develop a family of online algorithms that are intuitive, easy to implement, and provide a constant factor guarantee from optimal. Finally, we calibrate our model using realistic workload data and test our approach in a practical setting. Our analysis and experiments illustrate the benefit of overcommitment in cloud services and suggest a cost reduction of 1.5% to 17%, depending on the provider's risk tolerance.

1. Introduction

Bin packing is an important problem with numerous applications, such as hospitals; call centers; filling up containers; loading trucks with weight capacity constraints; creating file backups; and more recently, cloud computing. A cloud provider needs to decide how many physical machines to purchase to accommodate the incoming jobs efficiently. This is typically modeled as a bin-packing optimization problem; one minimizes the cost of acquiring the physical machines subject to a capacity constraint for each machine. The jobs are assumed to arrive in an online fashion according to a given arrival process. In addition, the jobs come with a specific requirement, but the effective job size and duration are not exactly known until the actual scheduling has occurred. In practice, job size and duration can be estimated from historical data. One straightforward way to schedule jobs is to assume that each job will fully utilize its requirement (e.g., if a job requests 32 CPU cores, the cloud provider allocates this exact amount). However, there is empirical evidence that most of the virtual machines do not use the full requested capacity. This offers an opportunity for the cloud provider to employ an *overcommitment policy*, that is, to schedule sets of jobs with the total requirement exceeding the capacities of physical machines. On the one hand, the provider faces the risk that usage exceeds the physical capacity, which can result in severe penalties (e.g., acquiring or reallocating machines on the fly, canceling and rescheduling running jobs, mitigating interventions). On the other hand, if many jobs do not fully utilize their requested resources, the provider can potentially reduce the costs significantly. This becomes even more impactful in the cloud computing market, which has become increasingly competitive in recent years as Google, Amazon, and Microsoft aim to replace private data centers. "The race to zero price" is a commonly used term for this industry, in which cloud providers have cut their prices very aggressively. According to an online article in *Business Insider* in January 2015, "Amazon Web Services (AWS), for example, has cut its price 44 times during 2009-2015, while Microsoft and Google have both decreased prices multiple times to keep up with AWS. RBC Capital's Mark Mahaney published a chart that perfectly captures this trend and shows that the average monthly cost per

gigabyte of RAM has dropped significantly: AWS dropped prices 8% from October 2013 to December 2014 while both Google and Microsoft cut prices by 6% and 5%, respectively, in the same period. Other companies who charge more, such as Rackspace and AT&T, dropped prices even more significantly."

As a result, designing the right overcommitment policy for servers has a clear potential to increase the cloud provider's profit. The goal of this paper is to study this question and propose a model that helps guide this type of decisions. In particular, we explicitly model job size uncertainty to motivate new algorithms and evaluate them on realistic workloads. Our model and approaches are not limited to cloud computing and can be applied to several resource-allocation problems. However, we illustrate most of the discussions and applications using examples borrowed from the cloud computing world. Note that describing the cloud infrastructure and hardware is beyond the scope of this paper. For surveys on cloud computing, see, for example, Fox et al. (2009) and Dinh et al. (2013).

We propose to model the problem as bin packing with chance constraints, that is, the total load assigned to each machine should be below physical capacity with a high, prespecified probability. Chance constraints are a common modeling tool to capture risks and constraints on random variables (Charnes and Cooper 1963). Introducing chance constraints to several continuous optimization problems was extensively studied in the literature (see, e.g., Calafiore and El Ghaoui 2006 and Delage and Ye 2010). This paper is among the first to incorporate capacity chance constraints in the bin-packing problem and to propose efficient algorithms to solve it. Using results from distributionally robust optimization (Calafiore and El Ghaoui 2006), we reformulate the problem as bin packing with submodular capacity constraints. Our reformulations are exact under independent Gaussian resource usages. More generally, they provide an upper bound and a good practical approximation in the realistic case in which the jobs' usages are arbitrarily distributed but bounded.

Using some machinery from previous work (see Goemans et al. 2009 and Svitkina and Fleischer 2011), we show that for the bin-packing problem with general monotone submodular constraints, it is impossible to find a solution within any reasonable factor from optimal (more precisely, $\frac{\sqrt{N}}{\ln(N)}$, where N is the number of jobs). In this paper, we show that our problem can be solved using a class of simple online algorithms that guarantee a constant factor of 8/3 from optimal (Theorem 2). This class of algorithms includes the commonly used *best-fit* and *first-fit* heuristics. We also develop an improved constant guarantee of 9/4 for the online problem (Theorem 4) and a two-approximation for the off-line version (Theorem 6). We further refine our results to the case in which a large

number of jobs can be scheduled on each machine (i.e., each job has a small size relative to the machine capacity). In this regime, our approach asymptotically converges to a 4/3 approximation. More importantly, our model and algorithms allow us to draw interesting insights on how one should schedule jobs. In particular, our approach (i) translates to a transparent recipe on how to assign jobs to machines, (ii) explicitly exploits the risk-pooling effect, and (iii) can be used to guide an overcommitment strategy that significantly reduces the cost of purchasing machines.

We apply our algorithm to a synthetic but realistic workload inspired by historical production workloads in Google data centers and show its good performance. In particular, our method reduces the necessary number of physical machines while limiting the risk borne by the provider.

1.1. Contributions

Scheduling jobs on machines can be modeled as a bin-packing problem. Jobs arrive online with their requirements, and the scheduler decides how many machines to purchase and how to schedule the jobs. The objective is to minimize the number of machines, subject to capacity constraints on each machine. In this paper, we model the capacity constraints as chance constraints and study the potential benefit of overcommitment. Our contributions can be summarized as follows.

• **Formulating the overcommitment bin-packing problem.** We discuss an optimization formulation for scheduling jobs on machines while allowing the provider to overcommit. We first model the problem as *bin packing with chance constraints* (BPCC). Then we present an alternative *submodular bin-packing* (SMBP) formulation that explicitly captures the risk-pooling effect on each machine. We show that SMBP is equivalent to BPCC under independent Gaussian usage distributions and that it is distributionally robust for usages with given means and a diagonal covariance matrix. Perhaps most importantly from a practical perspective, SMBP provides an upper bound and a good approximation under generic independent distributions over bounded intervals (see Proposition 1).

• **Developing simple algorithms that guarantee a constant factor approximation from optimal.** We show that the (SMBP) can be solved by well-known online algorithms, such as first fit and best fit, while guaranteeing a factor of 8/3 from optimal (Theorem 2). We further refine this result in the case in which a large number of jobs can be scheduled on each machine and obtain a 4/3 approximation asymptotically (Corollary 1). We also develop an improved constant guarantee of 9/4 for the online problem using first fit (Theorem 4), and a two-approximation for the off-line version (Theorem 6). We then use our analysis

to infer how one should assign jobs to machines and show how to obtain a nearly optimal assignment (Theorem 5).

• **Using our model to draw practical insights on the overcommitment policy.** Our approach translates to a transparent and meaningful recipe on how to assign jobs to machines by clustering similar jobs in terms of statistical information. In addition, our approach explicitly captures the risk-pooling effect: as we assign more jobs to a given machine, the "safety buffer" needed for each job decreases. Finally, our approach can guide practical overcommitment strategies to reduce the cost of purchasing machines by allowing a low risk of violating capacity constraints.

• **Calibrating and applying our model to a practical setting.** We use realistic workload data inspired by Google Compute Engine to test our results. We observe that our proposed algorithm outperforms other natural scheduling schemes and achieves a cost saving of 1.5% to 17% relative to the no-overcommitment policy.

1.2. Structure of the Paper
In Section 2, we present our model and assumptions, and in Section 3 we review the relevant literature. Then we present the results and insights for special cases in Section 4. In Section 5, we consider the general case and develop a class of efficient approximation algorithms that guarantee a constant factor from the optimal number of machines. We then exploit the structure of the problem to obtain a nearly optimal assignment. In Sections 6 and 7, we present extensions and computational experiments, respectively. Finally, our conclusions are reported in Section 8. Most of the proofs of the technical results are relegated to the online appendix.

2. Model
In this section, we start by formulating the problem we want to solve and then propose an alternative formulation. As discussed, job requests for cloud services (or other resource-allocation problems) come with a requested capacity. This can be the memory or CPU requirements for virtual machines in the context of cloud computing or job duration in more traditional scheduling problems in which jobs are processed sequentially.[1] We refer to A_j as the size of job j and assume that A_j is a random variable. Historical data can provide insight into the distribution of A_j. For simplicity, we first consider the offline version of the problem in which all the jobs arrive simultaneously at time 0, and our goal is to pack them onto the minimum possible number of machines. Jobs cannot be delayed or preempted. The methods we develop in this paper can be applied to the more interesting online version of the problem as we discuss in Section 5. We denote the capacity of machine i by V_i. Motivated by practical problems, and in accordance with prior work, we

assume that all the machines have the same capacity, that is, $V_i = V; \forall i$. In addition, each machine has the same cost, and our goal is to maximize the total profit (or, equivalently, minimize the number of machines) while scheduling all the jobs and satisfying the capacity constraints. Note that we consider a single-dimensional problem in which each job has one capacity requirement. Although cloud virtual machine packing may be modeled as a low-dimensional vector bin-packing problem (Lee et al. 2011), one resource is often effectively binding and/or more critical.[2]

2.1. Bin-Packing Problem
For the case in which A_j is deterministic, we obtain the classical *deterministic bin-packing* problem:

$$
\begin{aligned}
B = \min_{x_{ij}, y_i} \quad & \sum_{i=1}^{N} y_i \\
\text{s.t.} \quad & \sum_{j=1}^{N} A_j x_{ij} \le V y_i \quad \forall i \\
& \sum_{i=1}^{N} x_{ij} = 1 \quad \forall j \\
& x_{ij} \in \{0, 1\} \quad \forall i, j \\
& y_i \in \{0, 1\} \quad \forall i
\end{aligned} \quad \text{(DBP)}
$$

For the offline version, we have a total of N jobs, and we need to decide how many machines to purchase (captured by the decision variable y_i that is equal to 1 if machine i is purchased and 0 otherwise). The solution is a B-partition of the set $\{1, 2, \ldots, N\}$ that satisfies the capacity constraints. The decision variable x_{ij} equals to 1 if job j is assigned to machine i and 0 otherwise. As we discuss in Section 3, there is extensive literature on the DBP problem.

