A Theory of Auto-Scaling for Resource Reservation in Cloud Services

Konstantinos Psychas, Javad Ghaderi
Department of Electrical Engineering, Columbia University

ABSTRACT

We consider a distributed server system consisting of a large number of servers, each with limited capacity on multiple resources (CPU, memory, disk, etc.). Jobs with different rewards arrive over time and require certain amounts of resources for the duration of their service. When a job arrives, the system must decide whether to admit it or reject it, and if admitted, in which server to schedule the job. The objective is to maximize the expected total reward received by the system. This problem is motivated by control of cloud computing clusters, in which, jobs are requests for Virtual Machines or Containers that reserve resources for various services, and rewards represent service priority of requests or price paid per time unit of service by clients. We study this problem in an asymptotic regime where the number of servers and jobs' arrival rates scale by a factor L, as L becomes large. We propose a resource reservation policy that asymptotically achieves *at least* $1/2$, and under certain monotone property on jobs' rewards and resources, at least $1-1/e$ of the optimal expected reward. The policy automatically scales the number of VM slots for each job type as the demand changes, and decides in which servers the slots should be created in advance, without the knowledge of traffic rates. It effectively tracks a low-complexity greedy packing of existing jobs in the system while maintaining only a small number, $g(L) = \omega(\log L)$, of reserved VM slots for high priority jobs that pack well.

KEYWORDS

Scheduling, Loss Systems, Fluid Limits, Resource Allocation

1 INTRODUCTION

There has been a rapid migration of computing, storage, applications, and other services to cloud. By using cloud (e.g., Amazon AWS [1], Microsoft Azure [22], Google Cloud [9]), clients are no longer required to install and maintain their own infrastructure. Instead, clients use the cloud resources on demand, by procuring Virtual Machines (VMs) or Containers [2, 10] with specific configurations of CPU, memory, disk, and networking in the cloud data center, depending on their needs.

A key challenge for the cloud service providers is to efficiently support a wide range of services on their physical platform. They usually offer QoS guarantees (in SLAs) [3] for clients' applications and services, and allow the number of VM instances to scale up or down with demand to ensure QoS guarantees are met. Various predictive and reactive schemes have been proposed for dynamically allocating VMs to different services, e.g., [8, 12, 15, 21, 27, 29], however, they mostly assume a dedicated hosting model where VMs of each application run on a dedicated set of servers. Such models do not consider potential consolidation of VMs in servers which is known to significantly improve efficiency and scalability [6, 30].

This research was supported by NSF grants CNS-1717867 and CNS-1652115.

In this paper, we consider a cloud data center consisting of a large number of servers. As an abstraction in our model, a VM is simply a multi-dimensional object (vector of resource requirements) that should be served by one server and *cannot be fragmented*. Each server has a limited fixed capacity on its available resources (CPU, memory, disk, networking). VM requests belong to a collection of VM types, each with a specific resource requirement vector, and a specific reward that represents its service priority or the price that will be paid per time unit of service by the client. When a VM request arrives, we must decide in an online manner whether to accept it, and, if so, in which server to schedule it. The objective is to *maximize the expected total reward* received by the system. Note that finding the right packing for a given workload is a hard combinatorial problem (related to multi-dimensional Knapsack [17]).

In this paper, we study a stochastic version of the *Online Multiple Knapsack* problem in an asymptotic regime, where the number of servers L is large and requests for VMs of type j arrive at rate $\lambda_j L$, $j = 1, \cdots, J$, and each requires service with mean duration $1/\mu_j$. The (normalized) load of the system is defined as $\boldsymbol{\rho} := (\lambda_j/\mu_j, j = 1, \cdots, J)$. This is the *heavy-traffic* regime, e.g. [13, 14, 16, 18, 23, 33, 34], and it has been shown that algorithms with good performance in such a regime also show good performance in other regimes. The interesting scenario occurs when not all VM requests can be scheduled (e.g., $\boldsymbol{\rho} > \boldsymbol{\rho_c}$ for a *critical load* $\boldsymbol{\rho_c}$ on the boundary of system capacity), in which case a fraction of the traffic has to be rejected even by the optimal policy. We propose an adaptive reservation policy that makes admission and packing decisions without the knowledge of $\boldsymbol{\rho}$. Packing decisions include placement of admitted VM in one of the feasible servers, and migration of at most one VM across servers when a VM finishes its service.

Related Work. There is classical work on large loss networks, e.g. [4, 13, 14, 18], where calls with different bandwidth requirements and priorities arrive to a telecommunication network. Trunk reservation has been shown to be a robust and effective call admission policy in this setting, in which each call type is accepted if the residual link bandwidth is above a certain threshold for that type. The performance of trunk reservation policies has been analyzed in the asymptotic regime where the call arrival rates and link's capacity scale up by a factor N, as $N \to \infty$. This is different from our large-scale server model, where the server's capacity is "fixed" and only the number of servers scales (a.k.a. system scale-out as opposed to scale-up).

The works [19, 20, 24, 25] consider a queueing model where VM requests are placed in a queue and then served by the system. In this paper, we are considering a loss model without delay, i.e., each VM request upon arrival has to be served immediately, otherwise it is lost. The recent works [7, 32] study a system with an infinite number of servers and their objective is to minimize the number of occupied servers. The auto-scaling algorithm proposed in [11] also assumes such an infinite server model. These are different from our

setting where we consider a finite number of servers and study the total reward of served VMs by the system.

The works [16, 23, 31, 34] study the blocking probability in a large-scale server system where *all VMs have the same reward*. The work [31] assumes a subcritical system load and only shows local stability of fluid limits. The works [16, 23, 34] show that, under a power-of-d choices routing, the blocking probability drops much faster compared to the case of uniform random routing. However, there is no analysis of optimality, especially in a supercritical regime where even the optimal policy has a non-zero blocking probability. Moreover, such algorithms treat all VMs with the same priority (reward) when making decisions, thus a low priority VM can potentially block multiple high priority ones.

