On the cluster admission problem for cloud computing

Ludwig Dierks
University Zurich
dierks@ifi.uzh.ch

Ian Kash
University of Illinois at Chicago
iankash@uic.edu

Sven Seuken
University Zurich
seuken@ifi.uzh.ch

ABSTRACT

Cloud computing providers must handle heterogeneous customer workloads for resources such as (virtual) CPU or GPU cores. This is particularly challenging if customers, who are already running a job on a cluster, scale their resource usage up and down over time. The provider therefore has to continuously decide whether she can add additional workloads to a given cluster or if doing so would impact existing workloads' ability to scale. Currently, this is often done using simple threshold policies to reserve large parts of each cluster, which leads to low average utilization of the cluster. In this paper, we propose more sophisticated policies for controlling admission to a cluster and demonstrate that they significantly increase cluster utilization. We first introduce the cluster admission problem and formalize it as a constrained Partially Observable Markov Decision Process (POMDP). As it is infeasible to solve the POMDP optimally, we then systematically design heuristic admission policies that estimate moments of each workload's distribution of future resource usage. Via simulations we show that our admission policies lead to a substantial improvement over the simple threshold policy. We then evaluate how much further this can be improved with learned or elicited prior information and how to incentivize users to provide this information.

CCS CONCEPTS

• **Computing methodologies** → *Partially-observable Markov decision processes*; *Planning under uncertainty*; • **Computer systems organization** → *Cloud computing*.

ACM Reference Format:
Ludwig Dierks, Ian Kash, and Sven Seuken. 2019. On the cluster admission problem for cloud computing. In *The 14th Workshop on the Economics of Networks, Systems and Computation (NetEcon'19), June 28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3338506.3340274

1 INTRODUCTION

Cloud computing is a fast expanding market with high competition where small efficiency gains translate to multi-billion dollar profits.[1] Nonetheless, most cloud clusters currently run at low average utilization (i.e., only a relatively low fraction of resources is actually used by customers). Some of this is caused by purely technical limitations, such as the need to reserve capacity for node failures or maintenance, outside factors such as fluctuations in overall demand, or inefficiencies in scheduling procedures, especially if virtual machines (VMs) might change size or do not use all of their requested capacity [28]. Another cause is the nature of many modern workloads: highly connected tasks running on different VMs that should be run on one cluster to minimize latency and bandwidth use [5]. In practice, different VMs from one user are thus bundled together into a *deployment* of interdependent workload.

In this paper, we pay special attention to the fact that, when the workload of a deployment changes, it can request a *scale out* in the form of additional VMs or shut some of its active VMs down. To get a sense of the difficulty of this problem, consider that, over time, the number of VMs needed by a specific deployment could vary by a factor 10 or even 100, and a request to scale out should almost always be accepted on the same cluster, as denying it would impair the quality of the service, possibly alienating customers. This means that providers face the difficult problem of deciding to which cluster to assign a deployment, as a deployment which is small today may, without warning, increase dramatically in size. Providers consequently hold large parts of any cluster as idle reserves to guarantee that only a very low percentage of these requests is ever denied.

1.1 Cluster Admission Control

We reduce the problem of determining to which cluster to assign a new deployment to the problem of determining, for a particular cluster, whether it is safe to *admit a deployment*, or if doing so would risk running out of capacity if the deployment scales [5]. While a lot of research has been done on scheduling *inside* the cluster [22, 27, 29], the admission decision has not been well studied before. Consequently, cloud providers are still often using simple policies like thresholds that only depend on the current utilization of a cluster.[2] These policies may seem reasonable, because the law of large numbers suggests that the overall utilization carries most information for large clusters. But as Cortez et al. [5] have shown, a relatively small number of deployments account for most of the utilization. This suggests that specific deployments have a larger impact on the failure probability than may be apparent at first sight.