The problem faced by a cloud provider is typically *online* in nature because jobs arrive and depart over time. Unfortunately, it is not possible to continually resolve the DBP problem as the data are updated because of both practical and computational reasons. Keeping with the majority of prior work, we start by basing our algorithms on static, single-period optimization formulations such as the DBP problem rather than explicitly modeling arrivals and departures. The next section explains how, unlike prior work, our single-period optimization model efficiently captures the uncertainty faced by a cloud provider. We consider both the online and off-line versions of our model.

We remark that, although our online analysis considers sequentially arriving jobs, none of our results explicitly consider departing jobs. This is in line with the bin-packing literature, in which results typically apply to general arrival processes $\{A_j\}$, but it is usually assumed that packed items remain in their assigned bins. In practice, a large cloud provider is likely to be

interested in a steady state in which the distribution of jobs is stable over time even if individual jobs come and go. Note that several works considers bin packing with item departures (see, e.g., Stolyar and Zhong 2015 and the references therein). In this work, the authors design a simple greedy algorithm for general packing constraints and show that it can be asymptotically optimal. Nevertheless, Gupta and Radovanovic (2015) show simulation results in which, for certain distributions, the best-fit algorithm under static packing is suboptimal but is asymptotically optimal under dynamic packing.

2.2. Chance Constraints

The DBP problem suffers from the unrealistic assumption that the jobs' sizes A_j are deterministic. In reality, jobs' requirements can be highly unpredictable, especially from the perspective of a cloud provider with no control over the software executed in a virtual machine. Ensuring that the capacity constraints are satisfied for any realization of A_j generally yields a conservative outcome. For example, if the jobs' true requirements are binary random variables taking on either 0.3 or 1.0 with equal probability, one needs to plan as if each job consumes a capacity of 1.0. By overcommitting resources, the provider can reduce the cost significantly. Caution is required though because overcommitting can be expensive if not done properly. Planning according to the expected value (in the previous example, 0.65), for instance, would result in capacity being insufficient for several machines. Depending on the specific resource and the degree of violation, such performance could be catastrophic for a cloud provider. Concretely, sustained CPU contention among virtual machines would materially affect customers' performance whereas a shortage of memory could require temporarily "swapping" some data to a slower storage medium with usually devastating consequences on performance. With this motivation in mind, our goal is to propose a formulation that finds the right overcommitment policy. We show that by slightly overcommitting, one can reduce the costs significantly while satisfying the capacity constraints with high probability.

Although not strictly required by our approach, in practice, there is often an upper bound on A_j, denoted by $\bar{A}_j$. In the context of cloud computing, $\bar{A}_j$ is the requested capacity that a virtual machine is not allowed to exceed (e.g., 32 CPU cores). However, the job may end up using much less, at least for some time. If the cloud provider schedules all the jobs according to their respective upper bounds $\bar{A}_j$, then there is no overcommitment. If the jobs are scheduled according to

sizes smaller than $\bar{A}_j$, then some of the machines may be overcommitted.

We propose to solve a bin-packing problem with capacity chance constraints. Chance constraints are widely used in optimization problems, starting with Charnes and Cooper (1963) for linear programs, and more recently in convex optimization (Nemirovski and Shapiro 2006) and in finance (see, e.g., Abdelaziz et al. 2007). In our case, the capacity constraints are replaced by

$$\mathbb{P}\left(\sum_{j=1}^{N} A_j x_{ij} \le V y_i\right) \ge \alpha, \tag{1}$$

where α represents the confidence level of satisfying the constraint ($\alpha = 0.999$, say) and is exogenously set by the cloud provider. Note that when $\alpha = 1$, this corresponds to the setting with no overcommitment or, in other words, to the worst-case solution that covers all possible realizations of all the A_j's. One of our goals is to study the trade-off between the probability of violating physical capacity and the cost reduction resulting from a given value of α. The problem becomes *bin packing with chance constraints*, parameterized by α:

$$B(\alpha) = \min_{x_{ij}, y_i} \sum_{i=1}^{N} y_i$$

$$\text{s.t.} \quad \mathbb{P}\left(\sum_{j=1}^{N} A_j x_{ij} \le V y_i\right) \ge \alpha \quad \forall i$$

$$\sum_{i=1}^{N} x_{ij} = 1 \quad \forall j \tag{BPCC}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j$$

$$y_i \in \{0, 1\} \quad \forall i$$

2.3. A Variant of Submodular Bin Packing

We next propose an alternative formulation that is closely related to the (BPCC) problem. Under some mild assumptions, we show that the latter is either exactly or approximately equivalent to the following *submodular bin-packing* problem:

$$B_S(\alpha) = \min_{x_{ij}, y_i} \sum_{i=1}^{N} y_i$$

$$\text{s.t.} \quad \sum_{j=1}^{N} \mu_j x_{ij} + D(\alpha)\sqrt{\sum_{j=1}^{N} b_j x_{ij}} \le V y_i \quad \forall i$$

$$\sum_{i=1}^{N} x_{ij} = 1 \quad \forall j$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j$$

$$y_i \in \{0, 1\} \quad \forall i \tag{SMBP}$$

The difference between the (BPCC) and the (SMBP) is the way the capacity constraints are written. Here, we have replaced each chance constraint with a linear term plus a square root term. These constraints are submodular with respect to the vector **x**. The variable μ_j denotes the expected value of A_j. The left-hand side of the capacity constraint for each machine i in the (SMBP) is a submodular function[3] as shown in Atamtürk and Narayanan (2008). In what follows, we consider different definitions of b_j and $D(\alpha)$. The first two are concrete motivational examples whereas the third one is a generalization.

a. Gaussian Case. Assume that A_j are Gaussian and independent. In this case, the random variable $Z = \sum_{j=1}^{N} A_j x_{ij}$ for any given binary vector **x** is Gaussian, and therefore, one can use the following simplification:

$$\mathbb{P}\left(\sum_{j=1}^{N} A_j x_{ij} \le V y_i \right) = \mathbb{P}(Z \le V y_i) \ge \alpha.$$

For each machine i, constraint (1) becomes

$$\sum_{j=1}^{N} \mu_j x_{ij} + \Phi^{-1}(\alpha) \cdot \sqrt{\sum_{j=1}^{N} \sigma_j^2 x_{ij}} \le V y_i, \qquad (2)$$

where $\Phi^{-1}(\cdot)$ is the inverse CDF of a normal $N(0,1)$, $\mu_j = \mathbb{E}[A_j]$, and $\sigma_j^2 = \mathrm{Var}(A_j)$. Note that we have used the fact that **x** is binary so that $x_{ij}^2 = x_{ij}$. Consequently, the (BPCC) and the (SMBP) are equivalent with the values $b_j = \sigma_j^2$ and $D(\alpha) = \Phi^{-1}(\alpha)$. When A_j are independent but not normally distributed, if there are a large number of jobs per machine, one can apply the central limit theorem and obtain a similar approximate argument. In fact, using a result from Calafiore and El Ghaoui (2006), one can extend this equivalence to any radial distribution.[4]

b. Hoeffding's Inequality. Assume that A_j are independent with a finite support $[\underline{A}_j, \overline{A}_j]$, $0 \le \underline{A}_j < \overline{A}_j$ with mean μ_j. One can use historical data to estimate μ_j and $\underline{A}_j$ (see more details in Section 7). Assume that the mean usages fit on each machine, that is, $\sum_{j=1}^{N} x_{ij}\mu_j < y_i V_i$. Then, Hoeffding's inequality states that

$$\mathbb{P}\left(\sum_{j=1}^{N} A_j x_{ij} \le V y_i \right) \ge 1 - e^{\frac{-2[V y_i - \sum_{j=1}^{N} \mu_j x_{ij}]^2}{\sum_{j=1}^{N} (\overline{A}_j - \underline{A}_j)^2 x_{ij}}}.$$

Equating the right hand side to α, we obtain

$$\frac{-2[V y_i - \sum_{j=1}^{N} \mu_j x_{ij}]^2}{\sum_{j=1}^{N} b_j x_{ij}} = \ln(1 - \alpha),$$

where $b_j = (\overline{A}_j - \underline{A}_j)^2$ represents the range of job j's usage. Rearranging the equation, we obtain

$$\sum_{j=1}^{N} \mu_j x_{ij} + D(\alpha) \sqrt{\sum_{j=1}^{N} b_j x_{ij}} \le V y_i, \qquad (3)$$

where, in this case, $D(\alpha) = \sqrt{-0.5 \ln(1 - \alpha)}$. Note that in this setting the (BPCC) and the (SMBP) are not equivalent. Instead, any solution of the latter is a feasible solution for the former. We demonstrate in Section 7 that, despite being conservative, this formulation based on Hoeffding's inequality actually yields good practical solutions. The next case is a generalization of the last two.

c. Distributionally Robust Formulations. Assume that A_j are independent with some unknown distribution that belongs to a family of probability distributions $\mathcal{D}$. We consider two commonly used examples of such families: (i) the family $\mathcal{D}_1$ of distributions with a given mean $\boldsymbol{\mu}$ and (diagonal) covariance matrix $\boldsymbol{\Sigma}$ and (ii) the family $\mathcal{D}_2$ of generic distributions of independent random variables over bounded intervals $[\underline{A}_j, \overline{A}_j]$. In this setting, the chance constraint is enforced robustly with respect to the entire family $\mathcal{D}$ of distributions on $\mathbf{A} = (A_1, A_2, \ldots, A_N)$:

$$\inf_{\mathbf{A} \sim \mathcal{D}} \mathbb{P}\left(\sum_{j=1}^{N} A_j x_{ij} \le V y_i \right) \ge \alpha. \qquad (4)$$

In this context, we have the following result.

Proposition 1. *Consider the robust bin-packing problem with the capacity chance constraints* (4) *for each machine i. Then, for any $\alpha \in (0, 1)$, we have*
- *For the family of distributions $\mathcal{D}_1$, the robust problem is equivalent to the (SMBP) with $b_j = \sigma_j^2$ and $D_1(\alpha) = \sqrt{\alpha/(1 - \alpha)}$.*
- *For the family of distributions $\mathcal{D}_2$, the robust problem can be approximated by the (SMBP) with $b_j = (\overline{A}_j - \underline{A}_j)^2$ and $D_2(\alpha) = \sqrt{-0.5 \ln(1 - \alpha)}$.*

The details of the proof are omitted for conciseness. In particular, the proof for $\mathcal{D}_1$ is analogous to an existing result in continuous optimization that converts linear programs with a chance constraint into a linear program with a convex second-order cone constraint (see El Ghaoui et al. 2003 and Calafiore and El Ghaoui 2006). The proof for $\mathcal{D}_2$ follows directly from the fact that Hoeffding's inequality applies for the infimum of the probability distributions.

