Stochastic Systems

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

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

Konstantinos Psychas, Javad Ghaderi

To cite this article:
Konstantinos Psychas, Javad Ghaderi (2022) A Theory of Auto-Scaling for Resource Reservation in Cloud Services. Stochastic Systems

Published online in Articles in Advance 01 Feb 2022

. https://doi.org/10.1287/stsy.2021.0091

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

http://pubsonline.informs.org/journal/stsy

STOCHASTIC SYSTEMS
Articles in Advance, pp. 1–26
ISSN 1946-5238 (online)

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

Konstantinos Psychas,[a] **Javad Ghaderi**[a]

[a] Department of Electrical Engineering, Columbia University, New York, New York 10027
Contact: kp2547@columbia.edu, https://orcid.org/0000-0002-5655-4868 (KP); jghaderi@columbia.edu, https://orcid.org/0000-0001-8038-550X (JG)

Received: January 5, 2021
Revised: August 25, 2021
Accepted: November 17, 2021
Published Online in Articles in Advance: February 1, 2022

https://doi.org/10.1287/stsy.2021.0091

Abstract. We consider a distributed server system consisting of a large number of servers, each with limited capacity on multiple resources (CPU, memory, 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 it. 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 (VMs) or containers that reserve resources for various services, and rewards represent service priority of requests or price paid per time unit of service. 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.

Funding: This research was supported by the National Science Foundation [Grants CNS-1717867 and CNS-1652115].

Supplemental Material: The online appendices are available at https://doi.org/10.1287/stsy.2021.0091.

Keywords: scheduling • loss systems • fluid limits • resource allocation • cloud computing

1. Introduction

There has been a rapid migration of computing, storage, applications, and other services to cloud. By using the cloud (Amazon Web Services 2020c, Google Cloud 2020a, Microsoft Azure 2020), 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 (Amazon Web Services 2020a, Google Cloud 2020c) 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 quality of service (QoS) guarantees (in service level agreements) (Amazon Web Services 2020d) 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. For example, in Amazon EC2 auto-scaling (Amazon Web Services 2020b), clients can define simple rules to launch or terminate VM instances as their application demand increases or decreases. Various predictive and reactive schemes have been proposed for dynamically allocating VMs to different services (Mao et al. 2010, Roy et al. 2011, Han et al. 2012, Jiang et al. 2013, Ghobaei-Arani et al. 2018, Qu et al. 2018); 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 that is known to significantly improve efficiency and scalability (Song et al. 2013, Corradi et al. 2014). For instance, suppose a CPU-intensive VM, a disk-intensive VM, and a memory-intensive VM are located on three individual servers (for our purpose, we use the terms VM and container interchangeably). The cloud operator can pack these VMs in a single server to fully use its resources along CPU, disk, and memory, and then the two unused servers can be used to pack additional VMs and serve more requests. However, in the absence of an accurate

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

estimate of the workload, or when the workload varies over time and space, it is not clear how many VM instances an application launches and which VMs must be packed in which servers to ensure efficiency.

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 multidimensional 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. Finding the right packing for a given workload is a hard combinatorial problem (related to multidimensional knapsack; Kellerer et al. 2004). The absence of accurate estimate of workload (VM traffic rates and service durations) makes the problem even more challenging. For instance, consider a simple scenario with three types of VMs with the following (CPU, memory) requirement and rewards: (0.6, 0.6) with reward 4, (0.7, 0.1) with reward 3, and (0.1, 0.7) with reward 3. Server's capacity is normalized to (1,1). Hence, a server can accommodate a single (0.6, 0.6) VM or pack one (0.7, 0.1) VM and one (0.1, 0.7) VM together. Suppose there is one empty server, and a (0.6, 0.6) VM request arrives. Should we admit this request and receive a reward of 4, or reserve the server to pack one (0.7, 0.1) VM and one (0.1, 0.7) VM in the future, which can potentially yield a maximum reward of 6?

This problem is related to the *online multiple knapsack* problem, in which there is a set of bins of finite capacity, items with various sizes and profits arrive one by one, and the goal is to pack them in an *online* manner into the bins so as to maximize their total profit. In general, this problem does not have any competitive (constant approximation) algorithm (Marchetti-Spaccamela and Vercellis 1995), even when items are allowed to be removed from any bin at any time. Hence, proposed competitive algorithms focus on more restricted cases of the problem (Iwama and Taketomi 2002, Cygan et al. 2016).

In this paper, we study a stochastic version of the problem in an asymptotic regime, where the number of servers L is large, 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 in loss networks (Whitt 1985, Kelly 1991, Hunt and Kurtz 1994, Hunt et al. 1997, Mukhopadhyay et al. 2015, Xie et al. 2015, Karthik et al. 2017), and it has been shown that algorithms with good performance in such a regime also show good performance in realistic systems and arrival rates. We do not make any assumption on the value of $\boldsymbol{\rho}$; however, notice that 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; see Remark 3), 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.

In this paper, we use the term migration in a generic way to describe the transfer of a VM from a server to another without interrupting its processing. In practice, this can be a live migration (Jo et al. 2020, Le 2020), or in the case of stateless services (Amazon 2021, RedHat 2021), it can translate to termination of the VM instance and launching another one in a different server.

1.1. Related Work

There is classical work on large loss networks (Kelly 1991, Hunt and Kurtz 1994, Bean et al. 1995, Hunt et al. 1997), 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). This makes the problem significantly more difficult, because, due to resource fragmentation when packing VMs in servers, the resources of servers *cannot* be viewed as one giant pool; hence, our policy not only needs to make admission decisions, but also decide in which server to place the admitted VM. Moreover, VMs have multidimensional resource as opposed to one-dimensional calls (bandwidth). If we restrict that every server can fit exactly one VM, our policy reduces to classical trunk reservation.

There has been past work on resource allocation in the cloud (Maguluri et al. 2012, 2014; Stillwell et al. 2012; Zhao et al. 2015; Psychas and Ghaderi 2017, 2018; Shi et al. 2018) and stochastic bin packing (Gupta and Radovanovic 2012, Stolyar 2013, Stolyar and Zhong 2013, Ghaderi et al. 2014, Stolyar and Zhong 2015); however, their models or objectives are different from ours. The works of Maguluri et al. (2012), Psychas and Ghaderi (2017),

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Maguluri et al. (2014), and Psychas and Ghaderi (2018) 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; that is, each VM request upon arrival has to be served immediately, and otherwise it is lost. The recent works of Stolyar and Zhong (2015), Stolyar (2013), Stolyar and Zhong (2013), and Ghaderi et al. (2014) 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 by Guo et al. (2018) 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, in the limit as the number of servers becomes large. In this regime, we have to address complex fluid limit behaviors, especially when the load is above the system capacity and VMs have different priorities.

The works of Xie et al. (2015), Karthik et al. (2017), Mukhopadhyay et al. (2015), and Stolyar (2017) study the blocking probability in a large-scale server system where *all VMs have the same reward*. The work of Stolyar (2017) assumes a subcritical system load and only shows local stability of fluid limits. The works of Xie et al. (2015), Karthik et al. (2017), and Mukhopadhyay et al. (2015) show that, under a power-of-d choices routing, the blocking probability drops much faster compared with 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 nonzero 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.

We remark that in real clouds, servers are monitored periodically, for resource management, security, recovery, billing, and so on (Aceto et al. 2013, Apache Software Foundation 2019); hence, scheduling decisions can be made based on available information about the global system state.

1.2. 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 $1-e^{-1}$ of the optimal expected reward, as the number of servers $L \to \infty$. Furthermore, simulations suggest that, for real cloud VM instances, the achieved ratio is in fact very close to one.

The main features of our policy and analysis technique can be summarized as follows.

- **Dynamic Reservation.** The policy *reserves slots* for VMs in advance. A slot for a VM type will reserve the VM's required resources on a specific server. An incoming VM request then will be admitted if there is enough reservation in the system, in which case it will fill an empty slot of that type. The policy effectively tracks a low-complexity greedy packing of existing VM requests in the system while maintaining only a small number $g(L) = \omega(\log L)$ of empty slots (e.g., $(\log L)^{1+\varepsilon}$), for VM types that have high priority at the current time. The reservation policy is *robust* and can automatically adapt to changes in the workload based on requests in the system and new arriving requests, without the knowledge of $\boldsymbol{\rho}$.
- **Analysis Technique.** Our proofs rely on analysis of fluid limits under the proposed policy; however, a major difficulty happens when the workload is above the critical load. In this regime the slot reservation process evolves at a much faster time-scale compared with the fluid-limit processes of the VMs and of the servers in different packing configurations in the system. To describe the behavior of fluid limits, we devise a careful analysis based on averaging the behavior of fluid-scale process over *small intervals of length* $\omega(\log L/L)$. We then introduce a *Lyapunov function* based on a linear program. It is designed to have a unique maximizer at a global greedy solution and determines the convergence properties of our policy in steady state.

1.3. Basic Mathematical Notations

For two positive-valued functions $x(n)$ and $y(n)$, with $n \in \mathbb{N}$, we write $x(n) = o(y(n))$ if $\lim_{n\to\infty} x(n)/y(n) = 0$, and $x(n) = \omega(y(n))$ if $y(n) = o(x(n))$. $\mathbb{1}(\cdot)$ is the indicator function. The term $\mathbf{e}_j$ denotes the jth basis vector. The terms $f(t^-)$ and $f(t^+)$ are the limits of $f(x)$ as $x \to t$ from left and right, respectively, $\mathbb{R}_+$ is the set of nonnegative real numbers, and $(\cdot)^+ = \max\{\cdot, 0\}$.

2. Model and Definitions

2.1. 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.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

2.2. VM Model

There is a collection of VM types denoted by the set $\mathcal{J}$. The VM types are indexed in arbitrary order from one 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 dth 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 is 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 $\mathcal{J}' \subseteq \mathcal{J}$, that is,

$$\mathcal{K}^{\mathcal{J}'} = \{\mathbf{k} \in \mathcal{K} : k_j = 0, \ \forall j \notin \mathcal{J}'\}. \tag{2.1}$$

We use $K < \infty$ to denote the maximum number of VMs that can fit in a server. We use $\mathbf{k}^\ell(t) = \mathbf{k}$ to denote that, at time t, server $\ell \in \mathcal{L}$ has configuration $\mathbf{k}$.

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 subadditive 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 (Rampersaud and Grosu 2014).

2.3. 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.

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. 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 $\hat{\mathbf{k}}$.

2.4. 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 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, that is, $U(\mathbf{k}) := \sum_{j=1}^J u_j \mathbf{k}_j$.

Definition 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 3 (MaxReward). Given a subset $\mathcal{K}_s \subseteq \mathcal{K}$, the maximum reward configuration of $\mathcal{K}_s$ is defined as

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

where ties are broken based on the ordering in Definition 2.

Definition 4 (System 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 one to $X_{\mathbf{k}}^L(t)$, starting from the most recent server assigned to $\mathbf{k}$ (without loss of generality).

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

The system state at time t can then be described as

$$\mathbf{S}^L(t) := ((\mathbf{k}^\ell(t), \hat{\mathbf{k}}^\ell(t), c^\ell(t)), \ell \in \mathcal{L}), \tag{2.2}$$

where for each server $\ell \in \mathcal{L}$, $\mathbf{k}^\ell(t) \in \mathcal{K}$ is its configuration, $\hat{\mathbf{k}}^\ell(t)$, with $\hat{\mathbf{k}}^\ell(t) \le \mathbf{k}^\ell(t)$, 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

$$Y_j^L(t) = \sum_{\ell \in \mathcal{L}} \hat{k}_j^\ell(t). \tag{2.3}$$

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 $\sum_{\mathbf{k} \in \mathcal{K}} X_{\mathbf{k}}^L(t) = L$ because there are L servers.

Table 1 summarizes the essential notations used in the model and rest of the paper.

2.5. Optimization Objective

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

$$F^\pi(L) = \lim_{t \to \infty} \mathbb{E}\left[\sum_{j \in \mathcal{J}} Y_j^L(t) u_j\right]. \tag{2.4}$$

Our goal is to maximize the expected reward, that is,

$$\text{maximize}_\pi F^\pi(L), \tag{2.5}$$

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.

Without loss of generality, we can assume that under any Markov policy, the system state $\mathbf{S}^L(t)$ is a continuous-time irreducible Markov chain over a finite state space.[1] It is also aperiodic. Hence, the stationary distributions of $\mathbf{X}^L(t)$ and $\mathbf{Y}^L(t)$ exist, as $t \to \infty$.

We study the problem (2.5) in the asymptotic regime where the number of servers $L \to \infty$, whereas the job arrival rates are λ_j^L, $j \in \mathcal{J}$. We do not make any assumption on the values of ρ_j. Let $F^{\text{opt}}(L)$ be the optimal expected reward in optimization (2.5). Then policy π is asymptotically γ-competitive if, for a positive constant $\gamma \le 1$,

$$\liminf_{L \to \infty} \frac{F^\pi(L)}{F^{\text{opt}}(L)} \ge \gamma.$$

As we will see, to show the asymptotic performance of our algorithm π, we will find a lower bound on the scaled expected reward $\frac{1}{L}F^\pi(L)$ and an upper bound on $\frac{1}{L}F^{\text{opt}}(L)$, as $L \to \infty$.