Contributions. We propose a dynamic resource reservation policy that makes admission and packing decisions based on the current system state, and prove that it asymptotically achieves *at least* $1/2$, and under certain monotone property on VMs' rewards and resources, at least $1-\frac{1}{e}$ of the optimal expected reward, as the number of servers $L \to \infty$ [1]. Further, simulations suggest that for real cloud VM instances, the achieved ratio is in fact very close to one.

2 MODEL AND DEFINITIONS

Cloud Model. We consider a collection of L servers denoted by the set $\mathcal{L}$. Each server $\ell \in \mathcal{L}$ has a limited capacity on different resource types (CPU, memory, disk, networking, etc.). We assume there are $n \geq 1$ types of resource.

VM Model. There is a collection of VM types denoted by the set $\mathcal{J}$. The VM types are indexed in arbitrary order from 1 to J. Each VM type j requires a vector of resources $\mathbf{R}_j = (R_j^1, \cdots, R_j^n)$, where R_j^d is its requirement for the d-th resource, $d = 1, \cdots, n$.

VMs are placed in servers and reserve the required resources. The sum of reserved resources by the VMs placed in a server should not exceed the server's capacity. A vector $\mathbf{k} = (k_1, \cdots, k_J) \in \mathbb{Z}_+^J$ is said to be a feasible configuration if the server can simultaneously accommodate k_1 VMs of type 1, k_2 VMs of type 2, $\cdots$, k_J VMs of type J. We use $\mathcal{K}$ to denote the set of all feasible configurations (including the empty configuration $\mathbf{0}_J$). The number of feasible configurations will be denoted by $C := |\mathcal{K}|$.

We define $\mathcal{K}^{\mathcal{J}'}$ to be the set of feasible configurations that include only VMs from a subset of types $\mathcal{J}' \subseteq \mathcal{J}$, i.e.,

$$\mathcal{K}^{\mathcal{J}'} = \{\mathbf{k} \in \mathcal{K} : k_j = 0, \forall j \notin \mathcal{J}'\}. \tag{1}$$

We do not necessarily need the resource requirements to be additive, only the monotonicity of the feasible configurations is sufficient, namely, if $\mathbf{k} \in \mathcal{K}$, and $\mathbf{k}' \leq \mathbf{k}$ (component-wise), then $\mathbf{k}' \in \mathcal{K}$. This will allow sub-additive resources as well, when the cumulative resource used by the VMs in a configuration could be less than the sum of the resources used individually [28].

Job and Reward Model. Jobs for various VM types arrive to the system over time. We can consider two models for jobs:

(i) *Revenue interpretation*: a job of type j is a request to create a new VM of type j.

(ii) *Service interpretation*: a job of type j is a request that must be served by an existing VM of type j in the system.

[1] The proofs of all Propositions and Theorems can be found in [26]

To simplify the formulations and use one model to capture both interpretations, we assume that each VM can serve at most one job at any time. As we will see, our algorithm works based on creating "*reserved VM slots*" in advance. Hence, serving a newly arrived type-j job can be interpreted as deploying a VM of type j in its reserved slot (revenue interpretation), or assigning it to an already deployed VM of type j in the slot (service interpretation).

Each job type j is associated with a reward u_j which represents its priority (service interpretation) or price paid per time unit of service (revenue interpretation).

We define the *feasible job placement* $\hat{\mathbf{k}} = (\hat{k}_1, \cdots, \hat{k}_J)$ to be the set of jobs that are simultaneously being served in a single server, where $\hat{k}_j$ corresponds to the number of type-j jobs. Note that by the definition of server configuration, it holds that $\hat{\mathbf{k}} \leq \mathbf{k}$, for some $\mathbf{k} \in \mathcal{K}$. Hence, $\mathbf{k} - \hat{\mathbf{k}}$ can be viewed as the reserved VM slots, where $k_j - \hat{k}_j$ is the number of reserved type-j VM slots. We use $\hat{\mathbf{k}}^\ell(t) = \hat{\mathbf{k}}$, when at time t, the job placement in server $\ell \in \mathcal{L}$ is $\hat{\mathbf{k}}$.

Traffic Model. Jobs of type j arrive according to a Poisson process of rate $\lambda_j L$, for a constant $\lambda_j > 0$. Once scheduled in a server (more accurately, in a reserved slot of type j), a job of type j requires an exponentially distributed service time with mean $1/\mu_j$, and generates *reward* at rate u_j during its service. We define the normalized workload of type-j jobs as $\rho_j := \lambda_j/\mu_j$ and the workload vector $\boldsymbol{\rho} = (\rho_j, j \in \mathcal{J})$.

Definition 2.1 (Configuration Reward). The reward $U(\mathbf{k})$ of a configuration $\mathbf{k} \in \mathcal{K}$ is defined as its total reward per unit time when its slots are full, i.e., $U(\mathbf{k}) := \sum_{j=1}^J u_j \mathbf{k}_j$.

Definition 2.2 (Configuration Ordering). For two vectors $\mathbf{k}, \mathbf{k}' \in \mathcal{K}$, we say $\mathbf{k} \succ \mathbf{k}'$, if either $U(\mathbf{k}) > U(\mathbf{k}')$, or $U(\mathbf{k}) = U(\mathbf{k}')$ and considering the smallest j for which $k_j \neq k'_j$, $k_j > k'_j$.