We formalize the cluster admission problem as a constrained Partially Observable Markov Decision Process (POMDP) [23] where each deployment behaves according to some stochastic process and the controlling agent (the cloud provider) tries to maximize the number of active compute cores without exceeding the cluster's capacity. Since the exact stochastic processes of arriving deployments are not known to the agent, it has to reason about the observed

[1] https://www.microsoft.com/en-us/Investor/earnings/FY-2018-Q2/press-release-webcast

[2] This is common knowledge in the industry and was additionally confirmed to us in communications with various domain experts. However, to the best of our knowledge, there exists no publicly available written source.

behavior. The large scale of the problem as well as the highly complicated underlying stochastic processes make finding optimal policies infeasible, even for the underlying (fully observable) MDP and with limited look-ahead horizon.

1.2 Overview of Contributions

We present tailored heuristics to solve the cluster admission control problem. For this, we first present a succinct mathematical formulation of the cluster admission problem as faced by cloud providers before defining the optimal policy as the solution to a constrained POMDP. Since optimally solving this POMDP is not feasible, we next propose a strategy for constructing heuristic policies via a series of simplifying assumptions. These assumptions reduce the highly branching look-ahead space down to the approximation of a random variable using its moments. We then present the currently used threshold policy that does not take probabilistic information into account as well as two new policies that take successively higher moments into account. We fit our model to data from a real-world cloud computing center (Microsoft Azure internal jobs [5]). Via simulations, we show that our higher moment policies produce a 14% improvement over current practice, which can translate to hundreds of millions of dollars of savings for large cloud providers. Then we examine how the utilization of the cluster can further be improved if more precise prior information about arriving deployments is available. We give a simple framework which captures a notion of the quality of information available, and through additional simulations quantify how the policies benefit from this additional information. Finally, we present a new *information elicitation approach*, introducing a variance-based pricing rule to elicit labels from users. This rule provides users with the right incentives to (a) label their deployments properly (into high and low variance deployments) and (b) structure their workloads in a way that helps the cluster run more efficiently.

1.3 Related work

While studying which deployments to admit to a cluster, we abstract away the question of exactly which resources they should use. Prior work on *cluster scheduling and load balancing* has already studied these questions [22, 27]. Other work addresses a different notion of admission control, namely how to manage queues for workloads which will ultimately be deployed to that cluster [6]. Another line of work on scheduling looks at how multidimensional resources can be fairly divided among deployments [10, 11, 14, 18]. There is also a literature that views scheduling through the lens of stochastic online bin packing [4, 25], dealing with issues of changing workloads on overcommitted resources, but at smaller scales.

In this paper, we examine the value of learning from prior deployments. Other work has explored similar opportunities in the context of resource planning and scheduling in analytics clusters [12]. There is a large literature on market design challenges in the context of the cloud [13]. Existing work has studied both, queueing models where decisions are made online with no consideration of the future [1, 8], and reservation models that assume very strong information about the future [2]. Our work sits in an interesting intermediate position where users may have rough information about the types of their deployments.

Solving POMDPs is a well-studied problem [20, 24]. Unfortunately, finding an optimal policy is known to be PSPACE-complete even for finite-horizon problems [17]. Even finding ϵ-optimal policies is NP-hard for any fixed ϵ [16]. In our case, the problem is further exacerbated by the existence of side constraints. Constrained POMDPS are far less well studied than unconstrained POMDPS. General (approximation) strategies proposed in the past include linear programming [19], point-based value iteration [15], a mix of online-look ahead and offline risk evaluation [26], and forward search with pruning [21]. None of these approaches is efficiently applicable when the state space of the underlying MDP is large or, as in our case, partly continuous. While some work has addressed continuous state space POMDPs [3, 9], none of this work is directly applicable to a constrained problem of the size we are considering.

2 PRELIMINARIES

2.1 The cluster admission problem

We consider a single cluster in a cloud computing center. A cluster consists of c *cores* that are available to perform work. c is also called the cluster's *capacity*. These cores are used by *deployments*, i.e. interdependent workloads that use one or more cores. The set of deployments currently on the cluster is denoted by X, and each deployment $x \in X$ is assigned a number of cores C^x. Any core that is assigned to a deployment is called *active*, while the remainder are *inactive*. All inactive cores are assumed to be ready to be assigned and become active at any time. The exact placement of cores inside the cluster is not taken into account at this level and in consequence we do not model the grouping of cores into VMs.