We have shown that the (SMBP) problem is a good approximation for the bin-packing problem with chance constraints. For independent random variables with given mean and covariance, the approximation is exact,

and for distributions over independent bounded intervals, it yields a feasible solution. As discussed, the capacity constraint in the (SMBP) is replaced by the following equation, called the *modified capacity constraint*:

$$\sum_{j=1}^{N} \mu_j x_{ij} + D(\alpha)\sqrt{\sum_{j=1}^{N} b_j x_{ij}} \le V y_i. \quad (5)$$

One can interpret Equation (5) as follows. Each machine has a capacity V. Each job j consumes capacity μ_j in expectation as well as an additional buffer to account for the uncertainty. This buffer depends on two factors: (i) the variability of the job, captured by the parameter b_j, and (ii) the acceptable level of risk through $D(\alpha)$. The function $D(\alpha)$ is increasing in α, and therefore, we impose a stricter constraint as α approaches one by requiring the extra buffer to be larger. Equation (5) can also be interpreted as a risk measure. For each machine i, the total (random) load is $\sum_{j=1}^{N} A_j x_{ij}$. If we consider that μ_j represents the expectation and b_j corresponds to the variance, then $\sum_{j=1}^{N} \mu_j x_{ij}$ and $\sqrt{\sum_{j=1}^{N} b_j x_{ij}}$ correspond to the expectation and standard deviation of the total load on machine i, respectively. As a result, the right-hand side of Equation (5) can be interpreted as an adjusted risk utility, where $D(\alpha)$ is the degree of risk aversion of the scheduler. The additional amount allocated for job j can be interpreted as a safety buffer to account for the uncertainty and for the risk that the provider is willing to bear. In Section 5, we develop efficient methods to solve the (SMBP) with analytical performance guarantees.

2.4. Two Naive Approaches
In this section, we explore the limitations of two approaches that come to mind. The first is to rewrite the problem as a linear integer program (IP): the decision variables are all binary, and the nonlinearity in (SMBP) can actually be captured by common modeling techniques (see Online Appendix A). Unfortunately, solving this IP is not a viable option. As for the classical deterministic bin-packing problem, solving even moderately large instances with commercial solvers takes several hours. Moreover, applying the approach to smaller specific instances provides little insight about the assignment policy and how the value of α affects the solution. The second potential approach is to develop an algorithm for a more general problem: bin packing with general monotone submodular capacity constraints. Using some machinery and results from Goemans et al. (2009) and Svitkina and Fleischer (2011), we next show that it is, in fact, impossible to find a solution within any reasonable factor from optimal.

Theorem 1. *Consider the bin-packing problem with general monotone submodular capacity constraints for each machine. Then there exists no polynomial time algorithm with an approximation factor better (i.e., smaller) than $\frac{\sqrt{N}}{\ln(N)}$.*

The proof can be found in Online Appendix B. In the rest of this paper, we show that the (SMBP) is more tractable as it relates to a specific class of monotone submodular constraints that capture the structure of the chance-constrained problem.

3. Literature Review
This paper is related to different streams of literature. In the optimization literature, the problem of scheduling jobs on machines has been extensively studied, and the bin-packing problem is a common formulation. Hundreds of papers study the bin-packing problem, including many of its variations, such as 2-D packing (e.g., Pisinger and Sigurd 2005), linear packing, packing by weight or cost, etc. The basic bin-packing problem is NP-hard, and Delorme et al. (2016) provide a recent survey of exact approaches. However, several simple online algorithms are often used in practice for large-scale instances. A common variation is the problem in which jobs arrive online with sizes sampled independently from a known distribution with integer support and must be packed onto machines upon arrival. The size of a job is known when it arrives, and the goal is to minimize the number of nonempty machines (or, equivalently, minimize the total unused space or the *waste*). For this variation, the *sum-of-squares* heuristic represents the state-of-the-art approach. It is almost distribution-agnostic and nearly universally optimal for most distributions by achieving sublinear waste in the number of items seen (Csirik et al. 2006). In Gupta and Radovanovic (2015), the authors propose two algorithms based on gradient descent on a suitably defined Lagrangian relaxation of the bin-packing linear program that achieve additive $O(\sqrt{N})$ waste relative to the optimal policy (where N is the number of items). This line of works bounds the expected waste for general classes of job size distribution in an asymptotic sense.

Worst-case analysis of (deterministic) bin-packing solutions has received a lot of attention as well. Several efficient algorithms have been proposed that can be applied online and admit approximation guarantees in both online and off-line settings. The off-line version of the problem can be solved using $(1 + \epsilon)OPT + 1$ bins in linear time (de La Vega and Lueker 1981).[5] A number of heuristics can solve large-scale instances efficiently while guaranteeing a constant factor cost relative to optimal. For a survey on approximation algorithms for bin packing, see, for example, Coffman et al. (1996). Three such widely used heuristics are first fit (FF), next fit (NF), and best fit (BF) (see, e.g., Bays 1977,

Kenyon et al. 1996, and Keller et al. 2012). FF assigns the newly arrived job to the first machine that can accommodate it and purchases a new machine only if none of the existing ones can fit the new job. NF is similar to FF but continues to assign jobs from the current machine without going back to previous machines. BF uses a similar strategy but seeks to fit the newly arrived job to the machine with the smallest remaining capacity. Although one can show that these heuristics provide a two-approximation guarantee, improved factors were also developed under special assumptions. Dósa and Sgall (2013) provide a tight upper bound for the FF strategy, showing that it never needs more than $1.7OPT$ machines. The off-line version of the problem also received a lot of attention, and the best-fit-decreasing (BFD) and first-fit-decreasing (FFD) strategies are among the simplest (and most popular) heuristics. They operate like BF and FF but first rank all the jobs in decreasing order of size. Dósa (2007) shows that the tight bound of FFD is $(11/9)OPT + 6/9$.

Stochastic bin-packing models in which job durations are modeled as random variables are particularly relevant to this paper. Coffman et al. (1980) study the asymptotic and convergence properties of the NF online algorithm. Lueker (1983) considers the case in which job durations are drawn uniformly from intervals of the form $[a, b]$ and derive a lower bound on the asymptotic expected number of bins. However, unlike this type of asymptotic results in which jobs' sizes are known when scheduling occurs, we are interested in computing a solution that is feasible with high probability before observing the actual sizes. Our objective is to assign the jobs to as few machines as possible such that the set of jobs assigned to each machine satisfies the capacity constraint with some given probability (say 99%). In other words, we solve a stochastic optimization problem and analyze simple heuristic solutions to achieve this goal. To make the difference with the worst-case analysis clear, we note that the worst-case analysis becomes a special case of our problem when the objective probability threshold is set to 100% (instead of any number strictly less than one). The point of our paper is to exploit the stochastic structure of the problem to reduce the scheduling costs via overcommitment.

An additional common application of the bin-packing problem is surgery planning (sometimes also called operating room scheduling). In this application, the surgery duration is assumed to be stochastic, and the objective is to schedule the surgeries while minimizing the total overtime cost (see, e.g., the survey papers by Cardoen et al. 2010 and Deng et al. 2016). In this paper, however, our objective is to minimize the total number of machines instead of the total cost of packing. Denton et al. (2010) consider the problem of optimizing surgery allocation by minimizing the total

cost of opening a room plus the expected penalty cost of overtime. The authors solve a two-stage stochastic binary integer program based on finite samples of the random surgery durations. In this context, Shylo et al. (2012) was among the first to use chance constraints for restricting the overtime in surgery operating rooms. By assuming that the surgery durations follow a multivariate Gaussian distribution, they reformulate the chance-constrained model as an equivalent semidefinite program based on convex analysis of probabilistic constraints.

Finally, bin-packing models find an application in bandwidth allocation for high-speed networks. For example, Kleinberg et al. (2000) relate stochastic bin packing to allocating bandwidth for bursty connections in high-speed networks and propose approximation algorithms for its online chance-constrained variant. The authors also show that chance constraints in binary knapsack problems are equivalent to those in bin packing. The knapsack problem with chance constraints was also extensively studied in the literature (see, e.g., Goyal and Ravi 2010 and Han et al. 2016).

A common way to solve the chance-constrained bin packing problem is to apply the sample average approximation (SAA) approach to approximate the problem as a mixed integer linear program (see Luedtke et al. (2010) and the references therein). In this approach, one needs to know the full distributional information. The recent work by Zhang et al. (2016) considers an off-line algorithm to solve bin packing with chance constraints when knowing only the first two moments. The authors show that the problem can be reformulated as a 0–1 second-order cone program and derive extended polymatroid inequalities to strengthen the formulation. They also demonstrate computationally that the branch-and-cut algorithm with extended polymatroid inequalities scales very well as the problem size grows. However, no theoretical guarantee on the worst-case performance is provided.

This paper is also related to the robust optimization literature and especially to distributionally robust optimization. In this context, the goal is to solve an optimization problem in which the input parameter distribution belongs to a family of distributions that share some properties (e.g., all the distributions with the same mean and covariance matrix) and consider the worst-case within the given family (concrete examples were presented in Section 2.3). Examples of such studies include El Ghaoui et al. (2003), Bertsimas and Popescu (2005), Calafiore and El Ghaoui (2006), and Delage and Ye (2010). These papers aim to convert linear or convex (continuous) optimization problems with a chance constraint into tractable formulations. Our paper shares a similar motivation but considers an integer problem. To the best of our knowledge, this paper is among the first to develop efficient algorithms

with constant approximation guarantees for the online bin-packing problem with capacity chance constraints.

Large-scale cluster management in general is an important area of computer systems research. Verma et al. (2015) provide a full, modern example of a production system. Much experimental studies seek to evaluate the real-world performance of bin-packing heuristics that also account for factors such as adverse interactions between jobs and the presence of multiple contended resources (see, for example, Lee et al. 2011 and Roytman et al. 2013). Although modeling these aspects is likely to complement the resource savings achieved with the stochastic model we propose, these papers capture fundamentally different efficiency gains arising from technological improvements and idiosyncratic properties of certain types (or combinations) of resources. In this paper, we limit our attention to the benefit and practicality of machine overcommitment in the case in which a single key resource is in short supply.

4. Results and Insights for Special Cases

In this section, we consider the (SMBP) for given μ_j, b_j, N, and $D(\alpha)$. Our goals are to (i) develop efficient approaches to solve the problem, (ii) to draw some insights on how to schedule the jobs, and (iii) to study the effect of the different parameters on the outcome. This will ultimately allows us to understand the impact of overcommitment in resource allocation problems.