Definition 2.3 (MaxReward). Given a subset $\mathcal{K}_s \subseteq \mathcal{K}$, the maximum reward configuration of $\mathcal{K}_s$ is defined as

$$\textsc{MaxReward}(\mathcal{K}_s) := \arg\max_{\mathbf{k} \in \mathcal{K}_s} U(\mathbf{k}),$$

where ties are broken based on the ordering in Definition 2.2.

Definition 2.4 (State Variables). Consider the system with L servers. We use $X_{\mathbf{k}}^L(t)$ to denote the number of servers assigned to configuration $\mathbf{k} \in \mathcal{K}$ at time t. To distinguish between servers assigned to the same configuration $\mathbf{k}$, we *index* them from 1 to $X_{\mathbf{k}}^L(t)$, starting from the most recent server assigned to $\mathbf{k}$ (without loss of generality).

The system state at time t can then be described as

$$\mathbf{S}^L(t) := ((\mathbf{k}^\ell(t), \hat{\mathbf{k}}^\ell(t), c^\ell(t)), \ell \in \mathcal{L}), \tag{2}$$

where for each server $\ell \in \mathcal{L}$, $\mathbf{k}^\ell(t) \in \mathcal{K}$ is its configuration, $\hat{\mathbf{k}}^\ell(t)$, with $\hat{\mathbf{k}}^\ell(t) \leq \mathbf{k}^\ell(t)$, is its job placement, and $c^\ell(t)$ is its index among the servers with configuration $\mathbf{k}^\ell(t)$.

The number of jobs of type j in the system at time t is given by

$$Y_j^L(t) = \sum_{\ell \in \mathcal{L}} \hat{k}_j^\ell(t). \tag{3}$$

We also define the vectors $\mathbf{Y}^L(t) = (Y_j^L(t), j \in \mathcal{J})$, and $\mathbf{X}^L(t) = (X_{\mathbf{k}}^L(t), \mathbf{k} \in \mathcal{K})$. Clearly $\sum_{\mathbf{k} \in \mathcal{K}} X_{\mathbf{k}}^L(t) = L$ since there are L servers.

Optimization Objective. Given a Markov policy π, we define the expected reward of the policy per unit time as

$$F^{\pi}(L) = \lim_{t\to\infty} \mathbb{E}\Big[\sum_{j\in\mathcal{J}} Y_j^L(t)u_j\Big]. \tag{4}$$

Our goal is to maximize the expected reward, i.e.,

$$\text{maximize}_{\pi} F^{\pi}(L), \tag{5}$$

where the maximization is over all Markov scheduling policies π. Hence, when jobs are requests for VMs, this optimization is a revenue maximization, whereas when jobs are requests to be served by existing VMs, it is a weighted QoS maximization where each service is weighted by its priority.

Note that under any Markov policy, the system state $\mathbf{S}^L(t)$ is a continuous-time irreducible Markov chain over a finite state space, hence it is positive recurrent and (4) is well defined. Let $\mathbf{X}^L(\infty)$ and $\mathbf{Y}^L(\infty)$ be random vectors with the stationary distributions of $\mathbf{X}^L(t)$ and $\mathbf{Y}^L(t)$, respectively, as $t \to \infty$. Note that if $\mathbf{Y}^{\star}(t)$ is the number of jobs in an $M/M/\infty$ system in which every job is admitted, then $\mathbf{Y}^L(\infty)$ is stochastically dominated by $\mathbf{Y}^{\star}(\infty)$ whose stationary distribution is Poisson with mean $L\boldsymbol{\rho}$.

We study the problem (5) in the asymptotic regime where the number of servers $L \to \infty$, while the job arrival rates are $\lambda_j L$, $j \in \mathcal{J}$. Note that we do not make any assumption on the values of ρ_j.

Notice that as $t \to \infty$, the scaled stationary random variables satisfy $\frac{1}{L}\mathbf{X}^L(\infty) \le \mathbf{1}$ and $\frac{1}{L}\mathbf{Y}^L(\infty) \le \frac{1}{L}\mathbf{Y}^{\star}(\infty)$. This implies that the sequence of scaled random variables is tight [5], therefore the (random) limits $\mathbf{x}(\infty) := \lim_{L\to\infty} \frac{1}{L}\mathbf{X}^L(\infty)$, and $\mathbf{y}(\infty) := \lim_{L\to\infty} \frac{1}{L}\mathbf{Y}^L(\infty)$ exist along a subsequence of L. The limits satisfy $x_{\mathbf{k}}(\infty) \ge 0$, $\sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}}(\infty) = 1$, and $\mathbf{y}(\infty) \le \boldsymbol{\rho}$, $\mathbf{y}(\infty) \le \sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}}(\infty)\mathbf{k}$.

To unify the algorithm descriptions for revenue maximization and QoS maximization, in the rest of the paper, we use the term "*slot*" of type j to refer to the resource (equal to a VM of type j) reserved for one job of type j in a server. Filled slots have jobs already in them, while empty slots could accept jobs. Therefore, the term configuration applies to all the slots in a server, while placement applies to the filled slots in the server.

3 A STATIC OPTIMIZATION AND ITS GREEDY SOLUTION

Given a workload reference vector $\hat{\mathbf{Y}}^L = (\hat{Y}_j^L, j \in \mathcal{J})$, let $F^{\star}(L, \hat{\mathbf{Y}}^L)$ be the optimal value of the following linear program:

$$\max_{\mathbf{X},\mathbf{Y}} \quad \sum_j u_j Y_j \tag{6a}$$

$$\text{s.t.} \quad Y_j \le \hat{Y}_j^L, \ \forall j \in \mathcal{J} \tag{6b}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} X_{\mathbf{k}} k_j \ge Y_j, \ \forall j \in \mathcal{J} \tag{6c}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} X_{\mathbf{k}} = L, \quad X_{\mathbf{k}} \ge 0, \ \forall \mathbf{k} \in \mathcal{K} \tag{6d}$$

where $\mathbf{Y}$ is the vector of jobs in the system, and $\mathbf{X}$ is the vector of the number of servers assigned to each configuration. If we choose $\hat{\mathbf{Y}}^L = \boldsymbol{\rho}L$, this optimization will provide an upper bound on optimization (5), i.e., $F^{\pi}(L) \le F^{\star}(L, \boldsymbol{\rho}L)$, for any Markov policy π. The interpretation of the result is as follows. The average number of type-j jobs in the system cannot be more than its workload (Constraint (6b)), and further, it cannot be more than the average number of slots of type j in the servers (Constraint (6c)). The sum of number of servers in different configurations is L, so their average should also satisfy (6d).