A deployment can request to *scale out*, i.e., increase its number of active cores. Each such request is for one or more additional cores and must be accepted whenever enough inactive cores are available. Following current practice, scale out requests must be granted entirely or not at all. Deployments may also stop using some of their cores over time, turning these cores inactive. A deployment *dies* when it reaches zero active cores. It can also die spontaneously, instantly turning all its cores inactive. Intuitively this models a user's decision to shut down the deployment. We assume that the processes for the time between scale outs and the time until a core becomes inactive are memoryless. This is common when modelling arrival and departure processes, and has been used in previous models of cloud computing [1, 8].

New deployment requests arrive over time and are accepted or rejected according to an *admission policy* based on the current state of the cluster and the arriving deployment. The policy has to ensure that the cluster is not forced to reject a higher percentage of scale out requests than is specified by an internal *service level agreement* (SLA) τ. If a scale out request cannot be accepted because the cluster is already at capacity, one failure for the purpose of meeting the SLA is logged. An optimal policy therefore maximizes the *utilization* of the cluster, i.e., the average number of active cores, while making sure the SLA is observed in expectation (i.e., over time).

2.2 POMDP Formulation

If the parameters governing the scale out behavior of deployments were known, the optimal policy could be formulated as the solution to a Markov Decision Process. But as they are generally not known, the cluster can only reason about them based on observed behavior and (optionally) some a priori information about individual deployments. The problem of finding an optimal policy can therefore be formalized as a constrained discrete time Partially Observable Markov Decision Process (POMDP) $(\mathbb{S}, \mathbb{A}, \mathbb{R}, \mathbb{T}, \mathbb{C}, \tau, \Omega, \mathbb{O})$. While deployments can arrive at arbitrary times, it takes time to make the acceptance decision. Thus, there is little loss in assuming (for the purposes of our POMDP) that time is discrete. $\mathbb{S}$ denotes the space of all possible states the cluster can be in. A state $s \in \mathbb{S}$ is assumed to contain all information about the cluster's active deployments $X(s)$ (including both their current size and scaling process parameters) as well as the deployments that arrived during the current time step. The action set $\mathbb{A}$ consists of accepting or rejecting the newly arrived deployments. The reward function $\mathbb{R}(s) = \sum_{x \in X(s)} C^x$ is simply the number of active cores in a state s. $\mathbb{T}(s'|s, a) \forall s's \in \mathbb{S}, \forall a \in \mathbb{A}$ denotes the state transition probabilities and $\mathbb{C}(s, a)$ denotes the expected number of failures that occur at the state transition from a given state-action pair. τ is the SLA, as described above. Ω is the set of possible observations and $\mathbb{O}$ an observation model.

The provider does not actually observe the full state space. Instead, an observation $o \in \Omega$ only reveals whether deployments arrived to the cluster or died and the current size C^x of any active deployment $x \in X$, while in particular not revealing the parameters of the deployments. Note that this means that our observation model $\mathbb{O}$ is deterministic, but that many states share the same observation. As is standard, we further denote the clusters' current knowledge about which state s it is in via a belief state $\mathbb{B}$, i.e. a probability distribution over all possible states. Whenever the state of the system changes in some time step n and the cluster obtains a new observation o, the belief state is updated according to the observation model, i.e.

$$b_{n+1}(s'|b_n, a, o) \propto \mathbb{O}(o|s') \sum_s \mathbb{T}(s'|s, a) b_n(s). \tag{1}$$

We can now formulate the problem of finding an optimal policy.

Problem 2.1. *An optimal policy p^* starting in belief state b can be found by solving*

$$p^* = argmax_p \sum_n \int_s f_{n,p,b}(s) \sum_{x \in X(s)} C^x ds \tag{2}$$

$$s.t. \int_s f_{n,p,b}(s) \mathbb{C}(s, a_p) ds \leq \tau \; \forall n > 0 \tag{3}$$

where $f_{n,p,b}$ is the probability density function of the distribution over the states of the system for time step n given policy p and starting belief b. An optimal policy therefore maximizes the expected reward, i.e. the expected number of active cores over all future time steps. The side constraint (3) guarantees that the optimal policy is only chosen from those policies that observe the SLA, given the provider's belief.