4.1. Identical Distributed Jobs

We consider the symmetric setting in which all A_j have the same distribution, such that $\mu_j = \mu$ and $b_j = b$ in the (SMBP). By symmetry, we only need to find the number of jobs n to assign to each machine. Because all the jobs are interchangeable, our goal is to assign as many jobs as possible in each machine. Thus, we want to pick the largest value n that satisfies constraint (5):

$$D(\alpha)^2 \leq \frac{[V - n\mu]^2}{nb}.$$

For a given α, this is the largest integer smaller than

$$n(\alpha) = \frac{V}{\mu} + \frac{1}{2\mu^2}\left[bD(\alpha)^2 - \sqrt{b^2D(\alpha)^4 + 4bD(\alpha)^2 V\mu}\right]. \quad (6)$$

• For a given α, the number of jobs $n(\alpha)$ increases with V/μ. Indeed, because μ represents the expected job size, increasing the ratio V/μ is equivalent to increasing the number of "average" jobs a machine can host. If the jobs are smaller or the machines larger, one can fit more jobs per machine as expected.

• For a given V/μ, $n(\alpha)$ is a nonincreasing function of α. When α increases, it means that we enforce the capacity constraint in a stricter manner (recall that $\alpha = 1$ corresponds to the case without overcommitment). As a result, the number of jobs per machine cannot increase.

• For given α and V/μ, $n(\alpha)$ is a nonincreasing function of b. Recall that b corresponds to some measure of spread (the variance in the Gaussian setting and the range for distributions with bounded support). Therefore, when b increases, the jobs' resource usage is more volatile, and hence, a larger buffer is needed. Consequently, the number of jobs cannot increase when b grows.

• For given α and V, $n(\alpha)$ is nonincreasing in $\sqrt{b}/\mu$. The quantity $\sqrt{b}/\mu$ represents the coefficient of variation of the job size in the Gaussian case or a similarly normalized measure of dispersion in other cases. Consequently, one should be able to fit fewer jobs as the variability increases.

The simple case of identically distributed jobs allows us to understand how the different factors affect the number of jobs that one can assign to each machine. In Figure 1, we plot Equation (6) for an instance with $\overline{A} = 1$, $\underline{A} = 0.3$, $\mu = 0.65$, $V = 30$, and $0.5 \leq \alpha < 1$. The large dot at $\alpha = 1$ represents the case without overcommitment. Interestingly, when the value of α approaches one, the benefit of allowing a small probability of violating the capacity constraint is significant so that one can increase the number of jobs per machine. In this case, when $\alpha = 1$, we can fit 30 jobs per machine, but when $\alpha = 0.992$, we can fit 36 jobs, hence, an improvement of 20%. Note that this analysis guarantees that the capacity constraint is satisfied with at least probability α. As we show in Section 7 for many instances, the capacity constraint is satisfied with an even higher probability. Alternatively, one can plot α as a function of n (see Figure 2a for an example with different values for V/μ) or n as a function of $\sqrt{b}/a$ for different values of α (see Figure 2b). As expected, the benefit of overcommitting increases with V/μ. In our example, when $V/\mu = 25$, by

Figure 1. (Color online) Parameters: $\overline{A} = 1$, $\underline{A} = 0.3$, $\mu = 0.65$, $V = 30$
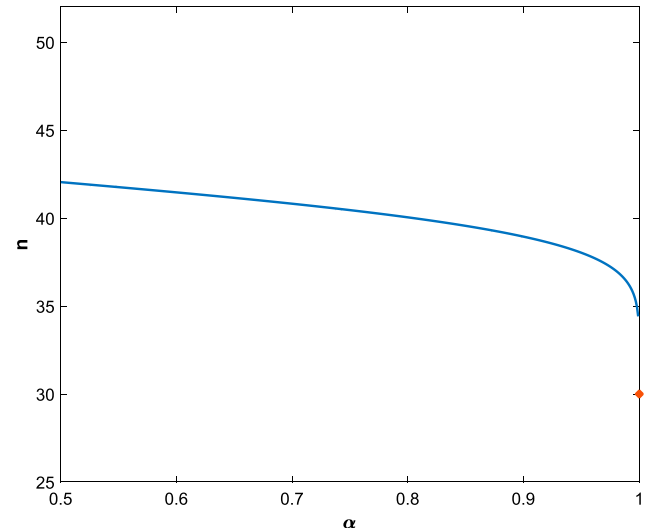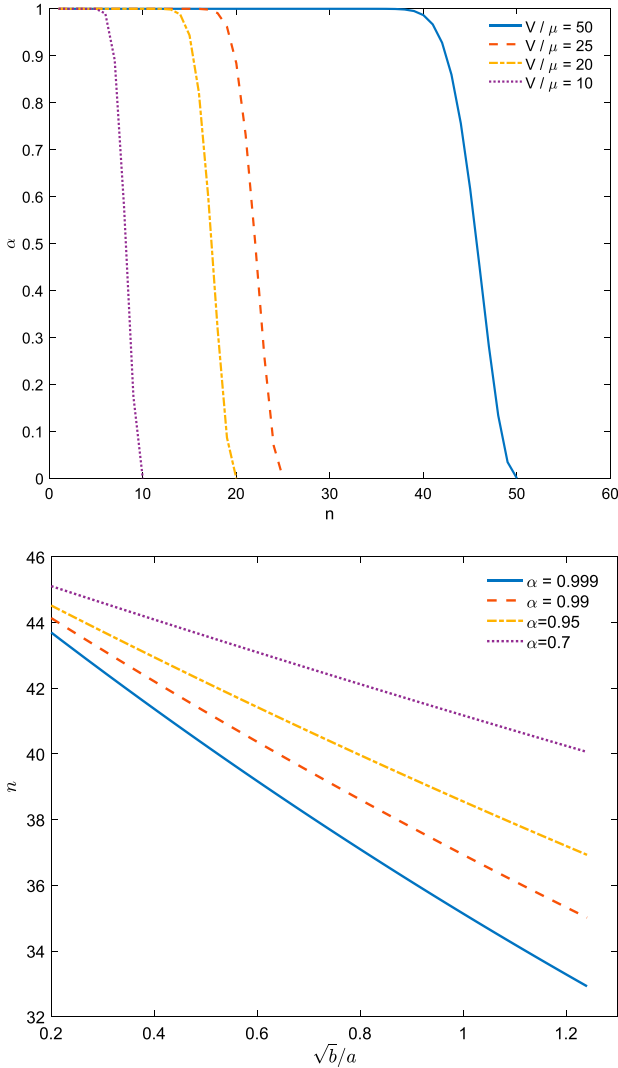
Figure 2. (Color online) Example for Identically Distributed Jobs

scheduling jobs according to $\overline{A}$ (i.e., no overcommitment), we can schedule 14 jobs, but if we allow a 0.1% violation probability, we can schedule 17 jobs. Consequently, by allowing 0.1% chance of violating the capacity constraint, one can save more than 20% in costs. Next, we discuss the case with a small number of different classes of job distributions.

4.2. Small Number of Job Distributions
We now consider the case in which the random variables A_j can be clustered in few different categories. For instance, suppose standard clustering algorithms are applied to historical data to treat similar jobs as a single class with a usage distribution. A concrete example is a setting with four types of jobs: (i) large jobs with no variability (μ_j is large and $b_j = 0$), (ii) small jobs with no variability (μ_j is small and $b_j = 0$), (iii) large jobs with

high variability (both μ_j and b_j are large), and (iv) small jobs with high variability (μ_j is small and b_j is large). In other words, we have N jobs, and they all are from one of the four types with given μ_j and b_j. The result for this setting is summarized in the following observation (the details can be found in Online Appendix C).

Observation 1. In the case in which the number of different job classes is not too large, one can solve the problem efficiently as a cutting-stock problem.

The resulting cutting-stock problem (see formulation (12) in Online Appendix C) is well studied in many contexts (see Gilmore and Gomory 1961 for a classical approach based on linear programming or the recent survey of Delorme et al. 2016). For example, one can solve the LP relaxation of (12) and round the fractional solution. This approach can be very useful for cases in which the cloud provider has enough historical data and when the jobs can all be grouped in a small number of different clusters. However, grouping all customer job profiles into a small number of classes, each described by a single distribution, is often unrealistic. For example, virtual machines are typically sold with 1, 2, 4, 8, 16, 32, or 64 CPU cores, each with various memory configurations, to a variety of customers with disparate use-cases (different usage means and variability). Unfortunately, if one decides to use a large number of job classes, solving a cutting-stock problem is not scalable. In addition, this approach requires advance knowledge of the number of jobs in each class and, hence, cannot be applied in an online fashion.

5. Online Constant Competitive Algorithms
In this section, we analyze the performance of a large class of algorithms for the online version of (SMBP). We note that the same guarantees hold for the off-line case as it is just a simpler version of the problem. We also present a refined result for the off-line problem in Section 6.1.

5.1. Lazy Algorithms Are $\frac{8}{3}$-Competitive
An algorithm is called *lazy* if it does not purchase a new machine unless necessary.

Definition 1. We call an online algorithm *lazy* if, upon arrival of a new job, it assigns the job to one of the existing (already purchased) machines given the capacity constraints are not violated. In other words, the algorithm purchases a new machine if and only if none of the existing machines can accommodate the newly arrived job.

Several commonly used algorithms fall into this category, for example, first fit and best fit. Let $\mathbb{O}PT$ be the optimal objective, that is, the minimum number of machines needed to serve all the jobs $\{1, 2, \cdots, N\}$.

Recall that each job $1 \leq j \leq N$ has two characteristics: μ_j and b_j, which represent the mean and the uncertain part of job j, respectively. For a set of jobs S, we define the corresponding cost $\mathbb{C}ost(S)$ to be $\sum_{j \in S} \mu_j + \sqrt{\sum_{j \in S} b_j}$. Without loss of generality, we can assume (by normalization of all μ_j and b_j) that the capacity of each machine is 1 and that $D(\alpha)$ is also normalized to one. We call a set S feasible if its cost is at most 1. In the following theorem, we show that any lazy algorithm yields a constant approximation for the (SMBP) problem.