Table 1. Notations

Notation	Definition
$\mathcal{J}$, J	Set of all VM types, number of VM types
$\mathcal{K}$, $\mathcal{K}^{\mathcal{J}'}$	Set of feasible configurations, set of feasible configurations with VMs from subset $\mathcal{J}'$
C, $C_{\boldsymbol{\rho}}^{(g)}$	Cardinality of feasible configuration set, cardinality of greedy configuration set
$\mathbf{k}$, $\mathbf{k}^\ell(t)$	A feasible configuration, a feasible configuration assigned to server ℓ at time t
$\hat{\mathbf{k}}$, $\hat{\mathbf{k}}^\ell(t)$	A job placement, jobs in server ℓ at time t
K	Maximum number of jobs in any feasible configuration
$\mathcal{L}$, L	Set of servers, number of servers
$c^\ell(t)$	Index of server ℓ among the servers with configuration $\mathbf{k}^\ell(t)$
$\mathbf{S}^L(t)$	System state at time t: $((\mathbf{k}^\ell(t), \hat{\mathbf{k}}^\ell(t), c^\ell(t)), \ell \in \mathcal{L})$
$X_{\mathbf{k}}^L(t)$	Number of servers with assigned configuration $\mathbf{k}$ in a system of L servers
$Y_j^L(t)$	Number of VMs of type j in a system of L servers
ρ_j	Workload of type j VMs
$\mathcal{K}^{(g)}$	Greedy configuration set
$\mathbf{k}^{(i)}$	ith global greedy configuration, defined in Proposition 1
$\bar{\mathbf{k}}^{(i)}$	ith configuration in $\mathcal{K}^{(g)}$, based on their reward in descending order
$\mathcal{J}[i]$	Set of VM types in iteration i of GPA ($\mathcal{J}[1] = \mathcal{J}$)
$\mathbf{k}[i]$	Configuration found by GPA at iteration i
$g(L)$	Reservation factor (ideal number of empty slots)

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

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 $t \to \infty$. The following observations will be useful when working with $\mathbf{X}^L(\infty)$ and $\mathbf{Y}^L(\infty)$ and their scaled versions $\frac{1}{L}\mathbf{X}^L(\infty)$ and $\frac{1}{L}\mathbf{Y}^L(\infty)$. If $\bar{\mathbf{Y}}(t)$ 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 $\bar{\mathbf{Y}}(\infty)$ whose stationary distribution is Poisson with mean $L\boldsymbol{\rho}$ (Bolch et al. 2006). Also, the scaled stationary random variables satisfy $\frac{1}{L}\mathbf{X}^L(\infty) \le \mathbf{1}$ and $\frac{1}{L}\mathbf{Y}^L(\infty) \le \frac{1}{L}\bar{\mathbf{Y}}(\infty)$. This implies that the sequence of scaled random variables is tight (Billingsley 2008); 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, whereas empty slots could accept jobs. Therefore, the term configuration applies to all the slots in a server, whereas placement applies to the filled slots in the server.

3. Static Optimization and Its Greedy Solution

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

$$\max_{\mathbf{X},\mathbf{Y}} \quad \sum_j u_j Y_j \tag{3.1a}$$

$$\text{s.t} \quad Y_j \le \hat{Y}^L_j, \qquad \forall j \in \mathcal{J} \tag{3.1b}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} X_{\mathbf{k}} k_j \ge Y_j, \quad \forall j \in \mathcal{J} \tag{3.1c}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} X_{\mathbf{k}} = L \tag{3.1d}$$

$$X_{\mathbf{k}} \ge 0, \qquad \forall \mathbf{k} \in \mathcal{K}, \tag{3.1e}$$

where $\mathbf{Y}$ is the vector of number of jobs of each type in the system, and $\mathbf{X}$ is the vector of 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 (2.5), that is, $F^\pi(L) \le F^\star(L, \boldsymbol{\rho}L)$, 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 (3.1b) with $\hat{Y}^L_j = L\rho_j$), and furthermore, it cannot be more than the average number of slots of type j in the servers (Constraint (3.1c)). The sum of number of servers in different configurations is L, so their average should also satisfy (3.1d).

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:

$$\max_{\mathbf{x},\mathbf{y}} \quad \sum_j u_j y_j \tag{3.2a}$$

$$\text{s.t} \quad y_j \le \rho_j, \qquad \forall j \in \mathcal{J} \tag{3.2b}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} k_j x_{\mathbf{k}} \ge y_j, \quad \forall j \in \mathcal{J} \tag{3.2c}$$

$$\sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}} = 1 \tag{3.2d}$$

$$x_{\mathbf{k}} \ge 0, \qquad \forall \mathbf{k} \in \mathcal{K}, \tag{3.2e}$$

where $x_{\mathbf{k}}$ can be interpreted as the ideal fraction of servers that 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 nonzero 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 because it achieves the normalized reward $U^\star[\boldsymbol{\rho}]$, as $L \to \infty$.

However, there are several issues with this approach: (i) solving Optimization (3.1) or its relaxation (3.2) 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 $\boldsymbol{\rho}$ that might not be available. Inaccurate estimates of workload can lead to poor performance for such policies; Key (1990) illustrates that static reservation policies in classical loss networks can give very poor performance. Even if we have an estimate of the workload and approximate the solution to (3.2), to handle time-varying workloads, the new solution may require rearranging a large number of VMs and jobs to make their placements match the new solution. This is costly and also causes interruption of many jobs in service.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

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

3.1. Greedy Solution

We describe a greedy algorithm, called the *greedy placement algorithm* (GPA), for solving Optimization (3.1).

GPA takes as input the workload reference vector $\hat{\mathbf{Y}}^L$, and returns an assignment vector $\hat{\mathbf{X}}^L$ that 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 $\mathbf{k}[i] = MaxReward(\mathcal{K}^{\mathcal{J}[i]})$, which is the configuration of highest reward among the configurations that have jobs from the set $\mathcal{J}[i]$, according to Definition 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 $\mathcal{J}[i]$.

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 $\mathcal{J}[i] \neq \varnothing$ **do**
    $\mathbf{k}[i] \leftarrow MaxReward(\mathcal{K}^{\mathcal{J}[i]})$
    $j^\star \leftarrow \text{argmin}_{j:k_j[i]>0} \lceil \frac{r_j}{k_j[i]} \rceil$ ▷ 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 $\hat{X}_{\mathbf{k}[j]}$, $j = 1, \cdots, J$

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 nonzero elements corresponding to $\mathbf{k}[i]$, $i = 1, \ldots, J$.

Remark 1. *The expression MaxReward finds the maximum reward configuration of a subset of job types, which is equivalent with unbounded Knapsack problem (unbounded number of items for each type). This problem is tractable with Pseudopolynomial algorithms to solve it exactly (*Martello and Toth 1990, Andonov et al. 2000*) or fully polynomial approximation algorithms (*Ibarra and Kim 1975*). GPA needs to solve at most J instances of this problem. Note that the number of different instances of the problem is bounded, and we can compute* MaxReward *for all of them offline as they are not workload dependent. This is in contrast to Optimization* (3.1)*, which is equivalent to the multi knapsack problem that is strongly NP-hard (*Kellerer et al. 2004*) and requires resolving when workload reference* $\hat{\mathbf{Y}}$ *changes.*

We next define the limit of $\hat{\mathbf{X}}^L/L$ for input $\hat{\mathbf{Y}}^L = L\boldsymbol{\rho}$, as $L \to \infty$, which we refer to as the *global greedy assignment*. To describe this assignment, we first define a unique ordering of the job types through the following proposition.

Proposition 1. *For any permutation* $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_J)$ *of job types in* $\mathcal{J}$*, let* $\mathcal{J}^\sigma_j := \{\sigma_j, \ldots, \sigma_J\}$*, and* $\mathbf{k}^{(j)} := MaxReward(\mathcal{K}^{\mathcal{J}^\sigma_j})$*. 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. *For any* $j \in \mathcal{J}$, $k^{(j)}_{\sigma_j} > 0$*, and there are constants* $z^{(j)}[\boldsymbol{\rho}] \geq 0$*, such that*

$$\rho_{\sigma_j} = \sum_{\ell=1}^{j} k^{(\ell)}_{\sigma_j} z^{(\ell)}[\boldsymbol{\rho}], \tag{3.3}$$

2. *For any two indexes* $j, j' \in \mathcal{J}$*, with* $j < j'$*, if*

$$\rho_{\sigma_{j'}} = \sum_{\ell=1}^{j} k^{(\ell)}_{\sigma_{j'}} z^{(\ell)}[\boldsymbol{\rho}], \tag{3.4}$$

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

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Proof. See Online Appendix A. □

The global greedy assignment is defined as follows.

Definition 5 (Global Greedy Assignment). Define the index $I_{\boldsymbol{\rho}} \le J$ for which

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

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

$$x^{(g)}_{\mathbf{k}^{(i)}}[\boldsymbol{\rho}] = \begin{cases} z^{(i)}[\boldsymbol{\rho}], & \text{for } i < I_{\boldsymbol{\rho}} \\ 0, & \text{for } i > I_{\boldsymbol{\rho}} \\ 1 - \sum_{j=1}^{i-1} x^{(g)}_{\mathbf{k}^{(j)}}[\boldsymbol{\rho}], & \text{for } i = I_{\boldsymbol{\rho}}, \end{cases} \tag{3.5}$$

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

Because global greedy configurations $\mathbf{k}^{(i)}$, $i = 1, \dots, J+1$ depend on $\boldsymbol{\rho}$, the following configurations will come in handy when the analysis needs to be agnostic to $\boldsymbol{\rho}$.

Definition 6 (Greedy Configurations). The greedy configuration set $\mathcal{K}^{(g)}$ includes all configurations that are output of MaxReward($\mathcal{K}^{\mathcal{J}'}$) for any $\mathcal{J}' \subseteq \mathcal{J}$. That is the set of all possible configurations that may be assigned by GPA and the empty configuration. We define $C^{(g)} := |\mathcal{K}^{(g)}|$. We enumerate configurations of $\mathcal{K}^{(g)}$ as $\bar{\mathbf{k}}^{(i)}$, for $i = 1, \dots, C^{(g)}$, such that $\bar{\mathbf{k}}^{(i_1)} \succ \bar{\mathbf{k}}^{(i_2)}$ if $i_1 < i_2$ (according to Definition 2), and $\bar{\mathbf{k}}^{(C^{(g)})} = \mathbf{0}_J$.

Notice that $\{\mathbf{k}^{(j)}, j = 1, \dots, J+1\} \subseteq \{\bar{\mathbf{k}}^{(i)}, i = 1, \dots, C^{(g)}\}$, and their order is consistent with Definition 2, as defined later.

Definition 7 (Mapping Global Greedy to Greedy). For any $j, j' \in \{1, \dots, J+1\}$, with $j < j'$, there are indexes $g_j, g_{j'} \in \{1, \dots, C^{(g)}\}$, such that $\mathbf{k}^{(j)} \equiv \bar{\mathbf{k}}^{(g_j)}$, $\mathbf{k}^{(j')} \equiv \bar{\mathbf{k}}^{(g_{j'})}$, and $g_j < g_{j'}$. We also define $C^{(g)}_{\boldsymbol{\rho}} := g_{I_{\boldsymbol{\rho}}}$ to be the index for which $\mathbf{k}^{(I_{\boldsymbol{\rho}})} \equiv \bar{\mathbf{k}}^{(C^{(g)}_{\boldsymbol{\rho}})}$.

Example 1. We pause to illustrate the model and definitions thus far through an example. Consider a server with two resources. The size of the first resource is five, and the size of the second one is four, so its resource vector is $\mathbf{R} = (5, 4)$. We consider two job types, one with resource requirement $\mathbf{R}_1 = (2, 1)$ and reward $u_1 = 10$, and the other one with resource requirement $\mathbf{R}_2 = (1, 2)$ and reward $u_2 = 1$. In this case, the feasible configurations given in descending order of their rewards are $\mathcal{K} = \{(2,1), (2,0), (1,1), (1,0), (0,2), (0,1), (0,0)\}$ and $C := |\mathcal{K}| = 7$. Furthermore, $K = 3$ because the maximum number of jobs in any feasible configuration is three. If we consider the subset of VM types $\mathcal{J}' = \{1\}$, then $\mathcal{K}^{\mathcal{J}'} = \{(0,0), (1,0), (2,0)\}$, which by definition has all the configurations of $\mathcal{K}$ with $k_2 = 0$. The greedy configuration set is $\mathcal{K}^{(g)} = \{(2,1), (2,0), (0,2), (0,0)\}$, because by Definition 6:

$$\begin{aligned} &MaxReward(\mathcal{K}^{\{1,2\}}) = (2,1) = \bar{\mathbf{k}}^{(1)} \\ &MaxReward(\mathcal{K}^{\{1\}}) = (2,0) = \bar{\mathbf{k}}^{(2)} \\ &MaxReward(\mathcal{K}^{\{2\}}) = (0,2) = \bar{\mathbf{k}}^{(3)} \\ &MaxReward(\mathcal{K}^{\{\}}) = (0,0) = \bar{\mathbf{k}}^{(4)}. \end{aligned}$$

We can also enumerate the global greedy configurations $\mathbf{k}^{(i)}$ defined in Definition 5. Depending on $\rho_1 \ge 2\rho_2$ or $\rho_1 < 2\rho_2$, we have one of the following cases, respectively:

- Case 1: $\mathbf{k}^{(1)} = \bar{\mathbf{k}}^{(1)} = (2,1)$, $\mathbf{k}^{(2)} = \bar{\mathbf{k}}^{(2)} = (2,0)$, $\mathbf{k}^{(3)} = \bar{\mathbf{k}}^{(4)} = (0,0)$,
- Case 2: $\mathbf{k}^{(1)} = \bar{\mathbf{k}}^{(1)} = (2,1)$, $\mathbf{k}^{(2)} = \bar{\mathbf{k}}^{(3)} = (0,2)$, $\mathbf{k}^{(3)} = \bar{\mathbf{k}}^{(4)} = (0,0)$.

Possible configurations $\mathbf{k}[i]$ returned by GPA are the same as the cases shown above, depending on $\hat{Y}_1 \ge 2\hat{Y}_2$ or $\hat{Y}_1 < 2\hat{Y}_2$, with the difference that they are not defined for $i = 3$ and they depend on GPA's input $\hat{\mathbf{Y}}$, as opposed to $\mathbf{k}^{(i)}$ which depends on $\boldsymbol{\rho}$.