Before moving on to try to find policies that solve the POMDP, we pause to examine some key features of our model. Note that if the underlying state were fully observable, with c cores there could be as many as c deployments each of which requires at least four parameters to describe its size and processes. The parameter space for each stochastic process is continuous, meaning a state $s \in \mathbb{S}$ consists of more than c discrete and $3c$ continuous parameters. Thus, even if we would fully discretize the parameter space, the number of states of our underlying MDP would be exponential in c. This makes solving it a challenge even without the added complexity of the SLA constraint and the partial observability. This means that approaches to solving our POMDP which would require solving versions of the MDP as a subroutine are infeasible in our setting, where clusters contain $10,000$ or more cores.

3 ADMISSION CONTROL POLICIES

In this section, we present the simple cluster admission policy currently used in practice, as well as our new, more sophisticated policies. Even with limited look-ahead horizons, optimal policies cannot feasibly be calculated for the POMDP. There is no simple closed form for the state transition probabilities and even if there was, the branching factor of the POMDP is too large. This problem persists even if we would only try to solve the underlying MDP. We therefore present heuristic policies tailored to the cluster admission problem. Our policies are based on a number of carefully chosen simplifying assumptions:

Disregard future arrivals. Our policy does *not* reject deployments simply because there is a chance that better behaved deployments arrive in the future. The optimal policy, under this assumption, accepts a new deployment whenever doing so does not violate the side constraint. This allows our policy to decide acceptance based only on the failure probability caused by the current population of the cluster and assuming no more deployments arrive in the future. In the cloud domain, this assumption is actually desirable, as even customers with high demand variability must be served by some cluster in the data center.

Relax capacity constraint. When disregarding future arrivals, the cluster's future state only depends on how the sizes of the current active deployments change. Despite this, the probabilities are complex because if one deployment helps fill the cluster then further scale out requests by other deployments are denied, introducing correlations between the size of deployments. Instead, the policy assumes the cluster can run an infinite number of cores which allows the evolution of each deployment to be independent. Intuitively this is reasonable because the cluster being full should be rare if the constraint on the policy is being met. Note that we only make this assumption for the prediction of state transitions; the policy still logs scale outs that would fail in reality as failures.

With these first two assumptions, we can now describe the future evolution of the cluster using independent random variables L_n^x to denote the size of every deployment $x \in X$ at time step n.

Assume at most one failure occurs per time step. Since the probability that more than one scale out request has to be denied in a single time step approaches zero with increased granularity of the time discretization, it is reasonable for the policy to assume that at most one failure to scale out is counted per time step. Adding this assumption now allows us to simplify the side constraint:

Proposition 1. *Under the three assumptions (disregard future arrivals, relax capacity constraint, and assume at most one failure* *occurs per time step), the side constraint (3) for any time step n becomes*

$$\int_s f_{n,p,b}(s)\mathbb{C}(s, a_p)ds = Pr(\sum_x L^x_{n+1} > c) \leq \tau. \tag{4}$$

An optimal policy that operates under these three assumptions accepts an arriving deployment whenever doing so does not violate inequality (4).

The statement follows directly from the three assumptions.

Approximate failure probability using (approximate) moments. While the last three assumptions greatly simplified the problem, the processes at hand leave us without an analytical form of inequality (4). Instead, we utilize the (approximate) moments of L_n to construct simple policies. In the following, we present three such policies that make use of successively higher moments.

3.1 Zeroth Moment Policy (Baseline)

The baseline admission control policy that is widely used in practice is a myopic policy that simply compares the current number of active cores to a threshold. This policy does not require any information about the set of deployments besides the total number of active cores. We also call it a *Zeroth Moment Policy* because it takes no information about the future into account. The limited amount of information it uses means that it has to be very conservative in how many deployments to accept, since it does not know how often or fast they will scale out.