Theorem 2. *Any lazy algorithm* ALG *purchases at most* $\frac{8}{3}\mathbb{O}PT$ *machines, where* $\mathbb{O}PT$ *is the optimum number of machines to serve all jobs.*

The proof can be found in Online Appendix D. Theorem 2 derives an approximation guarantee of 8/3 for any lazy algorithm. In many practical settings, one can further exploit the structure of the set of jobs and design algorithms that achieve better approximation factors. For example, if some jobs are usually larger relative to others, one can incorporate this knowledge into the algorithm. We next describe the main intuitions behind the 8/3 upper bound. In the proof of Theorem 2, we have used the following two main proof techniques:

- First, we show a direct connection between the feasibility of a set S and the sum $\sum_{j \in S}(\mu_j + b_j)$. In particular, we prove that $\sum_{j \in S}(\mu_j + b_j) \leq 1$ for any feasible set and greater than 3/4 for any infeasible set. Consequently, $\mathbb{O}PT$ cannot be less than the sum of $\mu_j + b_j$ for all jobs. The gap of 4/3 between the two bounds contributes partially to the final upper bound of 8/3.
- Second, we show that the union of jobs assigned to any pair of machines by a lazy algorithm is an infeasible set so that their sum of $\mu_j + b_j$ should exceed 3/4. One can then find $m/2$ disjoint pairs of machines and obtain a lower bound of 3/4 for the sum $\mu_j + b_j$ for each pair. The fact that we achieve this lower bound for every pair of machines (and not for each machine) contributes another factor of 2 to the approximation factor, resulting in $\frac{4}{3} \times 2 = \frac{8}{3}$.

Note that the second loss of a factor of 2 follows from the fact that the union of any two machines forms an infeasible set and nothing stronger. In particular, all machines could potentially have a cost of $1/2 + \epsilon$ for a very small ϵ and make the preceeding analysis tight. Nevertheless, if we assume that each machine is nearly full (i.e., has $\mathbb{C}ost$ close to 1), we can refine the approximation factor.

Theorem 3. *For any* $0 \leq \epsilon \leq 0.3$, *if the lazy algorithm* ALG *assigns all the jobs to m machines such that* $\mathbb{C}ost(S_i) \geq 1 - \epsilon$ *for every* $1 \leq i \leq m$, *we have* $m \leq (\frac{4}{3} + 3\epsilon)\mathbb{O}PT$, *that is, a* $(\frac{4}{3} + 3\epsilon)$ *approximation guarantee.*

Proof. To simplify the analysis, we denote β to be $1 - \epsilon$. For a set S_i, we define $x = \sum_{j \in S_i} \mu_j$ and $y = \sqrt{\sum_{j \in S_i} b_j}$. Because $\mathbb{C}ost(S_i)$ is at least β, we have $x + y \geq \beta$. Assuming $x \leq \beta$, we have

$$\sum_{j \in S_i}(\mu_j + b_j) = x + y^2 \geq x + (\beta - x)^2$$

$$= \left(x - \frac{2\beta - 1}{2}\right)^2 + \beta - \frac{1}{4} \geq \beta - \frac{1}{4} = \frac{3}{4} - \epsilon,$$

where the first equality is by the definition of x and y, the first inequality holds by $x + y \geq \beta$, and the rest are algebraic manipulations. For $x > \beta$, we also have $\sum_{j \in S_i}(\mu_j + b_j) \geq x > \beta > \frac{3}{4} - \epsilon$. We conclude that $\sum_{j=1}^{N}(\mu_j + b_j) \geq m \times (\frac{3}{4} - \epsilon)$. We also know that $\mathbb{O}PT \geq \sum_{j=1}^{N}(\mu_j + b_j)$, which implies that $m \leq \frac{\mathbb{O}PT}{3/4 - \epsilon} \leq \mathbb{O}PT(\frac{4}{3} + 3\epsilon)$, where the last inequality holds for $\epsilon \leq 0.3$. $\square$

A particular setting in which the condition of Theorem 3 holds is when the capacity of each machine is large compared with all jobs, that is, $\max_{1 \leq j \leq N} \mu_j + \sqrt{b_j}$ is at most ϵ. In this case, for each machine $i \neq m$ (except the last purchased machine), we know that there exists a job $j \in S_m$ (assigned to the last purchased machine m) such that the algorithm could not assign j to machine i. This means that $\mathbb{C}ost(S_i \cup \{j\})$ exceeds one. Because $\mathbb{C}ost$ is a subadditive function, we have $\mathbb{C}ost(S_i \cup \{j\}) \leq \mathbb{C}ost(S_i) + \mathbb{C}ost(\{j\})$. We also know that $\mathbb{C}ost(\{j\}) \leq \epsilon$, which implies that $\mathbb{C}ost(S_i) > 1 - \epsilon$.

Remark 1. As discussed, there are two main sources for losses in the approximation factor: nonlinearity of the cost function, which contributes up to 4/3, and machines being only partially full, which induces an extra factor of 2, implying the 8/3 approximation guarantee. In the classical bin-packing case (i.e., $b_j = 0$ for all j), the cost function is linear, and the nonlinearity losses in the approximation factor fade. Consequently, we obtain that (i) Theorem 2 reduces to a two-approximation factor and (ii) Theorem 3 reduces to a $(1 + \epsilon)$-approximation factor, which are both consistent with known results from the literature on the classical bin-packing problem.

Theorem 3 improves the bound for the case in which each machine is almost full. However, in practice, machines are often not full. We next derive a bound as a function of the minimum number of jobs assigned to the machines.

5.2. Algorithm First–Fit is $\frac{9}{4}$-Competitive

So far, we considered the general class of lazy algorithms. One popular algorithm in this class (both in the literature and in practice) is First–Fit. By exploiting the structural properties of allocations made by First–Fit, we can provide a better competitive ratio of $\frac{9}{4} < \frac{8}{3}$. Recall

that, upon the arrival of a new job, First–Fit purchases a new machine if the job does not fit in any of the existing machines. Otherwise, it assigns the job to the first machine (based on a fixed ordering, such as machine IDs) that it fits in. This algorithm is simple to implement and very well studied in the bin-packing context. First, we present an extension of Theorem 2 for the case in which each machine has at least K jobs.

Corollary 1. *If the First–Fit algorithm assigns jobs such that each machine hosts at least K jobs, the number of purchased machines does not exceed $\frac{4}{3}(1 + \frac{1}{K})\mathbb{O}PT$, where $\mathbb{O}PT$ is the optimum number of machines to serve all jobs.*

One can prove Corollary 1 in a similar fashion as the proof of Theorem 2 and using the fact that jobs are assigned using First–Fit (the details are omitted for conciseness). For example, when $K = 2$ (resp. $K = 5$), we obtain a two (resp. 1.6) approximation. We next refine the approximation factor for the problem by using the First–Fit algorithm.

Theorem 4. *The number of purchased machines by First–Fit for any arrival order of jobs is no more than $\frac{9}{4}\mathbb{O}PT + 1$.*

The proof can be found in Online Appendix E. We note that the approximation guarantees developed in this section do not depend on the factor $D(\alpha)$ and on the specific definition of the parameters μ_j and b_j. In addition, as we show computationally in Section 7, the performance of this class of algorithms is not significantly affected by $D(\alpha)$.

5.3. Insights on Job Scheduling
We next show that ensuring the following two guidelines in any allocation yields optimal solutions:
- Filling up each machine completely so that no other job fits in, that is, making each machine's $\mathbb{C}ost$ equal to 1.
- Each machine contains a set of *similar* jobs (defined formally next).

We formalize these properties in more detail and show how one can achieve optimality by satisfying these two conditions. We call a machine *full* if $\sum_{j \in S}\mu_j + \sqrt{\sum_{j \in S}b_j}$ is equal to 1 (recall that the machine capacity is normalized to one without loss of generality), where S is the set of jobs assigned to the machine. Note that it is not possible to assign any additional job (no matter how small the job is) to a full machine. Similarly, we call a machine *ϵ-full* if the cost is at least $1 - \epsilon$, that is, $\sum_{j \in S}\mu_j + \sqrt{\sum_{j \in S}b_j} \geq 1 - \epsilon$. We define two jobs to be *similar* if they have the same b/μ ratio. Note that similar jobs can have different values of μ and b. We say that a machine is *homogeneous* if it only contains similar jobs. In other words, if the ratio b_j/μ_j is the same for all the jobs j assigned to this machine. By convention, we define b_j/μ_j to be $+\infty$ when $\mu_j = 0$. In addition,

we introduce the relaxed version of this property: we say that two jobs are *δ-similar* if their b/μ ratios differ by at most a multiplicative factor of $1 + \delta$. A machine is called *δ-homogeneous* if it only contains δ-similar jobs (i.e., for any pair of jobs j and j' in the same machine, $\frac{b_j/\mu_j}{b_{j'}/\mu_{j'}}$ is at most $1 + \delta$).

Theorem 5. *For any $\epsilon \geq 0$ and $\delta \geq 0$, consider an assignment of all jobs to some machines with two properties: (i) each machine is ϵ-full and (ii) each machine is δ-homogeneous. Then, the number of purchased machines in this allocation is at most $\frac{\mathbb{O}PT}{(1-\epsilon)^2(1-\delta)}$.*

The proof can be found in Online Appendix F. In summary, we proposed an easy-to-follow recipe to schedule jobs on machines. Each arriving job is characterized by two parameters μ_j and b_j. Upon arrival of a new job, the cloud provider can compute the ratio $r_j = b_j/\mu_j$. Then one can decide on a few "buckets" for the different values of r_j, depending on historical data and performance restrictions. Finally, the cloud provider will assign jobs with similar ratios to the same machines while trying to fill in machines as much as possible. If one manages to make all the machines ϵ-full while packing similar jobs in each machine (i.e., δ homogeneous), the result of Theorem 5 suggests that such a strategy will yield a good performance in terms of minimizing the number of machines.

6. Extensions
In this section, we present two extensions of the problem considered in this paper.

6.1. Off-Line 2-Approximation Algorithm
Consider the off-line version of the (SMBP). In this case, all the N jobs already arrived, and one has to find a feasible schedule so as to minimize the number of machines. We propose the algorithm Local–Search that iteratively reduces the number of purchased machines and uses ideas inspired from First–Fit to achieve a two-approximation for the offline problem. Algorithm Local–Search starts by assigning all the jobs to machines arbitrarily and then iteratively refines this assignment. Suppose that each machine has a unique identifier number. We next introduce some notation before presenting the update operations. Let a be the number of machines with only one job, A_1 be the set of these a machines, and S_1 be the set of jobs assigned to these machines. Note that this set changes throughout the algorithm with the update operations. We say that a job $j \notin S_1$ is *good* if it fits in at least 6 of the machines in the set A_1.[6] In addition, we say that a machine is *large* if it contains at least 5 jobs, and denote the set of large machines by A_5. We say that a machine is *medium size* if it contains 2, 3, or 4 jobs and denote the set of medium machines by $A_{2,3,4}$. We call a medium-size machine

critical if it contains (exactly) one job that fits in none of the machines in A_1 and the rest of the jobs in this machine are all good. We present the update operations of LOCAL–SEARCH in Algorithm 1.