The following proposition states the connection between the GPA and global greedy assignment $x^{(g)}_{\mathbf{k}}[\boldsymbol{\rho}]$.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Proposition 2. *Let* $\hat{X}^L = GPA(L\boldsymbol{\rho})$. *Then*

$$\lim_{L\to\infty} \frac{\hat{X}^L_{\mathbf{k}}}{L} = x^{(g)}_{\mathbf{k}}[\boldsymbol{\rho}], \ \ \forall \mathbf{k} \in \mathcal{K}, \tag{3.6}$$

where $x^{(g)}_{\mathbf{k}}[\boldsymbol{\rho}]$ *is the global greedy assignment of Definition 5.*

Proof. See Online Appendix B. □

Clearly, $\mathbf{x}^{(g)}[\boldsymbol{\rho}]$ is a feasible solution for Optimization (3.2), and it is easy to see that its corresponding objective value is

$$U^{(g)}[\boldsymbol{\rho}] := \sum_{j=1}^{J} u_j \sum_{\ell=1}^{J} k_j^{(\ell)} x^{(g)}_{\mathbf{k}^{(\ell)}}[\boldsymbol{\rho}]. \tag{3.7}$$

It is also easy to see that in Optimization (3.2), we can replace the inequality in (3.2c) with equality, and the optimal value will not change. Let $\mathbf{x}^\star[\boldsymbol{\rho}]$ be one such optimal solution to Optimization (3.2) for workload $\boldsymbol{\rho}$. Then the optimal objective value is

$$U^\star[\boldsymbol{\rho}] := \sum_{j\in\mathcal{J}} u_j \sum_{\mathbf{k}\in\mathcal{K}} k_j x^\star_{\mathbf{k}}[\boldsymbol{\rho}]. \tag{3.8}$$

The following corollary is immediate from Proposition 2.

Corollary 1. *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*

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

This theorem bounds the previous ratio.

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

Proof. Consider the permutation of job types according to Proposition 1. By the global greedy definition and the feasibility of $\mathbf{x}^\star[\boldsymbol{\rho}]$, for any job type σ_j, $j = 1, \ldots, I_{\boldsymbol{\rho}} - 1$, we have

$$\sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} k_{\sigma_j} \leq \sum_{\ell=1}^{j} x^{(g)}_{\mathbf{k}^{(\ell)}} k^{(\ell)}_{\sigma_j} = \rho_{\sigma_j}, \tag{3.9}$$

from which it follows that

$$\sum_{j=1}^{I_{\boldsymbol{\rho}}-1} \sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} k_{\sigma_j} u_{\sigma_j} \leq \sum_{j=1}^{I_{\boldsymbol{\rho}}-1} \sum_{\ell=1}^{j} x^{(g)}_{\mathbf{k}^{(\ell)}} k^{(\ell)}_{\sigma_j} u_{\sigma_j} = \sum_{j=1}^{I_{\boldsymbol{\rho}}-1} \rho_{\sigma_j} u_{\sigma_j} \leq U^{(g)}[\boldsymbol{\rho}]. \tag{3.10}$$

Also for the job types σ_j, for $j = I_{\boldsymbol{\rho}}, \ldots, J$, we have

$$\begin{aligned} &\sum_{j=I_{\boldsymbol{\rho}}}^{J} \sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} k_{\sigma_j} u_{\sigma_j} = \sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} \sum_{j=I_{\boldsymbol{\rho}}}^{J} k_{\sigma_j} u_{\sigma_j} \overset{(a)}{\leq} \\ &\arg\max_{\mathbf{k}\in\mathcal{K}} \sum_{j=I_{\boldsymbol{\rho}}}^{J} k_{\sigma_j} u_{\sigma_j} \overset{(b)}{=} \sum_{j=I_{\boldsymbol{\rho}}}^{J} k^{(I_{\boldsymbol{\rho}})}_{\sigma_j} u_{\sigma_j} \overset{(c)}{\leq} U^{(g)}[\boldsymbol{\rho}]. \end{aligned} \tag{3.11}$$

where (a) is because of the fact that $\sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} = 1$, (b) is by the definition of $\mathbf{k}^{(I_{\boldsymbol{\rho}})}$, and (c) is because $U^{(g)}[\boldsymbol{\rho}]$ is a convex combination of rewards of $\mathbf{k}^{(1)}, \ldots, \mathbf{k}^{(I_{\boldsymbol{\rho}})}$, which all have a reward no less than that of $\mathbf{k}^{(I_{\boldsymbol{\rho}})}$. Then adding (3.10) and (3.11), we get

$$U^\star[\boldsymbol{\rho}] = \sum_{j=1}^{J} \sum_{\mathbf{k}\in\mathcal{K}} x^\star_{\mathbf{k}} k_{\sigma_j} u_{\sigma_j} \leq 2U^{(g)}[\boldsymbol{\rho}]. \ \ \square$$

Theorem 1 can be improved when job types and rewards satisfy a monotone greedy property described next.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Definition 8. We say the job types and the rewards have monotone greedy property if for any two instances of Optimization (3.2) with $\boldsymbol{\rho}_1 \geq \boldsymbol{\rho}_2$, $U^{(g)}[\boldsymbol{\rho}_1] \geq U^{(g)}[\boldsymbol{\rho}_2]$.

It is easy to verify that any system with two job types always has the property in Definition 8. However, in general, the property depends on the profile of jobs types and their rewards and might not hold for adversarial profiles. The next theorem describes the improved bound when the monotone greedy property holds.

Theorem 2. *If job types and rewards satisfy the monotone greedy property, then, for any* $\boldsymbol{\rho}$, $\frac{U^{(g)}[\boldsymbol{\rho}]}{U^{\star}[\boldsymbol{\rho}]} \geq 1 - 1/e$.

Proof. Define a workload $\boldsymbol{\rho}^{\star} := \sum_{\mathbf{k}\in\mathcal{K}} \mathbf{k} x^{\star}_{\mathbf{k}}[\boldsymbol{\rho}]$. We notice that $U^{\star}[\boldsymbol{\rho}] = U^{\star}[\boldsymbol{\rho}^{\star}]$ in LP (3.2). Also by the monotone greedy property, $U^{(g)}[\boldsymbol{\rho}] \geq U^{(g)}[\boldsymbol{\rho}^{\star}]$, because $\boldsymbol{\rho} \geq \boldsymbol{\rho}^{\star}$. Hence, it suffices to prove the theorem for instances where $\boldsymbol{\rho} = \boldsymbol{\rho}^{\star}$ or in other words, instances for which, in the optimal solution, workload fits exactly in servers.

Consider now the projection of the workload $\boldsymbol{\rho}^{\star} = \boldsymbol{\rho}$ onto the global greedy configuration space $\{\mathbf{k}^{(i)}[\boldsymbol{\rho}], i = 1, \ldots, J\}$. Because these configurations are independent, we can write

$$\boldsymbol{\rho}^{\star} = \boldsymbol{\rho} = \sum_{i=1}^{J} z^{(i)}[\boldsymbol{\rho}]\mathbf{k}^{(i)}, \tag{3.12}$$

for $z^{(i)}[\boldsymbol{\rho}]$ introduced in Proposition 1. For notational compactness, define $q_i = z^{(i)}[\boldsymbol{\rho}]$, $i = 1, \ldots, J$, and $p_i = x^{(g)}_{\mathbf{k}^{(i)}}[\boldsymbol{\rho}]$, $i = 1, \ldots, I_{\boldsymbol{\rho}}$, and let $W^{(i)} := U(\mathbf{k}^{(i)}) = \sum_{j=i}^{J} u_{\sigma_j} k^{(i)}_{\sigma_j}$.

Then,

$$\begin{aligned} \sum_{j=i}^{J} q_j W^{(j)} &= \sum_{j=i}^{J} u_{\sigma_j} \sum_{\ell=i}^{j} q_\ell k^{(\ell)}_{\sigma_j} \overset{(a)}{\leq} \sum_{j=i}^{J} \rho_{\sigma_j} u_{\sigma_j} \\ &\overset{(b)}{\leq} \sum_{j=i}^{J} k^{(i)}_{\sigma_j} u_{\sigma_j} = W^{(i)}. \end{aligned} \tag{3.13}$$

Inequality (a) is because $\sum_{\ell=i}^{j} q_\ell k^{(\ell)}_{\sigma_j} \leq \rho_{\sigma_j}$, and Inequality (b) is because we assumed there is an assignment that can completely accommodate workload $\boldsymbol{\rho}$, and hence, ρ_{σ_j} for $j = i, \ldots, J$. If we remove all jobs with types $1, \ldots, i-1$ from assignment $\mathbf{x}^{\star}$, the configurations used in the resulting assignment belong to the subset $\mathcal{K}^{\{\sigma_i, \ldots, \sigma_J\}}$ and $\mathbf{k}^{(i)}$ is the configuration with the highest reward from this set.

An equivalent representation of (3.13) is that, for some constants b_i, $0 \leq b_i \leq 1$, $i = 1, \ldots J$,

$$b_i W^{(i)} = \sum_{j=i}^{J} q_j W^{(j)}, \text{ and } (b_i - q_i) W^{(i)} = b_{i+1} W^{(i+1)}. \tag{3.14}$$

For completeness, we also define $b_{J+1} = 1$. Based on this representation, and using (3.12) and $\boldsymbol{\rho} = \boldsymbol{\rho}^{\star}$ by assumption, we get

$$\begin{aligned} \frac{U^{(g)}[\boldsymbol{\rho}]}{U^{\star}[\boldsymbol{\rho}]} = \frac{\sum_{i=1}^{I_{\boldsymbol{\rho}}} p_i W^{(i)}}{\sum_{i=1}^{J} q_i W^{(i)}} = \frac{\sum_{i=1}^{I_{\boldsymbol{\rho}}} p_i \prod_{j=1}^{i-1} \frac{b_j - q_j}{b_{j+1}} W^{(1)}}{b_1 W^{(1)}} = \\ \sum_{i=1}^{I_{\boldsymbol{\rho}}} \frac{p_i}{b_i} \prod_{j=1}^{i-1} \frac{b_j - q_j}{b_j} = 1 - \prod_{i=1}^{I_{\boldsymbol{\rho}}} \left(1 - \frac{p_i}{b_i}\right). \end{aligned} \tag{3.15}$$

The right-hand side is minimized if $b_i = 1$, $i = 1, \ldots, I_{\boldsymbol{\rho}}$, because $p_i \geq 0$. Then given $\sum_{i=1}^{I_{\boldsymbol{\rho}}} p_i = 1$, the expression is minimized for $p_i = 1/I_{\boldsymbol{\rho}}$, $i = 1, \ldots, I_{\boldsymbol{\rho}}$, and its minimum value is $1 - \left(1 - \frac{1}{I_{\boldsymbol{\rho}}}\right)^{I_{\boldsymbol{\rho}}} > 1 - e^{-1}$. □

Proposition 3. *The worst-case ratio of* $U^{(g)}[\boldsymbol{\rho}]/U^{\star}[\boldsymbol{\rho}]$ *is not greater than* $1 - 1/e$.

Proof. We construct an adversarial example that achieves this bound. See Online Appendix C. □

Hence, the global greedy assignment achieves a factor within 1/2 to $1 - 1/e$ of the optimal normalized reward in "all" cases. Furthermore, the bound $1 - 1/e$ is tight when monotone greedy property holds. The assignment might actually achieve $1 - 1/e$ in all the cases but requires a more careful analysis. In view of Corollary 1, *GPA*($\boldsymbol{\rho}$L) asymptotically achieves the same factor of the optimal reward. In simulations in Section 7, based on real cloud VM instances, we were not able to find any scenario where the ratio is below $1 - 1/e$, and in fact the ratio is much better (≈ 0.97 on average).

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

However, $GPA(\boldsymbol{\rho}\mathrm{L})$ 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.

4. Dynamic Reservation Algorithm

We present a dynamic reservation algorithm (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 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 $o(L)$. For example, $g(L) = \log^{1+b}(L)$, for any constant $b > 0$ works.[2]

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

$$\hat{\mathbf{Y}}^L(t) = \mathbf{Y}^L(t) + g(L)\mathbf{1}, \tag{4.1}$$

where $\mathbf{Y}^L(t)$ in the vector of jobs in the system, *after* any job admission or job departure at time t. The parameter $g(L)$ 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, whereas 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.

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

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 $\hat{\mathbf{X}}^L(t) = GPA(\hat{\mathbf{Y}}^L(t))$. To do so, it assigns *ranks* to servers in different configurations, which range from 1 to $J + 1$. Ranks will later help us define which servers are in Accept Group and which ones are not. 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 $\ell_e = \text{ø}$.

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) $\ell_e = \text{ø}$. In either case, all servers of configuration $\mathbf{k}[i]$ get rank i.
- If $X^L_{\mathbf{k}[i]}(t) \geq \hat{X}^L_{\mathbf{k}[i]}$, 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 $I^*(t)$ to denote the first i for which $X^L_{\mathbf{k}[i]}$ cannot be matched to $\hat{X}^L_{\mathbf{k}[i]}$, that is, the first i at which $\ell_e = \text{ø}$. If all configurations are matched, then $I^*(t) = J$. At the end of CRA, servers with rank greater than $I^*(t)$ and index 1 in any configuration are classified as Reject Group, whereas 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 $\mathtt{AG}_j$ to denote one of the servers of Accept Group with empty type j slot. If no such server exists $\mathtt{AG}_j = \text{ø}$.

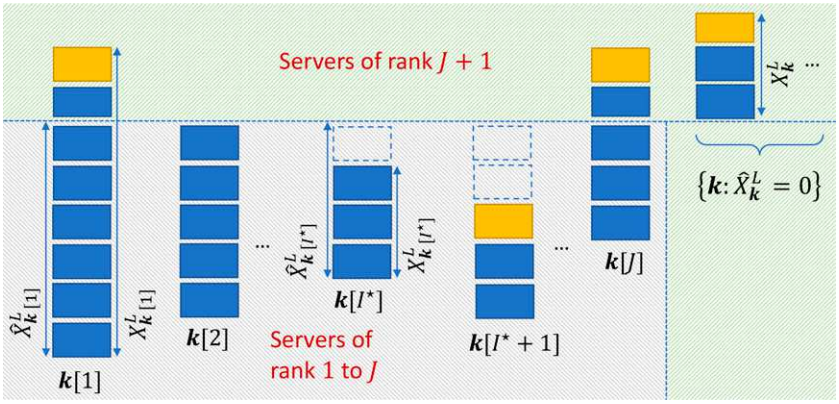
Migrating Job After Departure

Let $\mathtt{RG}_j$ denote the highest rank server among the Reject Group servers with type j jobs. If no such server exists, $\mathtt{RG}_j = \text{ø}$.

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

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 1. State at the End of CRA



Notes. 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 belong to Reject Group.