Definition 1 (Zeroth Moment policy). *Under a zeroth moment policy with threshold t, a newly arriving deployment is accepted if, after accepting the deployment, there would be less than t cores active.*

3.2 First Moment Policy

Frst moment policies approximate (4) by utilizing the first moments, i.e. the means, of the deployment processes. In the spirit of Markov's inequality, we propose to accept arriving deployments whenever the expected utilization lies below a chosen threshold.

Definition 2 (First Moment Policy). *Under a first moment policy with threshold t, a newly arriving deployment is accepted if, after accepting the deployment, the expected number of active cores would be less than t in all future time steps, i.e.*

$$\sum_{x \in X} E[L^x_n] \leq t \quad \forall n \tag{5}$$

Note that, unless the exact parameters of a deployment are known, exactly calculating the expectation is not feasible. Thus, we use an approximation that is shown in the paper's full version [7].

3.3 Second Moment Policy

First moment policies still fail to take into account much of the structure of deployments. In a sense they always have to take the worst possible population mix into account and run the risk of accepting deployments with low expected size but high variance when close to the threshold. One way around this is to also take the second moment, i.e. the variance of L_n, into account. To do so, we propose to use Cantelli's inequality, a single-tailed generalization of Chebyshev's inequality, to approximate inequality (4).

Table 1: Simulation results

Policy	Threshold	Utilization	Standard Error
Zeroth Moment	$t = 10,644$	59.2%	0.54%
First Moment	$t = 14,262$	67.3%	0.58%
Second Moment	$\rho = 0.1063$	67.5%	0.7%

Cantelli's Inequality states that, for a real-valued random variable L and $\epsilon \geq, 0$ it holds that

$$Pr(L - E[L] \geq \epsilon) \quad \leq \quad \frac{Var[L]}{Var[L] + \epsilon^2}. \tag{6}$$

If we now set $\epsilon = (c - \sum_{x \in X} E[L^x_n])$, we can bound the probability of running over capacity. While the bound given by the inequality is not tight enough to simply set it to τ, it can be used to bound the first two moments in a systematic way.

Definition 3 (Second Moment Policy). *Under a second moment policy with threshold ρ, a newly arriving deployment is accepted if, after accepting the deployment, the estimated probability of running over capacity would be less than ρ in all further time steps, i.e.*

$$\frac{\sum_{x \in X} Var[L^x_n]}{\sum_{x \in X} Var[L^x_n] + (c - \sum_{x \in X} E[L^x_n])^2} \leq \rho \quad \forall n \tag{7}$$

The formulas we use to approximate the variance are given in the full version of the paper [7].

Note that the computational overhead of the second moment policy with properly chosen prior distributions is in $O(Lcn)$, where L is the arrival rate of new deployments per hour, c is the size of the cluster and n is the number of evaluated time steps of the look-ahead. More details can be found in the full paper [7].

4 EMPIRICAL EVALUATION

In this section, we evaluate the performance of our admission policies using a model fitted to the real-world data trace of Cortez et al. [5]. [3] A discussion of our fitting procedure and the fitted model can be found in the full version of the paper [7].

We simulate clusters with capacity $c = 20,000$ for a 3-year period with all three policies. We determine the optimal threshold for each policy via binary search, subject to meeting an SLA of 0.01%. An average of 1 new deployment arrives per hour according to a Poisson process. The parameters of each arriving deployment are drawn from the fitted distributions (see the full version of the paper [7]). To simulate three years with a reasonable number of core-hours, the look ahead for the first and second moment policies is divided into 5 parts: a 24-hour look ahead, a week long look ahead, a month long look ahead and both a 1 and 3 year long look ahead. Each look ahead discretizes its time into 600 time steps and evaluates the policy at each time step until the arriving deployment becomes marginal (i.e. it's evaluated moments are less than $1e^{-5}$).

[3] Since the data set is limited, we cannot directly evaluate the policy on the historical deployments. We defer such evaluations to future work once more data is available.