Algorithm 1. LOCAL–SEARCH

Input: All the N jobs already arrived.

Procedure

1. Find a job j in machine i (i is the machine identifier number) and assign it to some other machine $i' < i$ if feasible (the outcome will be similar to FIRST–FIT).

2. Find a medium-size machine i that contains one job j_1 that fits in at least one machine in A_1 and the rest of the jobs in i are all good. Let $j_2, \cdots, j_\ell$ ($2 \le \ell \le 4$) be the other jobs in machine i. First, assign j_1 to one machine in A_1 that it fits in. If there are multiple such machines, select one of them arbitrarily. Then, assign j_2 to a different machine in A_1 that it fits in. There should be at least 5 ways to do so because j_2 is a good job. We continue this process until all the jobs in machine i (there are at most 4 of them) are assigned to distinct machines in A_1 and they all fit in their new machines. This way, we release machine i and reduce the number of purchased machines by one.

3. Find two critical machines i_1 and i_2. Let j_1 and j_2 be the only jobs in these two machines that fit in no machine in A_1. If both jobs fit and form a feasible assignment in a new machine, we purchase a new machine and assign j_1 and j_2 to it. Otherwise, we do not change anything and ignore this update step. There are at most $3 \times 2 = 6$ other jobs in these two machines because both are medium machines. In addition, the rest of the jobs are all good. Therefore, similar to the previous case, we can assign these jobs to six distinct machines in A_1. Note that any number less than 6 in the definition of good jobs will not suffice for this part of the algorithm to hold. This way, we release machines i_1 and i_2 and purchase a new machine. So, in total, we reduce the number of purchased machines by one.

Before presenting the performance result of LOCAL–SEARCH, we first motivate our choice of splitting the machines into small, medium, and large machines. According to our proof (see Online Appendix G), if there are only small machines, the LOCAL–SEARCH algorithm is optimal. Indeed, it becomes clear that the first iteration of the LOCAL–SEARCH algorithm ensures that if two jobs fit in one machine, they will be merged. On the other hand, if there are no small machines, we can show that the algorithm yields a two-approximation using the same argument as we used before (see Corollary 1, when $K = 2$). Consequently, we remain with the case in which we have a mixture of small and nonsmall machines. In this case, the argument presented in our proof yields a guarantee that is worse than two. Therefore, we need to find a way to compensate for

this gap and improve the total approximation guarantee to 2. This is the main motivation of introducing medium and large machines. By using our proof technique, if we only have large machines, one can show an approximation factor of 8/5, which is indeed strictly better than 2. We then exploit this improvement with respect to 2 to compensate for the deficit we have obtained from the mixed case of small and nonsmall machines. We are now ready to analyze this LOCAL–SEARCH algorithm that also borrows ideas from FIRST–FIT. We next show that the number of purchased machines is at most $2\mathbb{O}PT + O(1)$, that is, a two-approximation.

Theorem 6. *Algorithm* LOCAL–SEARCH *terminates after at most* N^3 *operations (where N is the number of jobs) and purchases at most* $2\mathbb{O}PT + 11$ *machines.*

The proof can be found in Online Appendix G. We conclude this section by comparing our results to the classical (deterministic) bin-packing problem. In classical bin packing, there are folklore polynomial time approximation schemes (see section 10.3 in Albers and Souza 2011) that achieve a $(1 - \epsilon)$-approximation factor by proposing an offline algorithm based on clustering the jobs into $1/\epsilon^2$ groups and treating them as equal-size jobs. Using dynamic programming techniques, one can solve the simplified problem with $1/\epsilon^2$ different job sizes in time $O(n^{\text{poly}(1/\epsilon)})$. In addition to the inefficient time complexity that makes such algorithms less appealing for practical purposes, one cannot generalize the same ideas to our setting. The main obstacle is the lack of total ordering among the different jobs. In classical bin packing, the jobs can be sorted based on their sizes. However, this is not true in our case because the jobs have two dimensional requirements (μ_j and b_j).

6.2. Alternative Constraints

Recall that in the (SMBP), we used the modified capacity constraint (5). Instead, one can consider the following family of constraints, parametrized by $0.5 \le p \le 1$:

$$\sum_{j=1}^{N} \mu_j x_{ij} + D(\alpha) \left(\sum_{j=1}^{N} b_j x_{ij} \right)^p \le V y_i.$$

For conciseness, the results of this case can be found in Online Appendix H.

7. Computational Experiments

In this section, we test and validate our results by solving the (SMBP) and by examining the impact on the number of purchased machines. We use realistic workload data inspired by Google Compute Engine and show how our model and algorithms can be applied in an operational setting.

7.1. Setting and Data

We use simulated workloads of 1,000 jobs (virtual machines) using a realistic VM size distribution (see Table 1). Typically, the GCE workload is composed of a mix of VM sizes from virtual machines belonging to cloud customers. These jobs can have highly varying workloads, including some large ones and many smaller ones.[7] We assume that each VM arrives to the cloud provider with a requested size (i.e., number of CPU cores), sampled from the distribution in Table 1.

In this context, the average utilization is typically low, but in many cases, the utilization can be highly variable over time. Although we decided to keep a similar VM size distribution as observed in a production data center, we also fitted parametric distributions to match the mean and variance of the measured usage. This allows us to obtain a parametric model that we could vary for simulation. We consider two different cases for the actual utilization as a fraction of the requested job size: either a Bernoulli (binary) distribution or a truncated Gaussian. As discussed in Section 2.3, we assume that each job j has lower and upper utilization bounds, $\underline{A}_j$ and $\overline{A}_j$. We sample $\underline{A}_j$ uniformly in $[0.3, 0.6]$, and $\overline{A}_j$ in the range $[0.7, 1.0]$. In addition, we uniformly sample μ'_j and $\sigma'_j \in [0.1, 0.5]$ for each VM to serve as the parameters of the truncated Gaussian (not to be confused with its true mean and standard deviation, μ_j and σ_j). For the Bernoulli (binary) case, $\mu'_j = \mu_j$ determines the probabilities of the realization corresponding to the lower and upper bounds.

For each workload of 1,000 VMs generated in this manner, we solve the online version of the (SMBP) by implementing the best-fit heuristic, using one of the three different variants for $D(\alpha)$ and b_j. We solve the problem for various values of α ranging from 0.5 to 0.99999. More precisely, when a new job arrives, we compute the modified capacity constraint in Equation (5) for each already purchased machine and assign the job to the machine with the smallest available capacity that can accommodate it.[8] If the job does not fit in any of the already purchased machines, the algorithm opens a new machine. We consider the three variations of the (SMBP) discussed earlier:

- The Gaussian case introduced in (2) with $b_j = \sigma_j^2$ and $D(\alpha) = \Phi^{-1}(\alpha)$. This is now also an approximation to the (BPCC) because the true distributions are truncated Gaussian or Bernoulli.
- The Hoeffding's inequality approximation introduced in (3) with $b_j = (\overline{A}_j - \underline{A}_j)^2$ and $D(\alpha) = \sqrt{-0.5\ln(1-\alpha)}$.

Table 1. Example Distribution of VM Sizes (Number of CPU Cores) in a Google Data Center

Number of cores	1	2	4	8	16	32
% VMs	36.3	13.8	21.3	23.1	3.5	1.9

This is equivalent to the distributionally robust approach with the family of distributions $\mathscr{D}_2$.

- The distributionally robust approximation with the family of distributions $\mathscr{D}_1$ with $b_j = \sigma_j^2$ and $D_1(\alpha) = \sqrt{\alpha/(1-\alpha)}$.

7.2. Linear Benchmarks

We also implement the following four benchmarks, which consist of solving the classical (DBP) problem. First, we have

- No overcommitment: This is equivalent to setting $\alpha = 1$ in the (SMBP) or solving the (DBP) with sizes $\overline{A}_j$.

Three other heuristics are obtained by replacing the square-root term in constraint (5) by a linear term; specifically, we replace the constraint with

$$\sum_{j=1}^{N} \mu_j x_{ij} + D(\alpha) \sum_{j=1}^{N} \sqrt{b_j x_{ij}} = \sum_{j=1}^{N} \left(\mu_j + D(\alpha)\sqrt{b_j}\right) x_{ij} \le V y_i,$$

$$(7)$$

where the equality follows from the fact that the variables x_{ij} are binary so that $\sqrt{x_{ij}} = x_{ij}$, and hence, we have $\sum_{j=1}^{N} \sqrt{b_j x_{ij}} = \sum_{j=1}^{N} \sqrt{b_j} x_{ij}$. Equation (7) allows us to obtain

- The linear Gaussian heuristic that mimics the Gaussian approximation in (2).
- The linear Hoeffding's heuristic that mimics the Hoeffding's approximation in (3).
- The linear robust heuristic that mimics the distributionally robust approach with $\mathscr{D}_1$.

Note that the linearized constraint (7) is more restrictive for a fixed value of α (by concavity of the square root), but we naturally vary the value of α in our experiments. We do not expect these benchmarks to outperform our proposed method because they do not capture the risk-pooling effect from scheduling jobs on the same machine. They do, however, still reflect different relative amounts of "buffer" beyond the expected utilization of each job because of the usage uncertainty.

The motivation behind the linear benchmarks lies in the fact that the problem is reduced to the standard (DBP) formulation, which admits efficient implementations for many heuristics. For example, the best-fit algorithm can run in time $O(N \log N)$ by maintaining a list of open machines sorted by the slack left on each machine (see Johnson 1974 for more details). In contrast, our best-fit implementation with the nonlinear constraint (5) takes time $O(N^2)$ because we evaluate the constraint for each machine when each new job arrives. Practically, in cloud VM scheduling systems, this quadratic-time approach may be preferred anyway because it generalizes to more complex "scoring" functions that also take into account additional factors besides the remaining capacity, such as multiple resource dimensions and correlation between jobs (see, e.g., Verma et al. 2015). In addition, the

computational cost could be mitigated by dividing the data center into smaller "shards," each consisting of a fraction of the machines, and then trying to assign each incoming job only to the machines in one shard. For example, in our experiments we found that there was little performance advantage in considering sets of more than 1,000 jobs at a time. Nevertheless, our results show that even the linear benchmarks may provide substantial savings (relative to the no-overcommitment policy) while only requiring minor changes relative to classical algorithms: instead of $\bar{A}_j$, we simply use job sizes defined by μ_j, b_j, and α.