Initialization

Initially servers have no indexes or classification (and might not even have configurations), so we need to specify how the system state is initialized (say at time 0) under DRA. If servers do not have configurations, but have jobs in them, we initialize $\mathbf{k}^\ell(0) = \hat{\mathbf{k}}^\ell(0)$, that is, 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.

Example 2. Consider the servers and jobs in Example 1. According to the initialization algorithm above, if a server has one job of each type and no explicit configuration assigned, then the initial configuration of the server will be (1, 1). Because (1, 1) is not in the greedy configuration set, it will never be one of configurations $\mathbf{k}[i]$ returned by GPA, and hence its server will always have rank $J+1=3$ until it empties and gets reassigned.

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** ▷ If empty rank $J+1$ server exists.
          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. ▷ Server in configuration $\mathbf{k}[i]$ with rank c.

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

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Remark 2. *Notice the duality of actions performed on arrivals and departures for any job type: jobs are admitted/migrated to empty slots in servers of Accept Group and depart/migrate from filled slots in servers of Reject Group. The number of servers in Reject Group under our algorithm is at most one per configuration, that is, at most $C^{(g)}$ servers (constant independent of L).*[3] *Furthermore, job admissions and migrations are performed to slots that are already deployed in advance. The reservation factor $g(L)$ is critical for maintaining enough deployed slots in the maximum reward configurations for future demand.*

In contrast, a naive static reservation algorithm that solves (3.1) *by replacing $\hat{\mathbf{Y}}$ with an estimate of workload might require changing the configuration of a constant fraction of servers (the equivalent of Reject Group), as workload estimate changes. This would result in preemptions (or migrations) in $O(L)$ interrupted servers.*

Last, more accurate estimates of workload, if available, can be simply used in the input $\hat{\mathbf{Y}}$ to CRA, and CRA itself can be executed less regularly, depending on the complexity and convergence time tradeoff.

The following theorem states the main result regarding DRA.

Theorem 3. *Let $F^{DRA}(L)$ be the expected reward under DRA and $F^{\text{opt}}(L)$ be the optimal expected reward in Optimization* (2.5). *Then*

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

Furthermore, under the monotone greedy property (Definition 8),

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

Remark 3. *We did not make any assumption on the value of $\boldsymbol{\rho}$, and Theorem 3 holds for any $\boldsymbol{\rho}$. Define the system's (normalized) capacity region as*

$$\Lambda = \left\{ \mathbf{y} : \mathbf{y} \leq \sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}}\mathbf{k}, \text{ for } x_{\mathbf{k}} \geq 0, \mathbf{k} \in \mathcal{K}, \sum_{\mathbf{k}\in\mathcal{K}} x_{\mathbf{k}} = 1 \right\}. \tag{4.2}$$

Theorem 3 holds even if $\boldsymbol{\rho}$ is outside Λ. In this scenario, a nonzero fraction of traffic must be rejected even by the optimal policy.

Proof Outline of Theorem 3. The proof of Theorem 3 is based on analysis of fluid limits and a suitable Lyapunov function to show convergence, as we do in Sections 5 and 6.

In Section 5, we describe the fluid limits. The job admission and configuration assignment under DRA crucially depends on an *effective slot deficit process* $\mathbf{q}$ (Definition 11), which, for any job type, effectively measures the deficit in the number of empty slots in the system compared with the required reservation of $g(L)$ slots. A major difficulty in describing the fluid limits is that the $\mathbf{q}$ process evolves at a much faster time-scale compared with the fluid-limit processes of the jobs and of the servers in different configurations in the system. For this reason, we divide the interval $[t, t+\epsilon)$ into smaller intervals of length $\omega(\log L/L)$, and describe the behavior of fluid limits over these small time intervals (Proposition 5).

In Section 6, we use a Lyapunov analysis to show that the fluid limits under DRA "effectively" converge to the global greedy assignment $x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}]$ (Definition 5). We design an LP-based Lyapunov function V, which is zero at states that are effectively $x_{\mathbf{k}}^{(g)}[\boldsymbol{\rho}]$ and positive at any other state (Proposition 6). We then show that at any state that V is positive, its derivative $dV/dt < 0$ (Proposition 7). To do this, we will use the local properties of fluid limits from Section 5 to describe the change in V over the small intervals of length $\omega(\log L/L)$ (Lemma 2). We then infer dV/dt by averaging the change of V over these small intervals in $[t, t+\epsilon)$, as $L \to \infty$, and then letting $\epsilon \to 0$ (Theorem 4). This along with Theorems 1 and 2 will complete the proof.

5. Fluid Limits Under DRA

Before describing the fluid limits, we first define the variables and notations that will be used in our convergence analysis in Sections 5.1 and 5.2.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

5.1. Effective Number of Assigned Servers

Definition 9. The effective number of servers in configuration $\mathbf{k}$ is defined as

$$X^{L(e)}_{\mathbf{k}}(t) := \min(X^L_{\mathbf{k}}(t), \hat{X}^L_{\mathbf{k}}(t)). \tag{5.1}$$

Note that $X^{L(e)}_{\mathbf{k}}(t) = \hat{X}_{\mathbf{k}}{}^L(t) = 0$ if $\mathbf{k} \notin \{\bar{\mathbf{k}}^{(i)}, i = 1, \dots, C^{(g)}\}$. With a minor abuse of terminology, we say the servers in configuration $\mathbf{k}$ with indexes from $X^L_{\mathbf{k}}(t) - X^{L(e)}_{\mathbf{k}}(t) + 1$ to $X^L_{\mathbf{k}}(t)$, have effective configuration $\mathbf{k}$.

Remark 4. *The value of $X^{L(e)}_{\mathbf{k}}(t)$ is independent of the indexing of servers in configuration $\mathbf{k}$. Also note if $\mathbf{k} = \mathbf{k}[j]$, where $\mathbf{k}[j]$, $j \le J$, is the jth configuration returned by GPA at time t, then in DRA, servers with effective configuration $\mathbf{k}[j]$ get rank j, and servers without effective configuration have rank $J + 1$.*

Definition 10. Given an integer $i \le C^{(g)}$, Reject Group servers can be divided as $\mathtt{RG} = \overline{\mathtt{RG}}(i) \bigcup \mathtt{RG}(i)$. The servers with index 1 without effective configuration in $\bar{\mathbf{k}}^{(\ell)}$, for $\ell = 1, \dots, i$, belong to $\overline{\mathtt{RG}}(i)$, whereas the rest of servers of Reject Group belong to $\mathtt{RG}(i)$.

5.2. Effective Slot Deficit: *q* Process

The job admission and configuration assignment under DRA crucially depends on the $\mathbf{q}$ process defined here.

Definition 11. For $i \in \{1, \cdots, C^{(g)}\}$, and $j \in \mathcal{J}$, we define

$$q^L_{\bar{\mathbf{k}}^{(i)},j}(t) := \sum_{\ell=1}^{i} X^{L(e)}_{\bar{\mathbf{k}}^{(\ell)}}(t)\bar{k}^{(\ell)}_j - Y^L_j(t) - g(L). \tag{5.2}$$

Note that, $\forall j \in \mathcal{J}$, $q^L_{\bar{\mathbf{k}}^{(i_2)},j}(t) \ge q^L_{\bar{\mathbf{k}}^{(i_1)},j}(t)$ if $i_2 \ge i_1$.

In words, $q^L_{\bar{\mathbf{k}}^{(i)},j}(t)$ measures the difference between the total number of type j slots (filled or empty) in servers that have effective configurations in the set $\{\bar{\mathbf{k}}^{(\ell)} : \ell \le i\}$ (see Definition 9) and the number of type j jobs in the system $Y^L_j(t)$ and $g(L)$ type j reservation slots.

DRA (specifically GPA) will stop assigning configurations that have type j slots once $Y^L_j(t) + g(L)$ slots can be accommodated in servers with effective configuration in $\{\bar{\mathbf{k}}^{(\ell)}, \ell \le i\}$. Because slots are created per server basis, by assigning configurations that each has at most K slots, we have $q^L_{\bar{\mathbf{k}}^{(i)},j}(t) < K$.

To gain more insight, when $q^L_{\bar{\mathbf{k}}^{(i)},j}(t) \ge 0$ for an $i \in \{1, \cdots, C^{(g)}\}$, it means type j jobs have enough reservation. When it is negative, it indicates the deficit of slots in servers with effective configuration $\{\bar{\mathbf{k}}^{(\ell)}, \ell \le i\}$. When $q^L_{\bar{\mathbf{k}}^{(i)},j}(t) > -g(L) + JK$, for an $i \in \{1, \cdots, C^{(g)}\}$, a type j arrival at time t will certainly find a valid empty slot ($\mathtt{AG}_j \ne \varnothing$) and will be admitted. This is because the number of empty slots of type j in Reject Group servers with any effective configuration is less than JK.

The $\mathbf{q}$ process also determines the configuration assigned by CRA to an empty server ℓ_e chosen for reassignment. The configuration would be $\bar{\mathbf{k}}^{(i)}$, $i < C^{(g)}$, if

$$\max_{j:\bar{k}^{(\ell)}_j > 0} q^L_{\bar{\mathbf{k}}^{(\ell)},j}(t) \ge 0, \quad \forall \ell \le i-1, \tag{5.3a}$$

$$\max_{j:\bar{k}^{(i)}_j > 0} q^L_{\bar{\mathbf{k}}^{(i)},j}(t) < 0. \tag{5.3b}$$

This also implies that if only (5.3b) holds, the server would be assigned to one of the configurations $\bar{\mathbf{k}}^{(\ell)}$, $\ell = 1, \dots, i$.

5.3. Existence of Fluid Limits

We define the scaled (normalized with L) processes $\mathbf{x}^{L(e)}(t)$, $\mathbf{y}^L(t)$, as follows. For $i \in \{1, \dots, C^{(g)}\}$, and $j \in \mathcal{J}$,

$$x^{L(e)}_{\bar{\mathbf{k}}^{(i)}}(t) = \frac{1}{L} X^{L(e)}_{\bar{\mathbf{k}}^{(i)}}(t), \quad y^L_j = \frac{1}{L} Y^L_j(t),$$

and define $z^L(t) := (\mathbf{x}^{L(e)}(t), \mathbf{y}^L(t))$. We also define the space

$$\mathcal{Z} = \left\{(\mathbf{x}^{(e)}, \mathbf{y}) : \mathbf{y} \in \Lambda, x^{(e)}_{\bar{\mathbf{k}}^{(i)}} \ge 0, \sum_{i=1}^{C^{(g)}} x^{(e)}_{\bar{\mathbf{k}}^{(i)}} \le 1, \sum_{i=1}^{C^{(g)}} x^{(e)}_{\bar{\mathbf{k}}^{(i)}} \bar{\mathbf{k}}^{(i)} \le \mathbf{y}\right\},$$

where Λ was defined in (4.2).

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Proposition 4. *Consider a sequence of systems with increasing L, and initializations $\mathbf{z}^L(0) = (\mathbf{x}^{L(e)}(0), \mathbf{y}^L(0)) \in \mathcal{Z}$, as $L \to \infty$. Then there is a subsequence of L such that $\mathbf{x}^{L(e)}(t) \to \mathbf{x}^{(e)}(t)$, $\mathbf{y}^L(t) \to \mathbf{y}(t)$, along the subsequence. Any limit $\mathbf{z}(t) := (\mathbf{x}^{(e)}(t), \mathbf{y}(t))$, $t \geq 0$, is called a fluid limit sample path. The convergence is almost surely u.o.c. (uniformly over compact time intervals), and the fluid limit sample paths are Lipschitz continuous.*

Proof. Proof is standard and can be found in Online Appendix D. □

5.4. Description of Fluid Limits

We provide an informal description of fluid limit equations here. The formal definitions and proofs can be found in Online Appendix G.

The properties of the fluid limit processes crucially depend on the $\mathbf{q}$ process (Definition 11). First, from (5.2) and because $q^L_{\bar{\mathbf{k}}^{(i)},j}(t) < K$, it follows that

$$\sum_{\ell=1}^{C^{(g)}} \bar{k}_j^{(\ell)} x^{(e)}_{\bar{\mathbf{k}}^{(\ell)}}(t) \leq y_j(t), \ \ \forall j \in \mathcal{J}. \tag{5.4}$$

Let $x_{\emptyset}(t)$ be the fraction of servers that are empty and of rank $J+1$ at the fluid limit. When $x_{\emptyset}(t) > 0$, then CRA always finds empty rank $J+1$ servers available for reassignment. In this case, every job type will have enough empty slots, and all the arrivals will be admitted; that is, we can find an ϵ sufficiently small such that for every job type j and every time $\tau \in [t, t+\epsilon)$, $q^L_{\bar{\mathbf{k}}^{(C^{(g)})},j}(\tau) \geq 0$. At the fluid limit, type j jobs arrive at rate λ_j and existing type j jobs depart at rate $y_j(t)\mu_j$. Hence, at any regular time t that $x_{\emptyset}(t) > 0$,

$$dy_j(t)/dt = \lambda_j - y_j(t)\mu_j, \ \ \forall j \in \mathcal{J}, \tag{5.5a}$$

$$\sum_{\ell=1}^{C^{(g)}} \bar{k}_j^{(\ell)} x^{(e)}_{\bar{\mathbf{k}}^{(\ell)}}(t) = y_j(t), \tag{5.5b}$$

where (5.5b) is based on (5.2) and because $\lim_{L\to\infty} \frac{1}{L} q^L_{\bar{\mathbf{k}}^{(C^{(g)})},j}(t) = 0$ in this case.