The results are summarized in Table 1. The zeroth moment policy obtains its best result with a threshold of $t = 10,644$. It results in an average utilization of 59.2% over the cluster's lifetime. The first moment policy with $t = 14,262$ increases the utilization by 8.1 absolute percentage points to 67.3%. This constitutes a relative increase of 13.6% over the zeroth moment policy and means that the same number of deployments can be served by roughly 12% less hardware, an invaluable competitive edge. Similarly, the second moment policy with threshold $\rho = 0.1063$ achieves a utilization of 67.5%, an relative improvement of 14%. With these thresholds, the overwhelming number of simulated clusters has a rejection rate of 0, i.e., during the lifetime of a cluster, no scale out is rejected. Conversely, in a few runs, too many very large, long lived deployments are accepted in the beginning of a cluster's lifetime, leading to mass rejections months or even years later. As this happens early in a cluster's lifetime when not much is known about deployments, the difference between first and second moment policies is relatively small.

5 EXTENSION 1: LEARNED PRIOR

So far, our observation model assumed that the cluster does not have any information about arriving deployments, except for their initial number of cores. The acceptance decision must therefore primarily depend on the state of the deployments that are already in the cluster. Intuitively, policies could better control whether accepting a deployment risks violating the SLA if they had more information about its future behavior. One way to obtain such information would be to use machine learning (ML) based on features of the arriving deployment and the submitting user's history [5]. While evaluating particular ML algorithms is beyond the scope of this paper, we evaluate the effect different levels of available information have. To parameterize the level of knowledge, we assume that the policy gets passed some number of samples from each true scaling process distribution of each arriving deployment.[4]

We simulated the first and second moment policies with 4 different levels of information (0, 1, 5, and 50 observations), with the same set-up as in Section 4. The results are shown in Figure 1. We see that having prior knowledge equivalent to even a single sample would improve utilization significantly, resulting in a utilization of 77.78% and 79.8% for the first and second moment policies, respectively. Here it becomes visible how the more complex model of the second moment policy is able to better utilize the available information. While the first moment policy struggles to obtain further improvements with better priors, the second moment policy can achieve a utilization of up to 83.77%. While it is infeasible to calculate the utilization an optimal solution to the POMDP would achieve, an upper bound of 92.1% is given by analyzing policies that do not have to satisfy any SLA. The second moment policy with good prior information is only 9% below this (unreachable) upper bound, but 24.1% above the same policy without prior information and 41.5% above the baseline policy. This shows both the power of our policies and the great importance of taking all available prior information about arriving deployments into account.

[4] As we have used conjugate prior distributions in our model, this approach matches the standard interpretation of parameters of the posterior distribution in terms of pseudo observations.

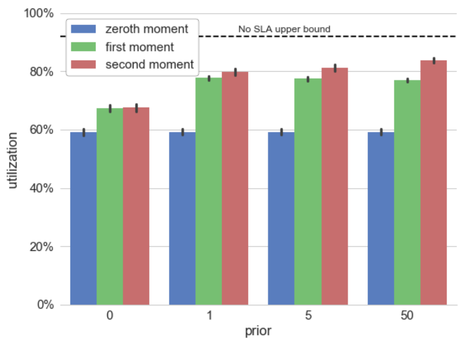

Figure 1: Performance of different policies depending on prior information (error bars indicate standard errors)

6 EXTENSION 2: ELICITED PRIOR

In this section, we use techniques from mechanism design to improve the quality of prior information. While using ML to predict the parameters of deployments is powerful, users do not typically submit deployments with arbitrary parameters. Instead, they may have a small number of different *types* of deployments. While ML may be able to learn this, it is better to directly elicit some information from the users. However, asking for estimates of parameters for a given deployment is problematic, as it either shifts risk to the consumer or enables manipulation. This leads to the idea of asking users to categorize their deployments into different *types* of similar deployments. Learning priors for each individual type then results in more precise priors and higher efficiency.