As $L \to \infty$, the normalized objective value $\frac{1}{L}F^{\star}(L, \boldsymbol{\rho}L) \to U^{\star}[\boldsymbol{\rho}]$, which is the optimal value of the linear program below

$$\max_{\mathbf{x},\mathbf{y}} \quad \sum_j u_j y_j \tag{7a}$$

$$\text{s.t.} \quad y_j \le \rho_j, \forall j \in \mathcal{J} \tag{7b}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} k_j x_{\mathbf{k}} \ge y_j, \forall j \in \mathcal{J} \tag{7c}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}} = 1, \quad x_{\mathbf{k}} \ge 0, \ \forall \mathbf{k} \in \mathcal{K} \tag{7d}$$

where $x_{\mathbf{k}}$ can be interpreted as the ideal fraction of servers which should be in configuration $\mathbf{k}$ when L is large. Hence, one can consider a static reservation policy where the cloud cluster is partitioned and $\lfloor x_{\mathbf{k}} L \rfloor$ servers are assigned to each non-zero configuration $\mathbf{k} \in \mathcal{K}$ (and the rest of servers can be empty to save resource or used to serve more jobs). Then once a type-j job arrives, it will be routed to an empty slot of type j in one of the servers, if any, otherwise it is rejected. This will provide an asymptotic optimal policy since it achieves the normalized reward $U^{\star}[\boldsymbol{\rho}]$, as $L \to \infty$.

However, there are several issues with this approach: (i) solving optimization (6) or its relaxation (7) has a very high complexity, as the number of configurations is exponential in the number of job types J, and (ii) it requires knowing an accurate estimate of the workload $\boldsymbol{\rho}$ which might not be available.

We first address the complexity issue, by presenting a greedy solution, and analyze its asymptotic performance below.

3.1 Greedy Solution

We describe a greedy algorithm, called *Greedy Placement Algorithm* (GPA), for solving optimization (6).

GPA takes as input the workload reference vector $\hat{\mathbf{Y}}^L$, and returns an assignment vector $\hat{\boldsymbol{X}}^L$ which indicates which configurations should be used and in how many servers. The assignment consists of at most J configurations, which are found in J iterations. In each iteration i, GPA maintains a set of candidate job types $\mathcal{J}[i]$, and finds a configuration $\mathbf{k}[i]$. Initially $\mathcal{J}[1] = \mathcal{J}$. In iteration i:

(1) It finds $\mathbf{k}[i] = \textsc{MaxReward}(\mathcal{K}^{\mathcal{J}[i]})$, which is the configuration of highest reward among the configurations that have jobs from the set $\mathcal{J}[i]$, according to Definition 2.3.
(2) It computes the number of servers $\hat{X}^L_{\mathbf{k}[i]}$ that should be assigned to $\mathbf{k}[i]$, until at least one of the job types j, for which $k_j[i] > 0$, has no more jobs left, or there are no more unused servers left. We refer to this job type as $j^{\star}$.
(3) It then creates $\mathcal{J}[i+1]$ by removing job type $j^{\star}$ from $\mathcal{J}[i]$.

A pseudocode for GPA is given by Algorithm 1. We use the vector $\hat{\mathbf{X}}^L = (\hat{X}^L_{\mathbf{k}}, \mathbf{k} \in \mathcal{K})$ to denote the output of GPA, which has at most J non-zero elements corresponding to $\mathbf{k}[i]$, $i = 1, \ldots, J$.

We next define the limit of $\hat{\boldsymbol{X}}^L/L$ for input $\hat{\mathbf{Y}}^L = L\boldsymbol{\rho}$, as $L \to \infty$, which we refer to as *Global Greedy Assignment*. To describe

Algorithm 1 Greedy Placement Algorithm (GPA)

function GPA($\hat{\mathbf{Y}}$)
  $\mathbf{r} \leftarrow \hat{\mathbf{Y}}$ ▷ tracks the vector of number of jobs left
  $N \leftarrow L$ ▷ tracks the number of servers left
  $i \leftarrow 1, \mathcal{J}[1] = \mathcal{J}$
  while $\mathcal{J}[i] \neq \emptyset$ **do**
    $\mathbf{k}[i] \leftarrow \textsc{MaxReward}(\mathcal{K}^{\mathcal{J}[i]})$
    $j^\star \leftarrow \arg\min_{j:k_j[i]>0} \lceil \frac{r_j}{k_j[i]} \rceil$ ▷ break ties arbitrarily
    $\hat{X}_{\mathbf{k}[i]} \leftarrow \min\left(\lceil \frac{r_{j^\star}}{k_{j^\star}[i]} \rceil, N \right)$
    $\mathbf{r} \leftarrow \mathbf{r} - \hat{X}_{\mathbf{k}[i]} \mathbf{k}[i]$
    $N \leftarrow N - \hat{X}_{\mathbf{k}[i]}$
    $\mathcal{J}[i+1] \leftarrow \mathcal{J}[i] - \{j^\star\}$
    $i \leftarrow i + 1$
  return $\hat{X}_{\mathbf{k}[j]}, j = 1, \cdots, J$

this assignment, we first define a unique ordering of the job types through the following proposition.