A major difficulty in describing fluid limits happens on the *boundary* $x_{\emptyset}(t) = 0$, that is, when there are not always empty rank $J+1$ servers available for reassignment when CRA runs. In this case, let $i^\star(t)$ be the *largest* index in $\{1, \cdots, C^{(g)} - 1\}$ such that for every $i \leq i^\star(t)$,

$$\sum_{\ell=1}^{i} \bar{k}_{j_i}^{(\ell)} x^{(e)}_{\bar{\mathbf{k}}^{(\ell)}}(t) = y_{j_i}(t), \text{ for some } j_i \in \mathcal{J}, \tag{5.6}$$

with the convention that $i^\star(t) = 0$ if (5.6) does not hold for $i = 1$. If $i^\star(t) < C^{(g)} - 1$, then for L sufficiently large, and every time $\tau \in [t, t+\epsilon)$ for ϵ sufficiently small,

$$\max_{j:\bar{k}_j^{(i^\star(t)+1)}>0} q^L_{\bar{\mathbf{k}}^{(i^\star(t)+1)},j}(\tau) < 0. \tag{5.7}$$

Based on Definition 10, servers in $\bar{\mathtt{RG}}(i^\star(t)+1)$ have higher ranks compared with those in $\mathtt{RG}(i^\star(t)+1)$, so any migrations by DRA will take place from $\overline{\mathtt{RG}}(i^\star(t)+1)$ first. We can then show that servers of $\overline{\mathtt{RG}}(i^\star(t)+1)$ empty at the fluid scale, at a rate of at least

$$\frac{\mu_{min}}{JKC^2}\left(1 - \sum_{\ell=1}^{i^\star(t)+1} x^{(e)}_{\bar{\mathbf{k}}^{(\ell)}}(t)\right), \tag{5.8}$$

where $\mu_{min} := \min_{j\in\mathcal{J}} \mu_j$ (see Lemma 5 in Online Appendix F).

The algorithm will reassign any such server that empties to one of configurations $\bar{\mathbf{k}}^{(\ell)}$ for $\ell = 1, \ldots, i^\star(t)+1$. If instead $i^\star(t) = C^{(g)} - 1$, then it is uncertain whether servers that empty need to be reassigned to a new configuration or not, depending on whether $\max_{j\in\mathcal{J}:\bar{k}_j^{(i)}>0} q^L_{\bar{\mathbf{k}}^{(i)},j}(\tau) < 0$, for some $i < C^{(g)}$ at time $\tau \in [t, t+\epsilon)$.

Hence, what we see is that, if $x_{\emptyset}(t) = 0$, when a server gets empty, it can be assigned to one of the configurations $\bar{\mathbf{k}}^{(i)}$, $i = 1, \ldots, i^\star(t)+1$. Exact characterization of these assignment rates, however, is not easy as they depend on values of processes $q^L_{\bar{\mathbf{k}}^{(i)},j}(\tau)$, $i \in \{1, \ldots, i^\star(t)\}$, $j \in \mathcal{J}$, which evolve at a much faster time-scale than the scaled processes $\mathbf{x}^{L(e)}$ and $\mathbf{y}^L$. By the continuity of the fluid limit sample paths, at any regular time t, we can choose ϵ small enough such that for all $\tau \in [t, t+\epsilon)$, $\mathbf{y}(\tau)$, and $\mathbf{x}^{(e)}(\tau)$ are approximately constant and equal to $\mathbf{y}(t)$ and $\mathbf{x}^{(e)}(t)$, respectively (their actual change being of order ϵ). However, over the same interval, the $\mathbf{q}^L$ process makes $O(L)$

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

transitions, and its elements can change in the range $[-LK, K]$. This phenomenon is known as *separation of time scales* and has been also observed in other systems (Hunt and Kurtz 1994, Hunt et al. 1997).

To further analyze fluid limits in our setting, we divide the interval $[t, t+\epsilon)$ into smaller intervals of length $\omega(\log L/L)$ and infer properties for the fluid limits over $[t, t+\epsilon)$ based on averaging the behavior of scaled processes over these smaller intervals, as $L \to \infty$, and then we let $\epsilon \to 0$. To this end, we first make a few definitions.

Because the rate of change of any of the processes $x^{L(e)}_{\vec{\mathbf{k}}^{(i)}}(\tau)$ and $y^L_j(\tau)$ over a subinterval is of interest, we give it a special name.

Definition 12 (Local Derivatives). Given an interval $[\tau_a, \tau_b)$, we define the "local derivatives" of the scaled processes as

$$\nabla x^{L(e)}_{\vec{\mathbf{k}}^{(i)}}[\tau_a, \tau_b) := \frac{x^{L(e)}_{\vec{\mathbf{k}}^{(i)}}(\tau_b) - x^{L(e)}_{\vec{\mathbf{k}}^{(i)}}(\tau_a)}{\tau_b - \tau_a} \quad i = 1, \dots, C^{(g)}, \tag{5.9}$$

$$\nabla y^L_j[\tau_a, \tau_b) := \frac{y^L_j(\tau_b) - y^L_j(\tau_a)}{\tau_b - \tau_a} \quad j \in \mathcal{J}. \tag{5.10}$$

Definition 13. For any $i \le C^{(g)}_{\boldsymbol{\rho}} - 1$, we define a set

$$\mathcal{J}^{(i)} := \{j \in \mathcal{J} : \vec{k}^{(i)}_j > 0, \sum_{\ell=1}^{i} \vec{k}^{(\ell)}_j x^{(g)}_{\vec{\mathbf{k}}^{(\ell)}} = \rho_j\}. \tag{5.11}$$

Definition 14. For given positive constants α_i, $i = 1, \dots, C^{(g)}_{\boldsymbol{\rho}} - 1$, we define $C_\alpha(t)$ to be the largest index at time t such that $C_\alpha(t) \le \min(i^*(t), C^{(g)}_{\boldsymbol{\rho}} - 1)$ and

$$\forall i \in [1, \dots, C_\alpha(t)] : x^{(g)}_{\vec{\mathbf{k}}^{(i)}} - x^{(e)}_{\vec{\mathbf{k}}^{(i)}}(t) < \alpha_i. \tag{5.12}$$

5.4.1. Subinterval Construction.

We first define a function $f(L)$, which will control the length of subintervals.

Definition 15. The function $f(L)$ is defined as

$$f(L) := \frac{\sqrt{g(L)\log(L)}}{L}, \tag{5.13}$$

where $g(L)$ is the reservation factor as defined in DRA.

We divide $[t, t+\epsilon)$ into smaller intervals $[\tau_n, \tau_{n+1})$, such that

$$\tau_0 = t, \ \tau_n = \tau_{n-1} + D_{L,\epsilon}, \ n = 1, \dots, N_L, \tag{5.14}$$

where $N_L = \lceil 1/f(L) \rceil$ is the number of such smaller intervals, and $D_{L,\epsilon} = \frac{\epsilon}{N_L}$ is the length of each one. We then further divide each $[\tau_n, \tau_{n+1})$ into a *bounded* number M_n of subintervals $[\tau^{(m-1)}_n, \tau^{(m)}_n)$, $m = 1, \dots, M_n$, $\tau^{(0)}_n = \tau_n$, $\tau^{(M_n)}_n = \tau_{n+1}$. For every n, the sequence of stopping times $\tau^{(m)}_n$ is recursively generated as follows:

Each time $\tau^{(m)}_n$ is associated with a *driving* set of job indexes $\bar{\mathcal{J}}[m]$, with the initialization $\bar{\mathcal{J}}[0] = \emptyset$ and $\tau^{(0)}_n = \tau_n$. Suppose $\bar{\mathcal{J}}[m-1] := \{j_i : i = 1, \dots, G_{m-1}\}$ at time $\tau^{(m-1)}_n$, where $j_i \in \mathcal{J}^{(i)}$ (Definition 13). Define $h^{\bar{\mathcal{J}}[m-1],(\ell)}(t)$, $\ell = 1, \dots, G_{m-1}$, to be the (unique) solution to the following system of equations

$$\sum_{\ell=1}^{i} \vec{k}^{(\ell)}_{j_i} h^{\bar{\mathcal{J}}[m-1],(\ell)}(t) = \lambda_{j_i} - \mu_{j_i} y_{j_i}(t), \quad i = 1, \dots, G_{m-1}. \tag{5.15}$$

The next $\tau^{(m)}_n$ is the *earliest* time $\tau \in [\tau^{(m-1)}_n, \tau_{n+1})$ such that $q^L_{\vec{\mathbf{k}}^{(G_m)},j}(\tau) \ge 0$ for some $G_m \le \min(G_{m-1}+1, C_\alpha(t))$ and some $j \in \mathcal{J}^{(G_m)}$. Furthermore, if $G_m \le G_{m-1}$, we additionally require that

$$\sum_{\ell=1}^{G_m} \vec{k}^{(\ell)}_j h^{\bar{\mathcal{J}}[m-1],(\ell)}(t) > \lambda_j - \mu_j y_j(t). \tag{5.16}$$

At such a time τ, we set $\tau^{(m)}_n = \tau$, and the driving index set is set to

$$\bar{\mathcal{J}}[m] := \{j'_i : i = 1, \dots, G_m\}, \tag{5.17}$$

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

where $j'_i = j_i$ for $i = 1, \ldots, G_m - 1$, and $j'_{G_m} = j$. Also, $h^{\bar{\mathcal{J}}[m],(\ell)}(t)$, $\ell = 1, \ldots, G_m$ is set to the solution of the system of Equations (5.15) for the set $\bar{\mathcal{J}}[m]$. If no time $\tau \in [\tau_n^{(m-1)}, \tau_{n+1})$ satisfies the given conditions, then $m = M_n$ and $\tau_n^{(M_n)} = \tau_{n+1}$.

The importance of quantities $h^{\bar{\mathcal{J}}[m],(i)}(t)$, $i = 1, \ldots, G_m$, will become evident later where we will show (see Lemma 7 in the online appendix) that

$$\nabla x^{L(e)}_{\bar{\mathbf{k}}^{(i)}}[\tau_n^{(m)}, \tau_n^{(m+1)}) = h^{\bar{\mathcal{J}},(i)}(t) + \frac{o(f(L))}{\tau_n^{(m+1)} - \tau_n^{(m)}}. \tag{5.18}$$

Hence, roughly, (5.15) gives the values of local derivatives, whereas when (5.16) occurs, the values of local derivatives change.

The number of stopping times M_n in any interval $[\tau_n, \tau_{n+1})$ is bounded. This is because the number of different driving sets $\bar{\mathcal{J}}[m]$ is finite and no set may appear twice in that sequence, because the comparison (5.16) induces a total ordering between the sets. Considering all possible driving set of indexes that may appear in the sequence, we have $M_n \le 1 + \sum_{i=1}^{C_a(t)} \prod_{\ell=1}^{i} |\mathcal{J}^{(i)}| < \infty$.

5.4.2. Properties of Fluid Limits Over Subintervals. Given an $\epsilon_\rho > 0$, we first define the set of fluid limit states

$$\Gamma[\epsilon_\rho] := \{(\mathbf{x}^{(e)}, \mathbf{y}) : \mathbf{y} \le \boldsymbol{\rho} + \epsilon_\rho\} \cap \mathcal{Z}. \tag{5.19}$$

The following lemma states the invariant property of $\Gamma[\epsilon_\rho]$.

Lemma 1. *If $(\mathbf{x}^{(e)}(0), \mathbf{y}(0)) \in \mathcal{Z}$, then for any $\epsilon_\rho > 0$, there is a time $T_{\epsilon_\rho} > 0$ such that for all $t \ge T_{\epsilon_\rho}$, $(\mathbf{x}^{(e)}(t), \mathbf{y}(t)) \in \Gamma[\epsilon_\rho]$. Furthermore, convergence is uniform over all initial states in $\mathcal{Z}$.*

Proof. See Online Appendix E. □

The following proposition states the behavior of scaled processes over the subintervals.

Proposition 5. *For every $m \in \{0, \ldots, M_n - 1\}$, let $\bar{\mathcal{J}}[m] = \{j_i : i = 1, \ldots, G_m\}$ be the index set corresponding to time $\tau_n^{(m)}$, and $\ell_m := G_m + 1$. Then we can choose α_i in Definition 14 and ϵ_ρ in (5.19) sufficiently small, such that, for any regular time $t \ge T_{\epsilon_\rho}$, with probability at least $1 - o(L^{-2})$, all the following properties hold:*

Property 1. *For every $i \in \{1, \ldots, \ell_m - 1\}$,*

$$\sum_{\ell=1}^{i} \bar{k}_{j_i}^{(\ell)} \nabla x^{L(e)}_{\bar{\mathbf{k}}^{(\ell)}}[\tau_n^{(m)}, \tau_n^{(m+1)}] = \lambda_{j_i} - \mu_{j_i} y_{j_i}(t) + \frac{o(f(L))}{\tau_n^{(m+1)} - \tau_n^{(m)}}. \tag{5.20}$$

Property 2. *If $\ell_m < C_{\boldsymbol{\rho}}^{(g)}$,*

$$\nabla x^{L(e)}_{\bar{\mathbf{k}}^{(\ell_m)}}[\tau_n^{(m)}, \tau_n^{(m+1)}] > \frac{\mu_{min}}{2} \alpha_{\ell_m} + \frac{o(f(L))}{\tau_n^{(m+1)} - \tau_n^{(m)}}. \tag{5.21}$$

Property 3. *If $\ell_m = C_{\boldsymbol{\rho}}^{(g)}$,*

$$\nabla x^{L(e)}_{\bar{\mathbf{k}}^{(C_{\boldsymbol{\rho}}^{(g)})}}[\tau_n^{(m)}, \tau_n^{(m+1)}] > \frac{o(f(L))}{\tau_n^{(m+1)} - \tau_n^{(m)}} +$$
$$\min\left\{ \frac{\mu_{min}}{JKC^2}\left(1 - \sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}} x^{(e)}_{\bar{\mathbf{k}}^{(i)}}(t)\right) - \sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}-1} \left(\nabla x^{L(e)}_{\bar{\mathbf{k}}^{(i)}}[\tau_n^{(m)}, \tau_n^{(m+1)}]\right)^+, \min_{j: \bar{k}_j^{(C_{\boldsymbol{\rho}}^{(g)})} > 0} \frac{\lambda_j - \mu_j y_j(t) - \sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}-1} \bar{k}_j^{(i)} \nabla x^{L(e)}_{\bar{\mathbf{k}}^{(i)}}[\tau_n^{(m)}, \tau_n^{(m+1)}]}{\bar{k}_j^{(C_{\boldsymbol{\rho}}^{(g)})}} \right\}. \tag{5.22}$$

In words, Property 1 states that, roughly, for any $i < \ell_m$, there is a job type j_i such that each of the effective number of servers with configurations $\{\bar{\mathbf{k}}^{(\ell)}$ for $\ell = 1, \ldots, i\}$ changes at a rate that can accommodate exactly additional type j_i arrivals.