Typically, users are charged a fixed payment per hour for each core their deployment uses. To this, we add a small additional charge based on the variance of the estimate for the deployment's scaling process and allow users to label the type of their deployments, resulting in an hourly variance-based payment rule of the form:

$$\pi(x) = \kappa_1 C^x + \kappa_2 Var(x), \tag{8}$$

where κ_1 and κ_2 are price constants and $Var(x)$ is an estimate of the variance of the deployment. A payment rule of this form incentivizes users to assign similar labels to similar deployments to minimize the estimated variance.

To see this, consider a user who has two types of deployment, x and y, with true variances $Var(x)$ and $Var(y)$. He could now simply submit the deployments under a single label. For the provider, this means that each submitted deployment is of either type with a certain probability, which increases the variance of his prediction. But if the user would label his deployments instead, the provider would know for each arriving deployment which type it is, reducing variance and therefore the need to reserve capacity. The following proposition, which is immediate from the law of total variance, shows that, at least in the long run, labeling his deployments also reduces a user's payments.

Proposition 2. *Let z be the mixture that results from submitting one of two types of deployments x, y chosen by a Bernoulli random variable $\alpha \sim Bernoulli(p_\alpha)$, i.e., such that z is of type x with probability p_α and of type y with probability $1 - p_\alpha$. Then it holds that*

$$p_\alpha Var(x) + (1 - p_\alpha) Var(y) \le Var(z) \tag{9}$$

The proof follows from the law of total variance and can be found in the full version of the paper [7]. Proposition 2 shows that the user would be better off, on average, by splitting the mixture and submitting the deployments under separate labels. Note that this abstracts away issues of learning and non-stationary strategic behavior; but for reasonable learning procedures we expect a consistent labeling to lead to lower variance than a mixture while learning. Further, this approach not only gives the user correct incentives to reveal the desired information, but actually incentivizes him to improve the performance of the system. In particular, another way he can lower his payment under this scheme (outside the scope of our model) is to design his deployments in such a way that they have lower variance in their resource use. Since more predictable deployments would allow the policy to maintain a smaller buffer, this provides an additional benefit to the system's efficiency.

How much any given user could ultimately save by labeling his deployments mostly depends on how different his deployment types are and on how high the provider sets the charge for variance. A user whose deployments are quite uniform will not save much, while a user with some deployments which never scale and some that scale a lot can potentially save a lot. Note that how much the provider should charge is not immediately clear. While he would want to set a high price to put a strong incentive on users, he also has to keep the competition from other providers in mind. At what point the loss of market share outweighs the gain in efficiency is an intriguing problem we leave for future work.

7 CONCLUSION

We have studied the problem of cluster admission control for cloud computing. The optimal policy would be given as the solution to a very large constrained POMDP which is infeasible to solve. In practice, simple threshold policies are therefore used for this problem, while we propose carefully designed heuristic policies. Our simulations, with parameters fit to traces from Microsoft Azure, show the potential gains based on our policies. Our results demonstrate that utilization can be increased by up to 14% just from learning about deployments while they are active in the cluster. This can be increased to a 41.5% gain if better prior information about arriving deployments is available, for example through learning or elicitation techniques. At cloud scale, these savings translate to many hundreds of millions of dollars over the course of a hardware lifetime, and any dollar saved directly translates to a gross profit increase for the cluster provider. Our overall approach is fundamentally about managing the tail risks of a stochastic process. In our case, these are the rare events where the cluster runs out of capacity. Thus, our approach also applies in other domains where the management of tail risks is important, for example in finance.

REFERENCES

[1] Vineet Abhishek, Ian A. Kash, and Peter Key. 2012. Fixed and market pricing for cloud services. In *7th Workshop on the Economics of Networks, Systems, and Computation (NetEcon)*. 157–162.

[2] Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. 2015. Truthful Online Scheduling with Commitments. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*. ACM, 715–732.

[3] Alex Brooks, Alexei Makarenko, Stefan Williams, and Hugh Durrant-Whyte. 2006. Parametric POMDPs for planning in continuous state spaces. *Robotics and Autonomous Systems* 54, 11 (2006), 887–897.