Proposition 3.1. *For any permutation $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_J)$ of job types in $\mathcal{J}$, let $\mathcal{J}_j^\sigma := \{\sigma_j, \ldots, \sigma_J\}$, and $\mathbf{k}^{(j)} := \textsc{MaxReward}(\mathcal{K}^{\mathcal{J}_j^\sigma})$. Given a workload $\boldsymbol{\rho}$, there is a "unique" permutation $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_J)$ of job types, such that the following holds:*

1) $\forall j \in \mathcal{J}$, $k_{\sigma_j}^{(j)} > 0$, and there are constants $z^{(j)}[\boldsymbol{\rho}] \geq 0$, such that

$$\rho_{\sigma_j} = \sum_{\ell=1}^{j} k_{\sigma_j}^{(\ell)} z^{(\ell)}[\boldsymbol{\rho}], \tag{8}$$

2) for any two indexes $j, j' \in \mathcal{J}$, with $j < j'$, if

$$\rho_{\sigma_{j'}} = \sum_{\ell=1}^{j} k_{\sigma_{j'}}^{(\ell)} z^{(\ell)}[\boldsymbol{\rho}], \tag{9}$$

then we should have $\sigma_j < \sigma_{j'}$.

We omit the proof of the above proposition due to space constraint. The Global Greedy Assignment is defined as follows

Definition 3.2 (Global Greedy Assignment). Define the index $I_{\boldsymbol{\rho}} \leq J$ for which

$$\textstyle\sum_{i=1}^{I_{\boldsymbol{\rho}}-1} z^{(i)}[\boldsymbol{\rho}] < 1, \quad \sum_{i=1}^{I_{\boldsymbol{\rho}}} z^{(i)}[\boldsymbol{\rho}] \geq 1,$$

with the convention that $I_{\boldsymbol{\rho}} = J + 1$ if $\sum_{i=1}^{J} z^{(i)}[\boldsymbol{\rho}] < 1$. The *global greedy assignment* $\mathbf{x}^{(g)}[\boldsymbol{\rho}]$ is defined as

$$x_{\mathbf{k}^{(i)}}^{(g)}[\boldsymbol{\rho}] = \begin{cases} z^{(i)}[\boldsymbol{\rho}], & \text{for } i < I_{\boldsymbol{\rho}} \\ 0, & \text{for } i > I_{\boldsymbol{\rho}} \\ 1 - \sum_{j=1}^{i-1} x_{\mathbf{k}^{(j)}}^{(g)}[\boldsymbol{\rho}], & \text{for } i = I_{\boldsymbol{\rho}}, \end{cases} \tag{10}$$

where $\mathbf{k}^{(i)}$ and $z^{(i)}[\boldsymbol{\rho}]$, $i = 1, \ldots, J$, were defined in Proposition 3.1, and $\mathbf{k}^{(J+1)} := \mathbf{0}$ (empty configuration). We call the ordered configurations $\mathbf{k}^{(i)}$, $i = 1, \ldots, J+1$, the "global greedy configurations" of workload $\boldsymbol{\rho}$. For any configuration $\mathbf{k} \in \mathcal{K}$ not in global greedy configurations, $x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}] = 0$. When it is clear from the context, the dependency $[\boldsymbol{\rho}]$ will be omitted.

The following proposition states the connection between GPA and Global Greedy Assignment $x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}]$. We omit its proof due to space constraint.

Proposition 3.3. *Let $\hat{\mathbf{X}}^L = \text{GPA}(L\boldsymbol{\rho})$. Then*

$$\lim_{L\to\infty} \frac{\hat{X}_{\mathbf{k}}^L}{L} = x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}], \ \forall \mathbf{k} \in \mathcal{K}, \tag{11}$$

where $x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}]$ is the Global Greedy Assignment of Definition 3.2.

Note that clearly $\mathbf{x}^{(g)}[\boldsymbol{\rho}]$ is a feasible solution for optimization (7) and it is easy to see that its corresponding objective value is

$$U^{(g)}[\boldsymbol{\rho}] := \sum_{j=1}^{J} u_j \sum_{\ell=1}^{J} k_j^{(\ell)} x_{\mathbf{k}^{(\ell)}}^{(g)}[\boldsymbol{\rho}]. \tag{12}$$

It is also easy to see that in optimization (7) we can replace the inequality in (7c) with equality and the optimal value will not change. Let $\mathbf{x}^\star[\boldsymbol{\rho}]$ be one such optimal solution to optimization (7) for workload $\boldsymbol{\rho}$. Then the optimal objective value is

$$U^\star[\boldsymbol{\rho}] := \sum_{j\in\mathcal{J}} u_j \sum_{\mathbf{k}\in\mathcal{K}} k_j x_{\mathbf{k}}^\star[\boldsymbol{\rho}]. \tag{13}$$

The following corollary is immediate from Proposition 3.3.

Corollary 3.4. *Let $F^{GPA}(L, \boldsymbol{\rho}L)$ be the total reward of* GPA *in the system with L servers given reference workload $\hat{\mathbf{Y}}^L = \boldsymbol{\rho}L$. Then*

$$\lim_{L\to\infty} \frac{F^{GPA}(L, \boldsymbol{\rho}L)}{F^\star(L, \boldsymbol{\rho}L)} = \frac{U^{(g)}[\boldsymbol{\rho}]}{U^\star[\boldsymbol{\rho}]}.$$

The theorem below bounds the above ratio.