Property 2 states that effective number of servers with configuration $\bar{\mathbf{k}}^{(\ell_m)}$ increases by an amount proportional to α_{ℓ_m}. This implies that the rate at which $x^{(e)}_{\bar{\mathbf{k}}^{(\ell_m)}}(t)$ converges to the global greedy solution is lower bounded by a constant independent of the system state.

Property 3 describes the change in the effective number of servers in $\bar{\mathbf{k}}^{(C_{\boldsymbol{\rho}}^{(g)})}$, which is the last configuration of the global greedy solution. The change either satisfies the same condition as Property 1, or it is bounded by the difference of how fast Reject Group servers empty (based on (5.8) for $i^\star(t) = C_{\boldsymbol{\rho}}^{(g)} - 1$) and at what rate they are assigned to configurations $\bar{\mathbf{k}}^{(i)}$ for $i < C_{\boldsymbol{\rho}}^{(g)}$.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Proof of Proposition 5. The proof, including all supporting lemmas, is provided in Online Appendix G. □

6. Convergence Analysis

We show that the fluid limit of the effective configuration process $\mathbf{x}^{(e)}(t)$ (which is a lower bound on the number of servers in each configuration) converges to the global greedy solution $\mathbf{x}^{(g)}$.

Theorem 4. *Consider the fluid limits of the system under DRA, under any workload* $\boldsymbol{\rho}$*, and any initial state* $\mathbf{z}(0) \in \mathcal{Z}$*. Then*

$$\lim_{t\to\infty} x^{(e)}_{\mathbf{k}}(t) = x^{(g)}_{\mathbf{k}}, \quad \mathbf{k} \in \mathcal{K}^{(g)}. \tag{6.1}$$

Proof. Recall that $\mathbf{z}(t) = (\mathbf{x}^{(e)}(t), \mathbf{y}(t))$. We want to show that $\mathbf{z}(t)$ converges to a point in the set $\Gamma^\star$ defined as

$$\Gamma^\star := \{\mathbf{z} := (\mathbf{x}^{(e)}, \mathbf{y}) \in \Gamma[\epsilon_\rho] : x^{(e)}_{\mathbf{k}} = x^{(g)}_{\mathbf{k}}, \mathbf{k} \in \mathcal{K}^{(g)}\}, \tag{6.2}$$

where $\Gamma[\epsilon_\rho]$ was defined in (5.19).

To show convergence, we use a Lyapunov function of the form

$$V(\mathbf{z}(t)) := \sum_{i=1}^{C^{(g)}_{\boldsymbol{\rho}}} Z_i\left(x^{(g)}_{\vec{\mathbf{k}}^{(i)}} - x^{(e)}_{\vec{\mathbf{k}}^{(i)}}(t)\right) + Z\sum_{j=1}^{J}(y_j(t) - \rho_j)^+, \tag{6.3}$$

where Z and Z_i, $i \in \{1, \ldots, C^{(g)}_{\boldsymbol{\rho}}\}$, are positive constants satisfying

$$Z > 4Z_1,\ Z_i > \xi Z_{i+1},\ i = 1, \ldots, C^{(g)}_{\boldsymbol{\rho}} - 1, \tag{6.4}$$

for a $Z_{C^{(g)}_{\boldsymbol{\rho}}} > 0$, and a sufficiently large constant $\xi > 2K + 1$.

By choosing ϵ_ρ sufficiently small and ξ sufficiently large, we can ensure the conditions of LaSalle's invariance principle (LaSalle 1960, Cohen and Rouhling 2017) hold for any $\mathbf{z} \in \Gamma[\epsilon_\rho]$. The conditions of LaSalle's invariance principle are as follows:

i. For any $\mathbf{z} \in \Gamma[\epsilon_\rho]$, we have $V(\mathbf{z}) \geq 0$ and $V(\mathbf{z}) = 0$ if and only if $\mathbf{z} \in \Gamma^\star$, and

ii. For any $\mathbf{z}(t) \in \Gamma[\epsilon_\rho] \setminus \Gamma^\star$, $\mathrm{d}V(\mathbf{z}(t))/\mathrm{d}t < 0$, almost surely.

These conditions together with Lemma 1 will then imply that the limit points of trajectory $\mathbf{z}(t)$ are in $\Gamma^\star$.

The constant ϵ_ρ has to be chosen sufficiently small so we can apply Proposition 5. Moreover, ξ has to be chosen sufficiently large so we can show the second condition of LaSalle's invariance principle (Proposition 7). We provide the exact bounds on these values in the proofs.

We state each condition of LaSalle's invariance principle as a proposition followed by its proof.

Proposition 6. *Consider* $V(\mathbf{z})$ *in* (6.3)*, with coefficients in* (6.4)*, for any* $\xi > (2K + 1)$*, and* $\epsilon_\rho > 0$*. Then we have* $V(\mathbf{z}) \geq 0$ *for any* $\mathbf{z} \in \Gamma[\epsilon_\rho]$*, and* $V(\mathbf{z}) = 0$ *if and only if* $\mathbf{z} \in \Gamma^\star$*.*

Proof of Proposition 6. Consider the following maximization problem over $\boldsymbol{\eta} \in \mathbb{R}^{C^{(g)}_{\boldsymbol{\rho}}}, \boldsymbol{\theta} \in \mathbb{R}^J$, where η_i corresponds to $x^{(e)}_{\vec{\mathbf{k}}^{(i)}}(t)$ and θ_j corresponds to $(y_j(t) - \rho_j)^+$ in (6.3),

$$\max_{\boldsymbol{\eta},\boldsymbol{\theta}} \quad \sum_{i=1}^{C^{(g)}_{\boldsymbol{\rho}}} Z_i\eta_i - \sum_{j=1}^{J} Z\theta_j \tag{6.5a}$$

$$\text{s.t} \quad \sum_{i=1}^{C^{(g)}_{\boldsymbol{\rho}}} \eta_i \leq 1, \tag{6.5b}$$

$$\sum_{i=1}^{C^{(g)}_{\boldsymbol{\rho}}} \vec{k}^{(i)}_j \eta_i - \theta_j \leq \rho_j, \quad j = 1, \ldots, J \tag{6.5c}$$

$$\theta_j \leq \epsilon_\rho, \quad j = 1, \ldots, J \tag{6.5d}$$

$$\eta_i \geq 0, \quad i = 1, \ldots, C^{(g)}_{\boldsymbol{\rho}} \tag{6.5e}$$

$$\theta_j \geq 0, \quad j = 1, \ldots, J. \tag{6.5f}$$

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

To prove the proposition, it is enough to show that the assignment $(\boldsymbol{\eta}^{(g)}, \boldsymbol{\theta}^{(g)})$ that corresponds to the global greedy solution $\mathbf{x}^{(g)}$ is the unique maximizer of the previous LP. This assignment is

$$\begin{aligned} \eta_i^{(g)} &= x^{(g)}_{\mathbf{k}^{(i)}}, \; i = 1, \ldots, C_{\boldsymbol{\rho}}^{(g)}, \\ \theta_j^{(g)} &= 0, \quad j = 1, \ldots, J. \end{aligned} \tag{6.6}$$

First, (6.6) is a basic feasible solution for LP (6.5); that is, it is a corner point of the LP's polytope, because it is on the boundary of $C_{\boldsymbol{\rho}}^{(g)} + J$ independent inequalities (equal to the number of variables).

To show that (6.6) is the "unique maximizer," we need to verify that every neighboring corner point has lower objective value, and to do this, it suffices to verify that, by moving along any valid direction within the polytope, starting from assignment (6.6), the objective value is reduced. This proves that Point (6.6) is locally optimal, which implies it is also global optimal, because the optimization is LP (and convex) (Boyd and Vandenberghe 2004). In the rest of the proof, we use g_j to be the mapping in Definition 7 for $j = 1, \ldots, I_{\boldsymbol{\rho}}$, and σ_j to be the permutation of indexes $\{1, \ldots, J\}$ as defined in Proposition 1.

We define $\Delta\eta_i := \eta_i' - \eta_i^{(g)}$ for $i \in \{1, \ldots, C_{\boldsymbol{\rho}}^{(g)}\}$, and $\Delta\theta_j := \theta_j'$ for $j \in \{1, \ldots, J\}$, where η_i' and θ_j' are the values of a feasible point. We prove that the change in objective is negative considering only one positive $\Delta\eta_i$ for some $i \in \{1, \ldots, C_{\boldsymbol{\rho}}^{(g)}\} \setminus \{g_j : j = 1, \ldots, I_{\boldsymbol{\rho}}\}$, whereas the other $\Delta\eta_i$ s in this set are zero, and Constraints (6.5b)–(6.5f) are not violated. This suffices because any feasible point can be constructed as a convex summation of the changes $\Delta\eta_i$, and if individual changes reduce objective, their convex sum will reduce the objective too.

Suppose $i^\star \in \{1, \ldots, C_{\boldsymbol{\rho}}^{(g)}\} \setminus \{g_j : j = 1, \ldots, I_{\boldsymbol{\rho}}\}$ is the index for which $\Delta\eta_{i^\star} > 0$. A feasible point will necessarily satisfy the following set of equations, which correspond to $C_{\boldsymbol{\rho}}^{(g)} + J$ constraints (specifically, (6.5b) and (6.5c) for $j \in \{\sigma_{j'} : j' = 1, \ldots, I_{\boldsymbol{\rho}}\}$ and (6.5f) for $j = 1, \ldots, J$), which held as equalities at Point (6.6),

$$\begin{aligned} &-\Delta\theta_j \le 0, \quad j = 1, \ldots, J, \\ &-\Delta\eta_{i^\star} < 0; \; \Delta\eta_i = 0, i \ne i^\star, i \in \{1, \ldots, C_{\boldsymbol{\rho}}^{(g)}\} \setminus \{g_j : j = 1, \ldots, I_{\boldsymbol{\rho}}\}, \\ &\bar{k}^{(i^\star)}_{\sigma_j} \Delta\eta_{i^\star} + \sum_{\ell=1}^{j} \bar{k}^{(g_\ell)}_{\sigma_j} \Delta\eta_{g_\ell} - \Delta\theta_{\sigma_j} \le 0, \quad j = 1, \ldots, I_{\boldsymbol{\rho}} - 1, \\ &\Delta\eta_{i^\star} + \sum_{j=1}^{I_{\boldsymbol{\rho}}} \Delta\eta_{g_j} \le 0. \end{aligned} \tag{6.7}$$

Conditions (6.7) are not necessarily sufficient, so even if all of them are satisfied, the resulting point may be infeasible. Nevertheless, we prove that in any case, the objective function will be reduced. The change in value of objective function is given by

$$\Delta F := \sum_{\ell=1}^{I_{\boldsymbol{\rho}}} Z_{g_\ell} \Delta\eta_{g_\ell} + Z_{i^\star} \Delta\eta_{i^\star} - Z \sum_{j=1}^{J} \Delta\theta_j. \tag{6.8}$$

Given Conditions (6.7), we show (6.8) will be negative by finding constants $\beta > 0$, $\beta_j > 0, j = 1, \ldots, I_{\boldsymbol{\rho}}$, and $\gamma_j > 0, j = 1, \ldots, J$, such that

$$\begin{aligned} \Delta F = \beta(-\Delta\eta_{i^\star}) &+ \sum_{j=1}^{I_{\boldsymbol{\rho}}-1} \beta_j \Big(\bar{k}^{(i^\star)}_{\sigma_j} \Delta\eta_{i^\star} + \sum_{\ell=1}^{j} \bar{k}^{(g_\ell)}_{\sigma_j} \Delta\eta_{g_\ell} - \Delta\theta_{\sigma_j} \Big) \\ &+ \beta_{I_{\boldsymbol{\rho}}} (\Delta\eta_{i^\star} + \sum_{\ell=1}^{I_{\boldsymbol{\rho}}} \Delta\eta_{g_\ell}) + \sum_{j=1}^{J} \gamma_j (-\Delta\theta_j). \end{aligned} \tag{6.9}$$

It is not difficult to show by matching the coefficients of (6.8) and (6.9) that the values of β and β_j, for $j = 1, \ldots, I_{\boldsymbol{\rho}}$ and γ_j for $j = 1, \ldots, J$, are strictly positive for the choice of Z and Z_i s in the proposition's statement. The details can be found in Online Appendix H. □

Proposition 7. *For function $V(\mathbf{z})$, as defined in (6.3) and (6.4), there is a constant $\xi > 2K + 1$, such that if $\mathbf{z}(t) \in \Gamma[\epsilon_\rho] \setminus \Gamma^\star$, then $\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{z}(t)) < 0$.*

To prove Proposition 7, we first prove the following lemma for the local derivatives over subintervals $[\tau_n, \tau_{n+1})$ defined in Section 5.4.1.

Lemma 2. *Consider the Lyapunov function $V(\mathbf{z})$ defined in (6.3). We can choose the constant $\xi > 2K + 1$ sufficiently large such that the following holds. If at a regular time t, $V(\mathbf{z}(t)) > \epsilon_V$, for some $\epsilon_V > 0$, then there is a $\delta(\epsilon_V) > 0$ such that for any $n \in \{0, \ldots, N_L - 1\}$,*

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

$$\sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}} Z_i \nabla x_{\vec{\mathbf{k}}^{(i)}}^{L(e)}[\tau_n, \tau_{n+1}] > \delta(\epsilon_V) + \sum_{j=1}^{J} Z \frac{\mathrm{d}}{\mathrm{d}t}(y_j(t) - \rho_j)^+ + o(1), \tag{6.10}$$

with probability greater than $1 - o(L^{-2})$.

Proof of Lemma 2. The proof of Lemma 2 is based on using (i) properties of fluid limits in Proposition 5, (ii) the boundedness of local derivatives (Lemma 4 in Online Appendix F), and (iii) the fact that $\frac{\mathrm{d}}{\mathrm{d}t}(y_j(t) - \rho_j)^+ \le -\mu_j(y_j(t) - \rho_j)^+$.

The detailed proof can be found in Online Appendix I. □

Finally, using Lemma 2, we can show that change of $V(\mathbf{z}(t))$ is negative, almost surely, by averaging the change of $V(\mathbf{z}(t))$ over all the subintervals $[\tau_n, \tau_{n+1})$ of $[t, t+\epsilon)$, as we do here.