7.3. Results and Comparisons

We compare the seven different methods in terms of the number of purchased machines and show that, in most cases, our approach significantly reduces the number of machines needed. We consider two physical machine sizes: 32 and 72 cores. As expected, larger machines achieve a greater benefit from the risk pooling. We draw 50 independent workloads, each composed of 1,000 VMs. For each workload, we schedule the jobs using best fit and report the average number of machines needed across the 50 workloads. Finally, we compute the probability of capacity violation as follows. For each machine used to schedule each of the workloads, we draw 5,000 utilization realizations (either from a sum of truncated Gaussian or a sum of Bernoulli distributions), and we count the number of realizations in which the total usage on a machine exceeds capacity.

The sample size was chosen so that our results reflect an effect that is measurable in a typical data center. Because our workloads require on the order of 100 machines each, this corresponds to roughly

$50 \times 100 \times 5,000 = 25,000,000$ individual machine-level samples. Seen another way, we schedule $50 \times 1,000 = 50,000$ jobs and collect 5,000 data points from each. Assuming a sample is recorded every 10 minutes, this corresponds to a few days of traffic even in a small data center with less than 1,000 machines.[9] The sample turns out to yield very stable measurements, and defining appropriate service level indicators is application-dependent and beyond the scope of this paper, so we do not report confidence intervals or otherwise delve into statistical measurement issues.

In Figure 3, we plot the average number of machines needed as a function of the probability that a constraint is violated, in the case in which the data center is composed of 72 CPU core machines. Each point in the curves corresponds to a different value of the parameter α. Without overcommitment, we need an average of more than 54 machines to serve all the jobs. By allowing a small chance of violation, say a 0.1% risk (or, equivalently, a 99.9% satisfaction probability), we only need 52 machines for the Bernoulli usage and 48 machines for the truncated Gaussian usage. If we allow a 1% chance of violation, we then only need 50 and 46 machines, respectively. The table of Figure 4 summarizes the relative savings, which amount to 4.5% and 11.5% with a 0.1% risk and to 8% and 14% with a 1% risk for the Bernoulli and truncated Gaussian usages, respectively.

Figure 3 shows that all three variations (Gaussian, Hoeffding's, and the distributionally robust) yield very similar results. This suggests that the results are robust to the method and the parameters. The same is true for the linear benchmarks although they perform worse as expected. We remark that although the final

Figure 3. (Color online) Average Number of 72-Core Machines Needed to Schedule a Workload vs. the Probability that Any Given Machine's Realized Load Exceeds Capacity
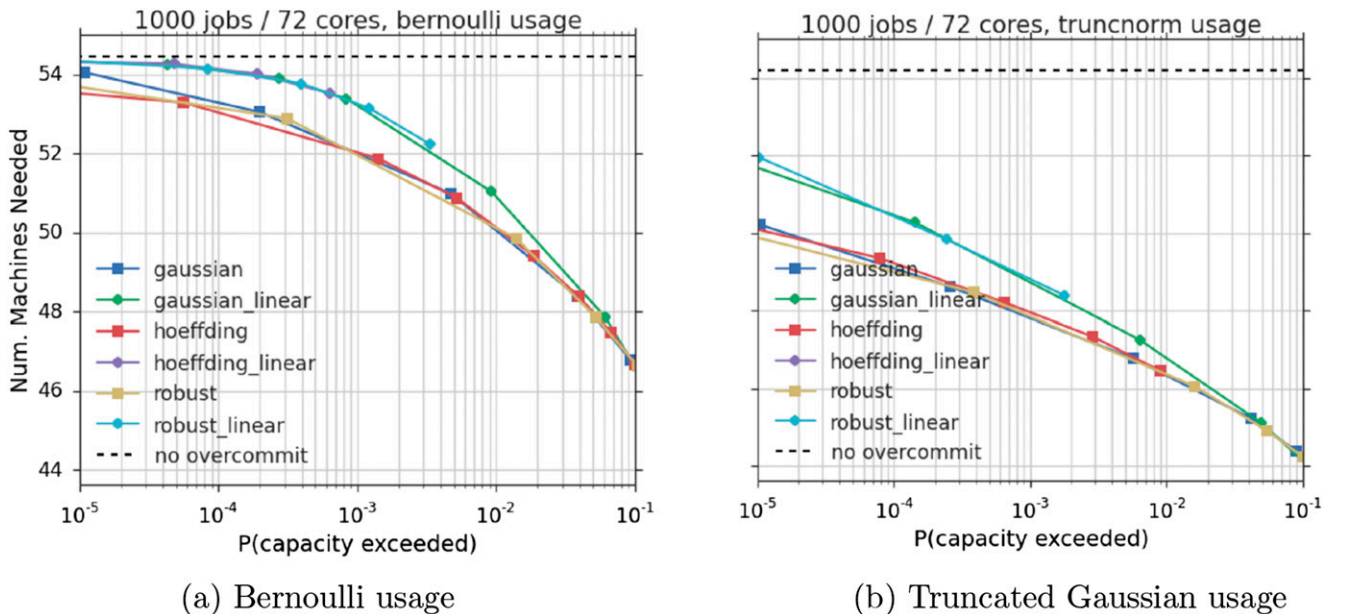

(a) Bernoulli usage

(b) Truncated Gaussian usage

Figure 4. Percentage Savings Resulting from Overcommitment for Two CPU Usage Distributions, Using the Three Proposed Variants of the Chance Constraint

	Algorithm	gaussian		gaussian_linear		hoeffding		robust	
	Capacity	32	72	32	72	32	72	32	72
Usage	P(sat.)								
Bernoulli	0.9999	0.2	2.1	0.1	0.7	0.7	2.6	0.3	2.4
	0.9990	1.5	4.5	0.5	2.3	1.6	4.5	1.9	4.6
	0.9900	4.5	8.1	1.9	6.5	4.2	7.9	4	8.0
	0.9500	8.2	12.0	5.8	11.5	8.0	11.9	8	12.0
Trunc. Gaussian	0.9999	5.7	9.4	1.1	6.9	5.7	9.2	-	9.6
	0.9990	8.1	11.8	2.8	10.1	7.8	11.5	8.2	11.7
	0.9900	11.2	14.5	8.2	13.7	9.8	13.4	11	14.4
	0.9500	14.4	16.9	13.8	16.8	9.8	13.4	14.3	17.0

Note. The linear Gaussian benchmark is shown for comparison.

performance trade-off is nearly identical, for a particular value of α, the achieved violation probabilities vary greatly. For example, with $\alpha = 0.9$ and the truncated normal distribution, each constraint was satisfied with probability 0.913 when using the Gaussian approximation but with much higher probabilities (0.9972 and 0.9998) for the Hoeffding and robust approximations. This is expected because the latter two are relatively loose upper bounds for a truncated normal distribution whereas the distributions $N(\mu_j, \sigma_j)$ are close approximations to the truncated Gaussian with parameters μ_j' and σ_j'. Practically, the normal approximation is likely to be the easiest to calibrate in cases in which the theoretical guarantees of the other two approaches are not needed because it would be nearly exact for normally distributed usages.

We repeat the same tests for smaller machines with 32 CPU cores in Figure 5 (see Online Appendix I). The smaller machines are more difficult to overcommit because there is a smaller risk-pooling opportunity as can be seen by comparing the columns of Table 4. The three variations of our approach still yield similar and significant savings but now they substantially outperform the linear benchmarks: the cost reduction is at least double with all but the largest values of α. We highlight that, with the "better behaved" truncated Gaussian usage, we still obtain a 5% cost savings at a 0.01% risk whereas the linear benchmarks barely improve the no-overcommitment case.

As mentioned in Section 2.2, the value of α should be calibrated so as to yield an acceptable risk level given the data center, the workload, and the resource in question. Any data center has a baseline risk resulting from machine (or power) failure, and a temporary CPU shortage is usually much less severe relative to such a failure. On the other hand, causing a VM to crash because of a memory shortage can be as bad as a machine failure from the customer's point of view. Ultimately, the risk tolerance will be driven by technological factors, such as the ability to migrate VMs or swap memory while maintaining an acceptable performance.

7.4. Impact

We conclude that our approach allows a substantial cost reduction for realistic workloads. More precisely, we draw the following four conclusions.

- **Easy to implement:** Our approach is nearly as simple to implement as classical bin-packing heuristics. In addition, it works online and in real time and can be easily incorporated into existing scheduling algorithms.

- **Robustness:** The three variations we proposed yield very similar results. This suggests that our approach is robust to the type of approximation. In particular, the uncertain term b_j and the risk coefficient $D(\alpha)$ do not have a strong impact on the results. It also suggests that the method is robust to estimation errors in the measures of variability that define b_j.

- **Significant cost reduction:** With modern 72-core machines, our approach allows an 8%–14% cost savings relative to the no-overcommitment policy. This is achieved by considering a manageable risk level of 1%, which is comparable to other sources of risk that are not controllable (e.g., physical failures and regular maintenance operations).

- **Outperforming the benchmarks:** Our proposals show a consistent improvement over three different linear benchmarks that reduce to directly apply the classical best-fit heuristic. The difference is more substantial in cases with smaller machines, which is intuitively more challenging.

8. Conclusion

In this paper, we formulated and developed a practical solution for bin packing with overcommitment. In particular, we focused on a cloud computing provider who is willing to overcommit when allocating capacity to virtual machines in a data center. We modeled the problem as bin packing with chance constraints, where the objective is to minimize the number of purchased machines while satisfying the capacity constraints of each machine with a high probability. We first showed that this problem is closely related to an alternative formulation that we call the SMBP (submodular bin packing). Specifically, the two problems are equivalent under independent Gaussian job sizes or when the job size distribution belongs to the distributionally robust family with a given mean and (diagonal) covariance matrix. In addition, bin packing with chance constraints can be approximated by the SMBP for distributions with bounded support.