Theorem 3.5. *The global greedy assignment $\mathbf{x}^{(g)}[\boldsymbol{\rho}]$ provides at least $\frac{1}{2}$ of the optimal normalized reward, i.e., $\frac{U^{(g)}[\boldsymbol{\rho}]}{U^\star[\boldsymbol{\rho}]} \geq \frac{1}{2}$, $\forall \boldsymbol{\rho} \geq 0$.*

Theorem 3.5 can be improved when job types and rewards satisfy a monotone greedy property described next.

Definition 3.6. We say the job types and the rewards have *monotone greedy* property if for any two instances of the optimization (7) with $\boldsymbol{\rho}_1 \geq \boldsymbol{\rho}_2$, $U^{(g)}[\boldsymbol{\rho}_1] \geq U^{(g)}[\boldsymbol{\rho}_2]$.

The next theorem describes the improved bound when the monotone greedy property holds.

Theorem 3.7. *If job types and rewards satisfy the monotone greedy property, then, for any $\boldsymbol{\rho}$, $\frac{U^{(g)}[\boldsymbol{\rho}]}{U^\star[\boldsymbol{\rho}]} \geq 1 - 1/e$.*

Proposition 3.8. *The worst-case ratio of $U^{(g)}[\boldsymbol{\rho}]/U^\star[\boldsymbol{\rho}]$ is not greater than $1 - 1/e$.*

Proof of Proposition 3.8 is by construction of an adversarial example and is omitted due to space constraint.

Hence, the global greedy assignment achieves a factor within $1/2$ to $1 - 1/e$ of the optimal normalized reward in "all" the cases. However, GPA($\boldsymbol{\rho}L$) requires the knowledge of $\boldsymbol{\rho}$. In the next section, we propose a dynamic reservation algorithm that is appropriate for use in online settings without the knowledge of $\boldsymbol{\rho}$. Its achievable normalized reward still converges to that of the global greedy assignment and it can also adapt to changes in the workload.

Algorithm 2 DRA: Dynamic Reservation Algorithm

function CRA($\hat{\mathbf{Y}}^L, \mathbf{X}^L$)
  $\hat{\mathbf{X}}^L \leftarrow$ GPA $(\hat{\mathbf{Y}}^L)$.
  Set rank of all servers to $J+1$.
  $I^\star \leftarrow J$
  for $i = 1$ to J **do** ▷ J configurations found in GPA
    $Z \leftarrow 0,\ c \leftarrow X^L_{\mathbf{k}[i]}$ ▷ c is the index of server
    while $Z < \hat{X}^L_{\mathbf{k}[i]}$ **do**
      $Z \leftarrow Z+1, c \leftarrow c-1$
      if $c \le 0$ **then**
        if $\ell_e \neq \emptyset$ **then**
          Set rank of ℓ_e to i.
          Reassign configuration of ℓ_e to $\mathbf{k}[i]$.
        else
          $I^\star \leftarrow \min(I^\star, i)$
      else
        Set rank of $\ell_{\mathbf{k}[i],c}$ to i.

procedure ARRIVAL(j, t) ▷ Type-j arrival at time t
  if $\mathsf{AG}_j \neq \emptyset$ **then**
    Schedule job in AG_j.
    CRA $(\mathbf{Y}^L(t) + g(L)\mathbf{1}, \mathbf{X}^L(t))$
  else
    Reject job.

procedure DEPARTURE(j, t) ▷ Type-j departure at time t
  if $\mathsf{RG}_j \neq \emptyset$ and the slot emptied is in Accept Group **then**
    Migrate the job in RG_j to the slot that emptied.
  CRA $(\mathbf{Y}^L(t) + g(L)\mathbf{1}, \mathbf{X}^L(t))$

4 DYNAMIC RESERVATION ALGORITHM

We present a Dynamic Reservation Algorithm, called DRA, which makes admission decisions and configuration assignments, without the knowledge $\boldsymbol{\rho}$. We first introduce the following notations:

- Recall the indexing of servers in the same configuration as in Definition 2.4. We use $\ell_{\mathbf{k},i}$ to refer to the server with configuration $\mathbf{k}$ and index i.
- A key parameter of DRA is the *reservation factor* $g(L)$. It is the number of empty slots (safety margin) that the algorithm ideally wants to reserve for each job type if possible. For later analysis, we assume that $g(L) = \omega(\log(L))$, and is $o(L)$.

The configuration assignment occurs at *update times*. To simplify the analysis, we consider update times to be times when a job is admitted to or departs from the system. To avoid preemptions, only servers that are empty (have no jobs running) can be assigned to a new configuration.

At update time t, DRA updates the workload reference vector $\hat{\mathbf{Y}}^L(t)$ as

$$\hat{\mathbf{Y}}^L(t) = \mathbf{Y}^L(t) + g(L)\mathbf{1}, \tag{14}$$

where $\mathbf{Y}^L(t)$ in the vector of jobs in the system, *after* any job admission or job departure at time t. $g(L)$ is the reservation factor as defined earlier.

Then DRA classifies the servers into two groups: *Accept Group* (AG) and *Reject Group* (RG). Servers in Accept Group keep their current configurations and DRA attempts to have all their slots filled by scheduling new jobs in them, while servers in Reject Group do not have desirable configurations and DRA attempts to make them empty, by not scheduling new jobs in them and possibly migrating their jobs to servers in Accept Group, so they can be reassigned to other configurations.