Proof of Proposition 7. At any regular time t,

$$\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{z}(t)) = -\sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}} Z_i \frac{\mathrm{d}}{\mathrm{d}t} x_{\vec{\mathbf{k}}^{(i)}}^{(e)}(t) + \sum_{j=1}^{J} Z \frac{\mathrm{d}}{\mathrm{d}t}(y_j(t) - \rho_j)^+, \tag{6.11}$$

and

$$\frac{\mathrm{d}}{\mathrm{d}t} x_{\vec{\mathbf{k}}^{(i)}}^{(e)}(t) = \lim_{\epsilon \to 0} \lim_{L \to \infty} \frac{x_{\vec{\mathbf{k}}^{(i)}}^{L(e)}(t+\epsilon) - x_{\vec{\mathbf{k}}^{(i)}}^{L(e)}(t)}{\epsilon}.$$

Hence, using the division of $[t, t+\epsilon)$ into N_L subintervals $[\tau_n, \tau_{n+1})$ of equal size, as defined in Section 5.4.1, we can write

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{z}(t)) &= -\lim_{\epsilon \to 0} \lim_{L \to \infty} \frac{1}{N_L} \sum_{n=1}^{N_L} \sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}} Z_i \nabla x_{\vec{\mathbf{k}}^{(i)}}^{L(e)}[\tau_n, \tau_{n+1}] + \sum_{j=1}^{J} Z \frac{\mathrm{d}}{\mathrm{d}t}(y_j(t) - \rho_j)^+ \\ &\overset{(a)}{<} -\delta(\epsilon_V) - \lim_{\epsilon \to 0} \lim_{L \to \infty} \frac{1}{N_L} \sum_{n=1}^{N_L} o(1) \overset{(b)}{=} -\delta(\epsilon_V) < 0,\end{aligned}$$

where in (a), we used (6.10) of Lemma 2 in every subinterval $[\tau_n, \tau_{n+1}]$, and in (b), we used the property that $\sum_{n=1}^{N_L} o(1)/N_L = o(1)$.

Let E_L be the event that

$$-\frac{1}{N_L} \sum_{n=1}^{N_L} \sum_{i=1}^{C_{\boldsymbol{\rho}}^{(g)}} Z_i \nabla x_{\vec{\mathbf{k}}^{(i)}}^{L(e)}[\tau_n, \tau_{n+1}] + \sum_{j=1}^{J} Z \frac{\mathrm{d}}{\mathrm{d}t}(y_j(t) - \rho_j)^+ > 0.$$

The probability that (6.10) holds for all N_L subintervals is at least $1 - N_L o(L^{-2}) = 1 - o(L^{-1})$, which follows from $N_L = \Theta(1/f(L))$ based on Definition 15. Hence, $\mathrm{IP}(E_L) < o(L^{-1})$, and $\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{z}(t)) < 0$ holds in probability. We can further show that convergence is almost sure. This is because $\sum_{L=1}^{\infty} \mathbb{P}(E_L) < \sum_{L=1}^{\infty} o(L^{-1}) < \infty$, and by the Borel-Cantelli Lemma (Billingsley 2008), $\frac{\mathrm{d}}{\mathrm{d}t} V(\mathbf{z}(t)) < 0$, almost surely. □

Propositions 6 and 7 complete the proof of Theorem 4. □

Proof of Theorem 3. The proof follows from Theorem 4 and Theorems 1 and 2. The details are standard and can be found in Online Appendix J. □

7. Simulation Results

7.1. Evaluation Using Synthetic Traffic

In this section, we evaluate the approximation ratio and convergence properties of DRA. We start by choosing the VM types considering the VM instances offered by major cloud providers like Google Cloud are mainly optimized for either memory, CPU, or regular use. Furthermore, instances are priced proportional to the resources they request, with each resource having a base pricing rate. To simplify simulations, we considered instances that only have memory and CPU requirements. In particular, we used representative VM instances, based on combination of vCPU (virtual CPU) and memory in Table 2.

Last, each vCPU use generates eight rewards per unit time, whereas each gigabyte of memory generates one. This choice was made based on the relative pricing of CPU and memory of VMs offered by Google Cloud,

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Table 2. Representative VM Instances from Google Cloud Based on Combination of vCPU and Memory

vCPU		Memory: Gigabyte per vCPU		
Small	Large	High	Low	Regular
2,4, or 8	32 or 64	8 or 16	1 or 2	4

according to which 8 GB of memory is approximately priced as much as 1 vCPU (Google Cloud 2020b). We generated random collections of VM types, each with three small and three large VMs, with vCPU and memory chosen randomly from Table 2. Servers always have capacity of 80 vCPUs and 640 GB of memory. The normalized workload ρ_j for each VM type j is selected uniformly at random between 0.2 to 2. The statistics we obtained based on 50 randomly generated VM collections and workloads was that, in 23 of them, reward of global greedy was identical to the optimal; on average, its ratio compared with the optimal was 0.972, and in the worst case, it was no less than 0.86. Recall that the optimal can be found by solving Optimization (3.2). For the rest of simulations, we considered a subset of the worst-case VM collection and its corresponding workload, namely, VM types are (1, 1), (4, 16), (2, 32), (32, 256), and $\boldsymbol{\rho}$ rounded to $(2, 1/2, 4/3, 1)$.

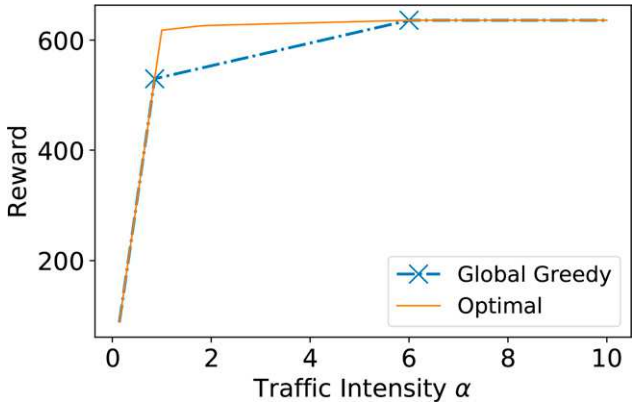
To better understand how workload may affect the approximation ratio, we study this worst-case example and scale its workload $\boldsymbol{\rho}$ by a factor α that ranges from 0 to 10. Figure 2 shows the reward for the global greedy $U^{(g)}[\alpha\boldsymbol{\rho}]$ and the optimal reward $U^*[\alpha\boldsymbol{\rho}]$. We notice there are two critical α points. Before the first point, the workload is low enough such that the global greedy assignment can fully accommodate it; hence, its reward is the same as the optimal that should also be able to accommodate the full workload. The second point is a point above; the workload is high such that it is possible to assign the configuration of maximum reward to all servers without leaving any slots empty. In this case, both the rewards will coincide again and take the maximum possible value. In Figure 2, the two critical points are $\alpha = 6/7$ and $\alpha = 6$. The worst ratio between the reward of global greedy and the optimal occurs at $\alpha = 1$, which is ≈ 0.862. In general, $U^{(g)}[\alpha\boldsymbol{\rho}]$ and $U^*[\alpha\boldsymbol{\rho}]$ might coincide even between the critical points, although this is not the case for this example.

To study the impact of the number of servers L, we run DRA in systems with various number of servers and compare the obtained average normalized reward (normalized with L) with the global greedy reward $U^{(g)}[\boldsymbol{\rho}]$ and the optimal reward $U^*[\boldsymbol{\rho}]$. The arrivals are generated at rate $\rho_j L$, and service times are exponentially distributed with mean 1. The result is depicted in Figure 3, which clearly shows that as the number of servers L becomes large, DRA approaches the global greedy reward and 86% of the optimal reward. Furthermore, Figure 4 shows how the reward of DRA evolves over time and converges to the global greedy reward when $L = 180$.

7.2. Evaluation Using Real Traffic Trace

We evaluate our algorithm using a more realistic setting with arrival and service times extracted from a Google cluster data set (Wilkes 2011). In particular, we extracted tasks that were completed within the time window of the trace and used the first one million in all simulations. Tasks were mapped to types by setting their resource

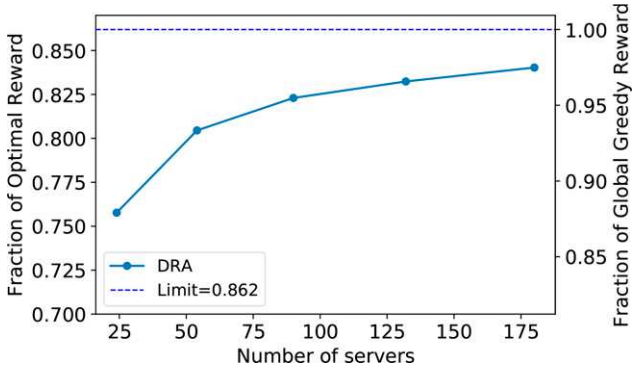
Figure 2. Global Greedy vs. Optimal as Workload $\alpha\boldsymbol{\rho}$ Increases



Note. The rewards coincide outside the marked points.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 3. Reward of DRA as a Fraction of the Optimal Reward (Left y Axis) and That of the Global Greedy (Right y Axis)



requirements to be the largest of the requested resources and rounding it up to the closest power of 1/2. Their reward was set to be equal to their rounded size multiplied by a factor that depends on their priority. The factor is 1, 3, and 9 for priorities 0, 1, and 2, respectively. Tasks have the same type if both their priority and normalized size are equal. The size of servers is normalized to one.

We compare the performance of DRA and three other algorithms:

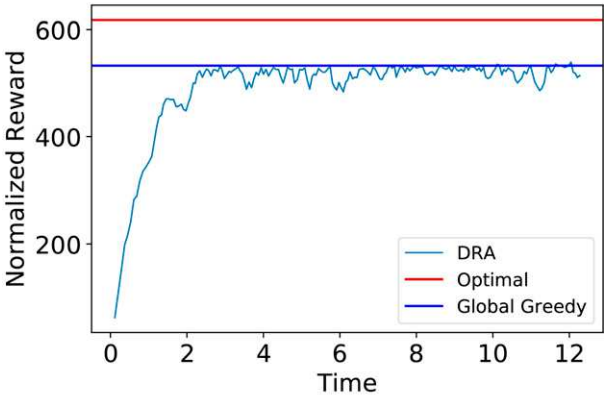
• **Upper Bound.** It solves Optimization (3.1) with $\hat{\mathbf{Y}}(t)$ being the number of jobs in an infinite server system that rejects no jobs. This gives an upper bound on the performance of any algorithm.

• **Power-of-d-Choices.** Upon an arrival, it picks d servers and attempts to schedule the job arrived in the least loaded server if it fits (Xie et al. 2015). We picked $d = 5$, but behavior of the algorithm is not expected to change significantly for larger d.

• **DRA+Preemption.** This is simply an extension of DRA that preempts some of the jobs of priority 0, when a job of type j with priority 1 or 2 gets rejected. Preemption of low-priority jobs is already considered in similar scenarios in Google cluster setting (Verma et al. 2015). Specifically, our algorithm attempts to preempt jobs of priority 0 starting from those of smallest size. Considering reservation factor is $g(L)$ and size of type j job is s_j, preemptions will stop if the total size of preempted jobs is $g(L)s_j$ or no more priority 0 jobs are available. The algorithm finds which jobs to preempt, if any, the same way it finds jobs to migrate so this addition needs minimal changes in implementation.

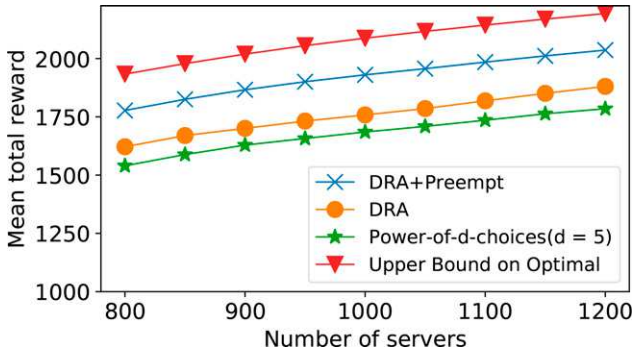
Figure 5 shows the performance results (the time-average of rewards) with varying number of servers, especially considering preemptions in DRA make a great difference. The upper bound may be impossible to achieve by any algorithm.

Figure 4. Convergence of the Reward of DRA to That of the Global Greedy Assignment Over Time When $L = 180$ Servers



Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Figure 5. Comparison of Rewards for Different Numbers of Servers Based on Google Trace

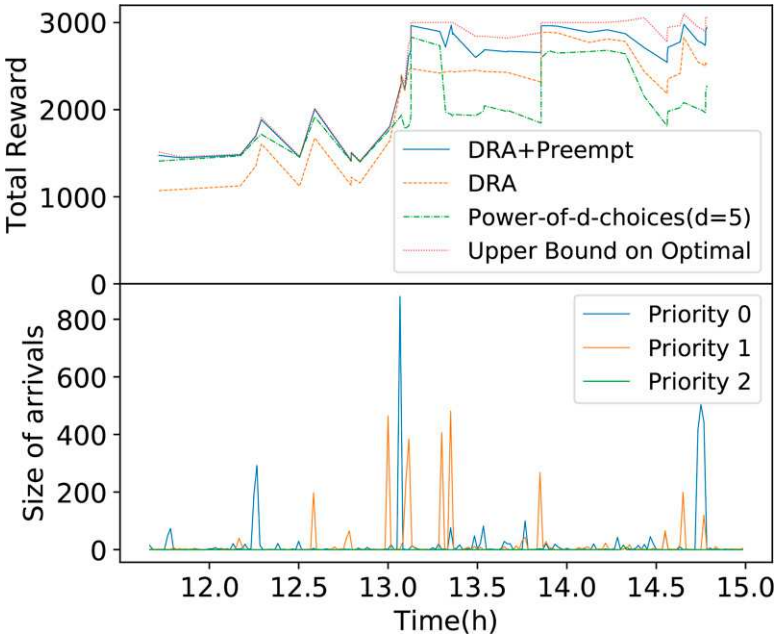


To give more insight, in Figure 6, we plot the total reward over time for all algorithms for a part of the simulation of 1,000 servers, including the corresponding total size of arrivals of all job types of each priority. We notice that the power-of-*d*-choices algorithm can be better than DRA in parts of trace in which a spike in demand of priority 0 jobs is followed by a spike in demand of priority 1 jobs. This is because reservation of DRA is not sufficient to account for spikes in demand, whereas power-of-*d*-choices does not efficiently use the resources of all servers and may have more free capacity when a spike occurs. DRA with preemptions is particularly effective in such scenarios because it does not need to reserve resources in advance. In addition, it makes efficient use of the resources of all the servers the same way DRA does and thus is strictly better than both of the other algorithms in almost all parts of the trace.