[4] Maxime C Cohen, Philipp W Keller, Vahab Mirrokni, and Morteza Zadimoghaddam. 2019. Overcommitment in Cloud Services: Bin Packing with Chance Constraints. *Management Science* (2019).

[5] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP '17)*. ACM, New York, NY, USA, 153–167.

[6] Christina Delimitrou, Nick Bambos, and Christos Kozyrakis. 2013. QoS-Aware Admission Control in Heterogeneous Datacenters.. In *ICAC*. 291–296.

[7] Ludwig Dierks, Ian Kash, and Sven Seuken. 2019. On the cluster admission problem for cloud computing. *arXiv preprint arXiv:1804.07571* (2019). https://arxiv.org/abs/1804.07571

[8] Ludwig Dierks and Sven Seuken. 2019. Cloud pricing: the spot market strikes back. In *Proceedings of the 2019 ACM Conference on Economics and Computation*. ACM.

[9] Michael O'Gordon Duff and Andrew Barto. 2002. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. Ph.D. Dissertation. University of Massachusetts at Amherst.

[10] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *USENIX Symposium on Networked Systems Design and Implementation*.

[11] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.. In *NSDI*, Vol. 11. 22–22.

[12] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, et al. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters.. In *OSDI*. 117–134.

[13] Ian A Kash and Peter B Key. 2016. Pricing the cloud. *IEEE Internet Computing* 20, 1 (2016), 36–43.

[14] Ian A. Kash, Ariel D. Procaccia, and Nisarg Shah. 2014. No Agent Left Behind: Dynamic Fair Division of Multiple Resources. *Journal of Artificial Intelligence Research* 51 (2014), 579–603.

[15] Dongho Kim, Jaesong Lee, Kee-Eung Kim, and Pascal Poupart. 2011. Point-based value iteration for constrained POMDPs. In *IJCAI*. 1968–1974.

[16] Christopher Lusena, Judy Goldsmith, and Martin Mundhenk. 2001. Nonapproximability results for partially observable Markov decision processes. *Journal of artificial intelligence research* 14 (2001), 83–103.

[17] Christos H. Papadimitriou and John N. Tsitsiklis. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12, 3 (Aug. 1987), 441–450.

[18] D. C. Parkes, A. D. Procaccia, and N. Shah. 2012. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *ACM Conference on Electronic Commerce (EC)*.

[19] Pascal Poupart, Aarti Malhotra, Pei Pei, Kee-Eung Kim, Bongseok Goh, and Michael Bowling. 2015. Approximate Linear Programming for Constrained Partially Observable Markov Decision Processes.. In *AAAI*, Vol. 1. 3342–3348.

[20] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

[21] Pedro Santana, Sylvie Thiébaux, and Brian Williams. 2016. RAO*: an algorithm for chance constrained POMDPs. In *Proc. AAAI Conference on Artificial Intelligence*.

[22] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. 2013. Omega: flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 351–364.

[23] Richard D Smallwood and Edward J Sondik. 1973. The optimal control of partially observable Markov processes over a finite horizon. *Operations research* 21, 5 (1973), 1071–1088.

[24] Trey Smith and Reid Simmons. 2012. Point-based POMDP algorithms: Improved analysis and implementation. *arXiv preprint arXiv:1207.1412* (2012).

[25] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. 2014. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Trans. Comput.* 63, 11 (2014), 2647–2660.

[26] Aditya Undurti and Jonathan P How. 2010. An online algorithm for constrained POMDPs. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 3966–3973.

[27] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM.

[28] Ying Yan, Yanjie Gao, Yang Chen, Zhongxin Guo, Bole Chen, and Thomas Moscibroda. 2016. TR-Spark: Transient Computing for Big Data Analytics. In *SoCC*.

[29] J. Zhao, K. Yang, X. Wei, Y. Ding, L. Hu, and G. Xu. 2016. A Heuristic Clustering-Based Task Deployment Approach for Load Balancing Using Bayes Theorem in Cloud Environment. *IEEE Transactions on Parallel and Distributed Systems* 27, 2 (Feb 2016), 305–316.