We first showed that for the bin-packing problem with general monotone submodular capacity constraints, it is impossible to find a solution within any reasonable factor from optimal. We then developed simple algorithms that achieve solutions within constant factors from optimal for the SMBP problem. We showed that any lazy algorithm is 8/3 competitive and that the first-fit heuristic is 9/4 competitive. Because the first-fit and

best-fit algorithms are easy to implement and well understood in practice, this provides an attractive option from an implementation perspective. Second, we proposed an algorithm for the off-line version of the problem and showed that it guarantees a two-approximation. Then, we used our model and algorithms to draw several insights on how to schedule jobs to machines and on the right way to overcommit. We convey that our method captures the risk-pooling effect as the "safety buffer" needed for each job decreases with the number of jobs already assigned to the same machine. Moreover, our approach translates to a transparent and meaningful recipe on how to assign jobs to machines by clustering similar jobs in terms of statistical information (i.e., jobs with similar b/μ should be assigned to the same machine).

Finally, we demonstrated the benefit of overcommitting and applied our approach to realistic workload data inspired by Google Compute Engine. We showed that our methods are (i) easy to implement, (ii) robust to the parameters, and (iii) significantly reduce the cost (1.5%–17% depending on the setting and the size of the physical machines in the data center).

Acknowledgments

Endnotes
[1] Although there is a job duration in cloud computing, it is generally very large (especially for transactional workload) and, hence, less constrained than the resource usage from the customer's perspective. The duration is also less important than the resource usage because most virtual machines tend to be long-lived, cannot be delayed or preempted, and are paid for by the minute.

[2] Insofar as many vector bin-packing heuristics are actually straightforward generalizations of the first-fit and best-fit rules, it becomes obvious how one can adapt our algorithms to the multiresource setting (see Section 5).

[3] A *submodular function* is a set function for which the difference in the incremental value made by adding a single element to an input set decreases as the size of the input set increases. Formally, a function $f : 2^\Omega \to \mathbb{R}$ is submodular if for every $X, Y \subseteq \Omega$, with $X \subseteq Y$ and every $x \in \Omega \backslash Y$, we have $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$.

[4] Radial distributions include all probability densities whose level sets are ellipsoids. The formal mathematical definition can be found in Calafiore and El Ghaoui (2006).

[5] This algorithm is linear in the number of items. The exact complexity is $C_\epsilon + C_n log(1/\epsilon)$, where C_ϵ depends only on ϵ and C is an absolute constant.

[6] The reason we need six jobs is technical and will be used in the proof of Theorem 6. In particular, one of the update operations does not hold if this number is smaller than six as we explain.

[7] The average distribution of workloads we present in Table 1 assumes small percentages of workloads with 32 and 16 cores and larger percentages of smaller VMs. The workload distributions we are using are representative for some segments of GCE. Unfortunately, we cannot unveil the real data because of confidentiality.

[8] Note that we clip the value of the constraint at the effective upper bound $(\sum_j x_{ij} \overline{A}_j)$ to ensure that no trivially feasible assignments are excluded. Otherwise, the Hoeffding's inequality-based constraint may perform slightly worse relative to the policy without overcommitment if it leaves too much free space on the machines.

[9] The exact time needed to collect a comparable data set from a production system depends on the data center size and on the sampling rate, which is a function of how quickly jobs enter and leave the system and of how volatile their usages are. By sampling independently in our simulations, we are assuming that the measurements from each machine are collected relatively infrequently (to limit correlation between successive measurements) and that the workloads are diverse (to limit correlation between measurements from different machines). This assumption is increasingly realistic as the size of the data center and the length of time covered increase.

References

Abdelaziz FB, Aouni B, El Fayedh R (2007) Multi-objective stochastic programming for portfolio selection. *Eur. J. Oper. Res.* 177(3): 1811–1823.

Albers S, Souza A (2011) Combinatorial algorithms lecture notes: Bin packing. Accessed August 1, 2017, https://www2.informatik.hu -berlin.de/alcox/lehre/lvws1011/coalg/bin_packing.pdf.

Atamtürk A, Narayanan V (2008) Polymatroids and mean-risk minimization in discrete optimization. *Oper. Res. Lett.* 36(5):618–622.

Bays C (1977) A comparison of next-fit, first-fit, and best-fit. *Comm. ACM* 20(3):191–192.

Bertsimas D, Popescu I (2005) Optimal inequalities in probability theory: A convex optimization approach. *SIAM J. Optim.* 15(3): 780–804.

Calafiore GC, El Ghaoui L (2006) On distributionally robust chance-constrained linear programs. *J. Optim. Theory Appl.* 130(1):1–22.

Cardoen B, Demeulemeester E, Beliën J (2010) Operating room planning and scheduling: A literature review. *Eur. J. Oper. Res.* 201(3):921–932.

Charnes A, Cooper WW (1963) Deterministic equivalents for optimizing and satisfying under chance constraints. *Oper. Res.* 11(1):18–39.

Coffman EG Jr, Garey MR, Johnson DS (1996) Approximation algorithms for bin packing: A survey. *Approximation Algorithms for NP-Hard Problems* (PWS Publishing Co., Boston), 46–93.

Coffman EG Jr, So K, Hofri M, Yao A (1980) A stochastic model of bin-packing. *Inform. Control* 44(2):105–115.

Csirik J, Johnson DS, Kenyon C, Orlin JB, Shor PW, Weber RR (2006) On the sum-of-squares algorithm for bin packing. *J. ACM* 53(1):1–65.

Delage E, Ye Y (2010) Distributionally robust optimization under moment uncertainty with application to data-driven problems. *Oper. Res.* 58(3):595–612.

de La Vega WF, Lueker GS (1981) Bin packing can be solved within $1+ \epsilon$ in linear time. *Combinatorica* 1(4):349–355.

Delorme M, Iori M, Martello S (2016) Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Eur. J. Oper. Res.* 255(1):1–20.

Deng Y, Shen S, Denton B (2016) Chance-constrained surgery planning under conditions of limited and ambiguous data. Working paper, University of Michigan, Ann Arbor.

Denton BT, Miller AJ, Balasubramanian HJ, Huschka TR (2010) Optimal allocation of surgery blocks to operating rooms under uncertainty. *Oper. Res.* 58(4, part 1):802–816.

Dinh HT, Lee C, Niyato D, Wang P (2013) A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Comm. Mobile Comput.* 13(18):1587–1611.

Dósa G (2007) The tight bound of first fit decreasing bin-packing algorithm is *ffd* ≤ 11/9*opt* + 6/9. *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies* (Springer, Berlin), 1–11.

Dósa G, Sgall J (2013) First fit bin packing: A tight analysis. Portier N, Wilke T, eds. *LIPIcs-Leibniz Internat. Proc. Informatics* (Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl Publishing, Germany), 538–549.

El Ghaoui L, Oks M, Oustry F (2003) Worst-case value-at-risk and robust portfolio optimization: A conic programming approach. *Oper. Res.* 51(4):543–556.

Fox A, Griffith R, Joseph A, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I (2009) Above the clouds: A Berkeley view of cloud computing. Report No. UCB/EECS 28(13):2009, Department Electrical Engineering and Computer Sciences, University of California, Berkeley.

Gilmore PC, Gomory RE (1961) A linear programming approach to the cutting-stock problem. *Oper. Res.* 9(6):849–859.

Goemans MX, Harvey NJ, Iwata S, Mirrokni V (2009) Approximating submodular functions everywhere. Mathieu C, ed. *Proc. 20th Ann. ACM-SIAM Sympos. on Discrete Algorithms* (Brown University, Providence, RI), 535–544.

Goyal V, Ravi R (2010) A PTAS for the chance-constrained knapsack problem with random item sizes. *Oper. Res. Lett.* 38(3):161–164.

Gupta V, Radovanovic A (2015) Lagrangian-based online stochastic bin packing. *ACM SIGMETRICS Perform. Eval. Rev.* 43(1):467–468.

Han J, Lee K, Lee C, Choi KS, Park S (2016) Robust optimization approach for a chance-constrained binary knapsack problem. *Math. Programming* 157(1):277–296.

Johnson DS (1974) Fast algorithms for bin packing. *J. Comput. System Sci.* 8(3):272–314.

Keller G, Tighe M, Lutfiyya H, Bauer M (2012) An analysis of first fit heuristics for the virtual machine relocation problem. Lobo J, Owezarski P, Zhang H, Medhi D, eds. *Proc. 8th Internat. Conf. Network Service Management* (IEEE, New York), 406–413.

Kenyon C, et al. (1996) Best-fit bin-packing with random order. *Proc. 7th Ann. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, (Society for Industrial and Applied Mathematics, Philadelphia), 359–364.

Kim U (2015) This one chart shows the vicious price war going on in Cloud computing. *Bus. Insider* (January 14), http://www.businessinsider.com/cloud-computing-price-war-in-one-chart-2015-1.

Kleinberg J, Rabani Y, Tardos É (2000) Allocating bandwidth for bursty connections. *SIAM J. Comput.* 30(1):191–217.

Lee S, Panigrahy R, Prabhakaran V, Ramasubramanian V, Talwar K, Uyeda L, Wieder U (2011) Validating heuristics for virtual machines consolidation. Technical Report, Microsoft Research Silicon Valley, Mountain View, CA.

Luedtke J, Ahmed S, Nemhauser GL (2010) An integer programming approach for linear programs with probabilistic constraints. *Math. Programming* 122(2):247–272.

Lueker GS (1983) Bin packing with items uniformly distributed over intervals [a, b]. *Proc. 24th Ann. Sympos. Foundations Computer Sci.* (IEEE, New York), 289–297.

Nemirovski A, Shapiro A (2006) Convex approximations of chance constrained programs. *SIAM J Optim.* 17(4):969–996.

Pisinger D, Sigurd M (2005) The two-dimensional bin packing problem with variable bin sizes and costs. *Discrete Optim.* 2(2):154–167.

Roytman A, Kansal A, Govindan S, Liu J, Nath S (2013) Algorithm design for performance aware vm consolidation. Technical Report, Microsoft Research, Redmond, WA.

Shylo OV, Prokopyev OA, Schaefer AJ (2012) Stochastic operating room scheduling for high-volume specialties under block booking. *INFORMS J. Comput.* 25(4):682–692.

Stolyar AL, Zhong Y (2015) Asymptotic optimality of a greedy randomized algorithm in a large-scale service system with general packing constraints. *Queueing Systems* 79(2):117–143.

Svitkina Z, Fleischer L (2011) Submodular approximation: Sampling-based algorithms and lower bounds. *SIAM J Comput.* 40(6):1715–1737.

Verma A, Pedrosa L, Korupolu MR, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at Google with Borg. *Proc. European Conf. Computer Systems* (EuroSys, Bordeaux, France), 1–17.

Zhang Y, Jiang R, Shen S (2016) Distributionally robust chance-constrained bin packing. Working paper, University of Michigan, Ann Arbor.