8. Discussion on Migrations

We used the term migration in a generic way to describe the transfer of a job from a server to another. Such transfers were needed in the proofs to establish the convergence properties of DRA. Specifically, when the system is overloaded and its state is not close to the global greedy assignment, migrations ensure that the Reject Group servers empty at a proper rate and can be reassigned to required configurations. However, in practice, live migrations could be costly operations. One might suggest to completely eliminate the migration step of our algorithm by stopping scheduling jobs in Reject Group servers that need to empty. Unfortunately, because we do not

Figure 6. Comparison of Reward Over Time of Different Algorithms for a Part of Google Trace



Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

have control over job departures and they happen from different servers independently, the rate at which Reject Group servers empty at the fluid limit will be zero, unless we extend the Reject Group to include a constant fraction of servers (i.e., $O(L)$ servers instead of constant $< C^{(g)}$ servers). For example, we can extend Reject Group to be all servers of rank $J+1$ (refer to Figure 1). Such an approach, apart from being wasteful, will make the convergence oscillatory and slow, if not impossible. Analysis of such an approach and its potential convergence properties is nontrivial and could be a topic of future research.

An alternative way to avoid migrations in DRA is as follows. A migration can be replaced with two operations: termination of the VM that needs to be transferred and restarting the job or rerouting the future request to a deployed VM in a different server. In Section 7.2, we actually presented and evaluated such an algorithm. This is indeed justified for stateless applications (Amazon 2021, RedHat 2021) as any VM instance of the application can serve a request if one or more go down. Moreover, cloud providers usually offer two pricing structures for VMs that are compatible with the revenue interpretation of our reward model: *on-demand* instances (Amazon On-Demand Instances 2021) and *spot* instances (Amazon Spot Instances 2021). The on-demand instances are more expensive and can only be terminated by the user. The spot instances are cheaper but can be terminated by the cloud provider. To make an application highly available and cost efficient, developers typically choose on-demand instances to satisfy critical availability of a service and then setup some auto-scaling rule to increase the number of spot instances when service demand is higher than usual (Amazon 2021). Notice that when servers are fully occupied, the configurations of Reject Group servers in our model are the ones with the lowest reward, so they will include spot instances, and hence they will not need migrations and can be terminated. On-demand instances may still need migration according to our algorithm; however, the provider can simply choose not to perform it if any benefit in the reward will not outweigh the cost of migration.

9. Conclusions

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. Our techniques for analysis of fluid-scale processes on the boundary in our problem and the design of LP-based Lyapunov functions with a unique maximizer at the given desired equilibrium can be of interest on their own.

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. Furthermore, 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 multipool server system, where constant fractions of servers belong to different server types. We postpone the details to a future work. Formalizing the ideas of Section 8 and eliminating the need for migrations can also be an interesting future work.

Acknowledgments

The authors thank the editor-in-chief Shane Henderson, anonymous associate editor, and anonymous reviewers for their comments and suggestions.

Endnotes

[1] If the chain is not irreducible, because it is finite state, there is at least one recurrent class. Starting from any state, the chain eventually enters a recurrent class and remains there forever, so we can simply truncate the Markov to the recurrent class.

[2] In practice, considering L is large yet bounded, one should choose $g(L)$ to be small compared with L, for example, $< L/100$, but sufficiently large to accommodate spikes in traffic.

[3] This is a loose upper bound that considers $C^{(g)}$ different configurations were assigned by our algorithm. Realistically the J configurations considered for assignment in each step of DRA rarely change.

References

Aceto G, Botta A, De Donato W, Pescapè A (2013) Cloud monitoring: A survey. *Comput. Networking* 57(9):2093–2115.

Amazon (2021) Amazon EC2 auto scaling with EC2 spot instances. Accessed January, 17, 2022, https://aws.amazon.com/getting-started/hands-on/ec2-auto-scaling-spot-instances/.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Amazon On-Demand Instances (2021) Amazon on-demand instances. Accessed January, 17, 2022, https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-on-demand-instances.html.
Amazon Spot Instances (2021) Amazon spot instances. Accessed January, 17, 2022, https://aws.amazon.com/ec2/spot/.
Amazon Web Services (2020a) Amazon AWS containers. Accessed January, 17, 2022, https://aws.amazon.com/containers/.
Amazon Web Services (2020b) Amazon EC2 auto-scaler. Accessed January, 17, 2022, https://docs.aws.amazon.com/autoscaling/.
Amazon Web Services (2020c) Amazon web Services (AWS). Accessed January, 17, 2022, https://aws.amazon.com/.
Amazon Web Services (2020d) AWS service level agreements (SLAs). Accessed January, 17, 2022, https://aws.amazon.com/legal/service-level-agreements/.
Andonov R, Poirriez V, Rajopadhye S (2000) Unbounded knapsack problem: Dynamic programming revisited. *Eur. J. Oper. Res.* 123(2): 394–407.
Apache Software Foundation (2019) Apache hadoop yarn. Accessed January, 17, 2022, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html.
Bean NG, Gibbens RJ, Zachary S (1995) Asymptotic analysis of single resource loss systems in heavy traffic, with applications to integrated networks. *Adv. Appl. Probability* 27(1):273–292.
Billingsley P (2008) *Probability and Measure* (John Wiley & Sons, Hoboken, NJ).
Bolch G, Greiner S, Meer H, Trivedi K (2006) *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*, 2nd ed., vol. 95 (John Wiley & Sons, Hoboken, NJ).
Boyd S, Vandenberghe L (2004) *Convex Optimization* (Cambridge University Press, New York).
Cohen C, Rouhling D (2017) A formal proof in Coq of LaSalle's invariance principle. Ayala-Rincón M, Muñoz CA, eds. *Interactive Theorem Proving* (Springer International Publishing, Cham, Switzerland), 148–163.
Corradi A, Fanelli M, Foschini L (2014) VM consolidation: A real case based on openstack cloud. *Future Generation Comput. Systems* 32: 118–127.
Cygan M, Jeż Ł, Sgall J (2016) Online knapsack revisited. *Theory Comput. Systems* 58(1):153–190.
Ghaderi J, Zhong Y, Srikant R (2014) Asymptotic optimality of bestfit for stochastic bin packing. *Performance Evaluation Rev.* 42(2):64–66.
Ghobaei-Arani M, Jabbehdari S, Pourmina MA (2018) An autonomic resource provisioning approach for service-based cloud applications: A hybrid approach. *Future Generation Comput. Systems* 78:191–210.
Google Cloud (2020a) Google Cloud. Accessed January, 17, 2022, https://cloud.google.com/.
Google Cloud (2020b) Google compute engine pricing. Accessed January, 17, 2022, https://cloud.google.com/compute/all-pricing.
Google Cloud (2020c) Google kubernetes. Accessed January, 17, 2022, https://cloud.google.com/kubernetes/.
Guo Y, Stolyar A, Walid A (2018) Online VM auto-scaling algorithms for application hosting in a cloud. *IEEE Trans. Cloud Comput.* 8(3): 889–898.
Gupta V, Radovanovic A (2012) Online stochastic bin packing. Preprint, submitted November 12, 2012. Accessed January, 17, 2022, https://arxiv.org/abs/1211.2687.
Han R, Guo L, Ghanem MM, Guo Y (2012) Lightweight resource scaling for cloud applications. *Proc. EEE/ACM Internat. Sympos. on Cluster, Cloud and Grid Comput.* (IEEE Computer Society, Los Alamitos, CA), 644–651.
Hunt P, Kurtz T (1994) Large loss networks. *Stochastic Processing Appl.* 53(2):363–378.
Hunt P, Laws C (1997) Optimization via trunk reservation in single resource loss systems under heavy traffic. *Ann. Appl. Probability* 7(4): 1058–1079.
Ibarra OH, Kim CE (1975) Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM* 22(4):463–468.
Iwama K, Taketomi S (2002) Removable online knapsack problems. *Proc. Internat. Colloquium on Automata, Languages, and Programming* (Springer, Berlin), 293–305.
Jiang J, Lu J, Zhang G, Long G (2013) Optimal cloud resource auto-scaling for web applications. *Proc. IEEE/ACM Internat. Sympos. on Cluster, Cloud, and Grid Comput.* (IEEE Press, Delft, Netherlands), 58–65.
Jo C, Kim H, Egger B (2020) Instant virtual machine live migration. *Proc. Internat. Conf. on the Econom. of Grids, Clouds, Systems, and Services* (Springer, Berlin), 155–170.
Karthik A, Mukhopadhyay A, Mazumdar RR (2017) Choosing among heterogeneous server clouds. *Queueing Systems* 85(1-2):1–29.
Kellerer H, Pferschy U, Pisinger D (2004) Multidimensional knapsack problem. *Knapsack Problems* (Springer, Berlin), 235–283.
Kelly FP (1991) Loss networks. *Ann. Appl. Probability* 1(3):319–378.
Key PB (1990) Optimal control and trunk reservation in loss networks. *Probability Engrg. Inform. Sci.* 4(2):203–242.
LaSalle J (1960) Some extensions of Liapunov's second method. *IRE Trans. Circuit Theory* 7(4):520–527.
Le T (2020) A survey of live virtual machine migration techniques. *Comput. Sci. Rev.* 38:100304.
Maguluri ST, Srikant R, Ying L (2012) Stochastic models of load balancing and scheduling in cloud computing clusters. *Proc. IEEE INFOCOM* (IEEE, New York), 702–710.
Maguluri ST, Srikant R, Ying L (2014) Heavy traffic optimal resource allocation algorithms for cloud computing clusters. *Performance Evaluation* 81:20–39.
Mao M, Li J, Humphrey M (2010) Cloud auto-scaling with deadline and budget constraints. *Proc. IEEE/ACM Internat. Conf. on Grid Comput.* (IEEE Computer Society, Los Alamitos, CA), 41–48.
Marchetti-Spaccamela A, Vercellis C (1995) Stochastic on-line knapsack problems. *Math. Programming* 68(1-3):73–104.
Martello S, Toth P (1990) An exact algorithm for large unbounded knapsack problems. *Oper. Res. Lett.* 9(1):15–20.
Microsoft Azure (2020) Microsoft Azure. Accessed January, 17, 2022, https://azure.microsoft.com/.
Mukhopadhyay A, Karthik A, Mazumdar RR, Guillemin F (2015) Mean field and propagation of chaos in multi-class heterogeneous loss models. *Performance Evaluation* 91:117–131.
Psychas K, Ghaderi J (2017) On non-preemptive VM scheduling in the cloud. *Proc. ACM on Measurement and Analysis of Comput. Systems*, vol. 1 (ACM, New York), 1–29.
Psychas K, Ghaderi J (2018) Randomized algorithms for scheduling multi-resource jobs in the cloud. *IEEE/ACM Trans. Networks* 26(5): 2202–2215.
Qu C, Calheiros RN, Buyya R (2018) Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Comput. Survey* 51(4):1–33.

Published in Stochastic Systems on February 01, 2022 as DOI: 10.1287/stsy.2021.0091.
This article has not been copyedited or formatted. The final version may differ from this version.

Rampersaud S, Grosu D (2014) A sharing-aware greedy algorithm for virtual machine maximization. *Proc. IEEE 13th Internat. Sympos. on Network Comput. and Applications* (IEEE Computer Society, Los Alamitos, CA), 113–120.

RedHat (2021) Cloud native applications: Stateful vs stateless. Accessed January, 17, 2022, https://www.redhat.com/en/topics/cloud-native-apps/stateful-vs-stateless.

Roy N, Dubey A, Gokhale A (2011) Efficient autoscaling in the cloud using predictive models for workload forecasting. *Proc. IEEE 4th Internat. Conf. on Cloud Comput.*, (IEEE Computer Society, Los Alamitos, CA), 500–507.

Shi J, Luo J, Dong F, Jin J, Shen J (2018) Fast multi-resource allocation with patterns in large scale cloud data center. *J. Comput. Sci.* 26:389–401.

Song W, Xiao Z, Chen Q, Luo H (2013) Adaptive resource provisioning for the cloud using online bin packing. *IEEE Trans. Comput.* 63(11): 2647–2660.

Stillwell M, Vivien F, Casanova H (2012) Virtual machine resource allocation for service hosting on heterogeneous distributed platforms. *Proc. IEEE Internat. Parallel Distributed Processing Sympos. (IPDPS)* 2012, Shanghai, China, 786–797.

Stolyar AL (2013) An infinite server system with general packing constraints. *Oper. Res.* 61(5):1200–1217.

Stolyar AL (2017) Large-scale heterogeneous service systems with general packing constraints. *Adv. Appl. Probability* 49(1):61–83.

Stolyar AL, Zhong Y (2013) A large-scale service system with packing constraints: Minimizing the number of occupied servers. *ACM SIGMETRICS Performance Evaluation Rev.* 41(1):41–52.

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.

Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J (2015) Large-scale cluster management at Google with Borg. *Proc. 10th Eur. Conf. on Comput. Systems* (Association for Computing Machinery, New York), 1–17.

Whitt W (1985) Blocking when service is required from several facilities simultaneously. *ATT Tech. J.* 64(8):1807–1856.

Wilkes J (2011) Google cluster data. Accessed January 17, 2022, https://github.com/google/cluster-data.

Xie Q, Dong X, Lu Y, Srikant R (2015) Power of d choices for large-scale bin packing: A loss model. *Performance Evaluation Rev.* 43(1):321–334.

Zhao Y, Huang Y, Chen K, Yu M, Wang S, Li D (2015) Joint VM placement and topology optimization for traffic scalability in dynamic datacenter networks. *Comput. Networks* 80:109–123.