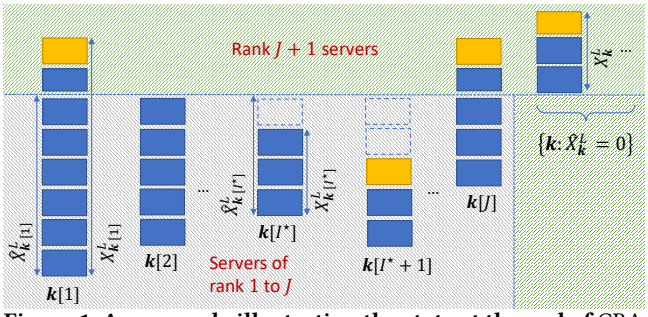

Figure 1: An example illustrating the state at the end of CRA**. Servers in each configuration are stacked from largest to smallest index.** $\mathbf{k}[1], \ldots, \mathbf{k}[J]$ **are the configurations returned by** GPA**. The dashed boxes indicate how many more servers need to be reassigned to a respective configuration to match the solution of** GPA **(horizontal line).** $I^\star$ **is the first** i **for which** $X^L_{\mathbf{k}[i]} < \hat{X}^L_{\mathbf{k}[i]}$ **at the end of the procedure. Orange servers are the servers of Reject Group.**

A pseudocode for DRA is given in Algorithm 2. It has three main components which we describe in detail below:

Classification and Reassignment Algorithm (CRA**).** This is the subroutine used by DRA to classify servers and possibly reassign some of them. It attempts to greedily reduce the disparity between the configuration assignment in the system $\mathbf{X}^L(t)$ and the output of GPA $\hat{\mathbf{X}}^L(t) = \text{GPA}(\hat{\mathbf{Y}}^L(t))$. To do so, it assigns *ranks* to servers in different configurations, which range from 1 to $J+1$. Initially, all servers are assigned rank $J+1$. Any empty server of rank $J+1$ can be reassigned to reduce the disparity between $\mathbf{X}^L(t)$ and $\hat{\mathbf{X}}^L(t)$. We use ℓ_e to denote one of empty rank $J+1$ servers, and if no such server exists $\ell_e = \emptyset$.

Iterating over configurations $\mathbf{k}[i]$ found by GPA, for $i = 1, \ldots, J$:

- If $X^L_{\mathbf{k}[i]} < \hat{X}^L_{\mathbf{k}[i]}$, it increases $X^L_{\mathbf{k}[i]}$ by reassigning any ℓ_e to $\mathbf{k}[i]$, until either (i) it matches $\hat{X}^L_{\mathbf{k}[i]}$, or (ii) $\ell_e = \emptyset$. In either case, all servers of configuration $\mathbf{k}[i]$ get rank i.
- If $X^L_{\mathbf{k}[i]}(t) \ge \hat{X}^L_{\mathbf{k}[i]}$, it assigns rank i to all servers of configuration $\mathbf{k}[i]$ with indexes greater than $X^L_{\mathbf{k}[i]}(t) - \hat{X}^L_{\mathbf{k}[i]}(t)$.

We use $I^\star(t)$ to denote the first i for which $X^L_{\mathbf{k}[i]}$ cannot be matched to $\hat{X}^L_{\mathbf{k}[i]}$, i.e. the first i at which $\ell_e = \emptyset$. If all configurations are matched, then $I^\star(t) = J$. At the end of CRA, servers with rank greater than $I^\star(t)$ and index 1 in any configuration are classified as Reject Group, while the rest of the servers are classified as Accept Group.

See Figure 1 for an illustrative example for the state of CRA.

Scheduling Arriving Job. When DRA needs to schedule an arriving job of type j, it places the job in one of the servers of Accept Group with empty type-j slot. If no such server exists, the

job is rejected. We use AG_j to denote one of the servers of Accept Group with empty type-j slot. If no such server exists $\mathsf{AG}_j = \emptyset$.

Migrating Job after Departure. Let RG_j denote the highest rank server among the Reject Group servers with type-j jobs. If no such server exists, $\mathsf{RG}_j = \emptyset$.

If a type-j job departs from a server in Accept Group, DRA migrates one of the type-j jobs from RG_j to the slot that emptied because of the departure, if $\mathsf{RG}_j \neq \emptyset$.

Initialization. Considering initialization at time 0, if servers do not have configurations, but have jobs in them, we initialize $\mathbf{k}^\ell(0) = \hat{\mathbf{k}}^\ell(0)$, i.e., the configuration of each server ℓ is set to its job placement. If servers have configurations, we keep their existing configuration. Indexing among the servers of a configuration can be arbitrary. We then run CRA that performs classification and reassigns any possibly empty servers.

The following theorem states the main result regarding DRA.

Theorem 4.1. *Let $F^{DRA}(L)$ be the expected reward under* DRA *and $F^\star(L)$ be the optimal expected reward in optimization (5). Then*

$$\lim_{L\to\infty} \frac{F^{DRA}(L)}{F^\star(L)} \geq \frac{1}{2}.$$

Further, under the monotone greedy property (Definition 3.6),

$$\lim_{L\to\infty} \frac{F^{DRA}(L)}{F^\star(L)} \geq 1 - \frac{1}{e}.$$

The proof of Theorem 4.1 is based on analysis of fluid limits and the choice of a suitable Lyapunov function.

5 CONCLUSIONS

In this paper, we proposed a VM reservation and admission policy that operates in an online manner and can guarantee at least $1/2$ (and under certain monotone property, $1 - 1/e$) of the optimal expected reward. Assumptions such as Poisson arrivals and exponential service times are made to simplify the analysis, and the policy itself does not rely on this assumption. The policy strikes a balance between good VM packing and serving high priority VM requests, by maintaining only a small number $g(L) = \omega(\log L)$ of reserved VM slots at any time.

Although we considered that the policy classifies and reassigns servers at arrival and departure events, this was only to simplify the analysis, and in practice CRA can make such updates periodically, by factoring all arrival or departures in the past period in its input for the current period. Further, if a more accurate estimate of the workload is available, we can incorporate that estimate in the vector $\hat{\mathbf{Y}}$ used by DRA, to improve the convergence time. Moreover, the policy can be extended to a multi-pool server system, where constant fractions of servers belong to different server types. We postpone the details to a future work.

REFERENCES

[1] Amazon AWS 2019. *Amazon Web Services (AWS).* https://aws.amazon.com/

[2] AWS container 2019. *Amazon AWS Containers.* https://aws.amazon.com/containers/

[3] AWS SLA 2019. *Amazon AWS Service Level Agreements (SLAs).* //https://aws.amazon.com/legal/service-level-agreements/

[4] N. G. Bean, R. J. Gibbens, and S. Zachary. 1995. Asymptotic analysis of single resource loss systems in heavy traffic, with applications to integrated networks. *Advances in Applied Probability* 27, 1 (1995), 273–292. https://doi.org/10.2307/1428107

[5] Patrick Billingsley. 2008. *Probability and measure.* John Wiley & Sons.

[6] Antonio Corradi, Mario Fanelli, and Luca Foschini. 2014. VM consolidation: A real case based on OpenStack Cloud. *Future Generation Computer Systems* 32 (2014), 118–127.

[7] Javad Ghaderi, Yuan Zhong, and Rayadurgam Srikant. 2014. Asymptotic optimality of BestFit for stochastic bin packing. *ACM SIGMETRICS Performance Evaluation Review* 42, 2 (2014), 64–66.

[8] Mostafa Ghobaei-Arani, Sam Jabbehdari, and Mohammad Ali Pourmina. 2018. An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Computer Systems* 78 (2018), 191–210.

[9] Google Cloud 2019. *Google cloud computing services.* https://cloud.google.com/

[10] Google Kubernetes 2019. *Kubernetes at Google Cloud.* https://https://cloud.google.com/kubernetes/

[11] Yang Guo, Alexander Stolyar, and Anwar Walid. 2018. Online vm auto-scaling algorithms for application hosting in a cloud. *IEEE Transactions on Cloud Computing* (2018).

[12] Rui Han, Li Guo, Moustafa M Ghanem, and Yike Guo. 2012. Lightweight resource scaling for cloud applications. In *EEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012).* 644–651.

[13] PJ Hunt and TG Kurtz. 1994. Large loss networks. *Stochastic Processes and their Applications* 53, 2 (1994), 363–378.

[14] PJ Hunt, CN Laws, et al. 1997. Optimization via trunk reservation in single resource loss systems under heavy traffic. *The Annals of Applied Probability* 7, 4 (1997), 1058–1079.

[15] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. 2013. Optimal cloud resource auto-scaling for web applications. In *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 58–65.

[16] A Karthik, Arpan Mukhopadhyay, and Ravi R Mazumdar. 2017. Choosing among heterogeneous server clouds. *Queueing Systems* 85, 1-2 (2017), 1–29.

[17] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. Multidimensional knapsack problems. In *Knapsack problems*. Springer, 235–283.

[18] Frank P Kelly. 1991. Loss networks. *The annals of applied probability* (1991), 319–378.

[19] Siva Theja Maguluri, Rayadurgam Srikant, and Lei Ying. 2012. Stochastic models of load balancing and scheduling in cloud computing clusters. In *2012 Proceedings IEEE Infocom*. IEEE, 702–710.

[20] Siva Theja Maguluri, Rayadurgam Srikant, and Lei Ying. 2014. Heavy traffic optimal resource allocation algorithms for cloud computing clusters. *Performance Evaluation* 81 (2014), 20–39.

[21] Ming Mao, Jie Li, and Marty Humphrey. 2010. Cloud auto-scaling with deadline and budget constraints. In *IEEE/ACM International Conference on Grid Computing*. 41–48.

[22] Microsoft Azure 2019. *Microsoft cloud computing service.* https://azure.microsoft.com/

[23] Arpan Mukhopadhyay, A Karthik, Ravi R Mazumdar, and Fabrice Guillemin. 2015. Mean field and propagation of chaos in multi-class heterogeneous loss models. *Performance Evaluation* 91 (2015), 117–131.

[24] Konstantinos Psychas and Javad Ghaderi. 2017. On non-preemptive VM scheduling in the cloud. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 2 (2017), 35.

[25] Konstantinos Psychas and Javad Ghaderi. 2018. Randomized Algorithms for Scheduling Multi-Resource Jobs in the Cloud. *IEEE/ACM Transactions on Networking* 26, 5 (2018), 2202–2215.

[26] Konstantinos Psychas and Javad Ghaderi. 2020. A Theory of Auto-Scaling for Resource Reservation in Cloud Services. *arXiv preprint arXiv:2005.13744* (2020).

[27] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–33.

[28] Safraz Rampersaud and Daniel Grosu. 2014. A sharing-aware greedy algorithm for virtual machine maximization. In *IEEE 13th International Symposium on Network Computing and Applications*. 113–120.

[29] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *IEEE 4th International Conference on Cloud Computing*. 500–507.

[30] Weijia Song, Zhen Xiao, Qi Chen, and Haipeng Luo. 2013. Adaptive resource provisioning for the cloud using online bin packing. *IEEE Trans. Comput.* 63, 11 (2013), 2647–2660.

[31] Alexander L Stolyar. 2017. Large-scale heterogeneous service systems with general packing constraints. *Advances in Applied Probability* 49, 1 (2017), 61–83.

[32] Alexander L Stolyar and Yuan Zhong. 2015. Asymptotic optimality of a greedy randomized algorithm in a large-scale service system with general packing constraints. *Queueing Systems* 79, 2 (2015), 117–143.

[33] Ward Whitt. 1985. Blocking when service is required from several facilities simultaneously. *AT&T technical journal* 64, 8 (1985), 1807–1856.

[34] Qiaomin Xie, Xiaobo Dong, Yi Lu, and Rayadurgam Srikant. 2015. Power of d choices for large-scale bin packing: A loss model. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 321–334.