Optimal Markovian Dynamic Control of Interference-Prone Server Farms

Scott Votke
Stony Brook University
scott.votke@stonybrook.edu

Jazeem Abdul Jaleel
University of Minnesota-Twin Cities
abdul314@umn.edu

Amoghavarsha Suresh
Stony Brook University
amsuresh@cs.stonybrook.edu

Mohammad Delasay
Stony Brook University
mohammad.delasay@stonybrook.edu

Sherwin Doroudi
University of Minnesota-Twin Cities
sdoroudi@umn.edu

Anshul Gandhi
Stony Brook University
anshul@cs.stonybrook.edu

***Abstract*—Interference is a key performance challenge faced by cloud users, and can significantly degrade application performance on virtual machines (VMs). For load-balanced cloud applications, a key question is how to distribute the load among VMs in the presence of interference. Using a Markov decision process (MDP) model, we investigate dynamic control polices to assign jobs among a cluster of VMs that are prone to interference with the goal of maximizing performance. We characterize the structural properties of the MDP optimality equation, and we prove that the optimal control policy in a system with a central queue and arbitrary number of VMs is a threshold policy based on the queue length. The optimal policy is characterized by multiple thresholds depending on the current condition of the VMs, including the number of busy under-interference VMs. We discuss the existence of an ordering among such thresholds, and we prove the ordering for a two-VM system. Our numerical results show that the optimal dynamic policy can significantly improve performance compared to the the commonly employed non-idling policy. For low utilization systems, we observe improvements on the order of around 20%. We further implement our policy in a real-world testbed using the HAProxy load balancer and show that our policy can reduce web server response times by as much as 40–60%, even for time-varying request rates.**

***Index Terms*—Markov Decision Process, Markov Chains, Optimal Control of Queues, Cloud Computing**

I. Introduction

Cloud computing is widely employed by application owners to access economical and virtually unlimited computing resources. Many online services and applications, such as Netflix [12] and Expedia [17], are now provided by cloud-deployed virtual machines (VMs). Applications are typically deployed on a cluster of VMs in the cloud, and load balancers or schedulers are then employed to facilitate scaling by distributing incoming load among the VMs. Recent studies estimate that cloud computing, owing to its resource sharing paradigm, can reduce data center energy consumption and carbon emissions by as much as 50%, translating to annual savings of $12.3 billion in energy costs and 85.7 million metric tons of carbon dioxide [34], [39], [49].

Despite its popularity, however, cloud computing does have its shortcomings. A key performance challenge faced by cloud users (or tenants) is *interference*, which is caused by contention for shared physical resources of the underlying host server, such as CPU, network, or last-level cache, among colocated VMs. Prior studies have shown that application performance on VMs can degrade significantly, by as much 5× due to interference [8], [48], [51]. Our own experimental results, illustrated in Figure 1, show the degradation in throughput for the Cassandra database as a function of the colocated CPU-intensive VM load; see Section VI for details on our experimental setup. Despite the significant body of prior work on alleviating interference [15], [22], [30], [50], this continues to be a notoriously hard problem [32], especially because resource contention among independent colocated VMs is often dynamic and unpredictable [33].

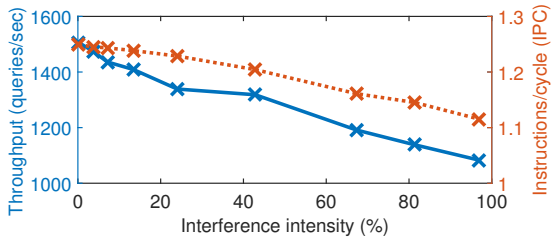


Fig. 1: Empirical results for decrease in throughput and IPC as a function of interference intensity.

This paper takes a complementary approach to interference and poses the following question "in the presence of dynamic interference, how can we schedule jobs among a cluster of VMs to minimize the impact of interference on application performance?" Specifically, we focus on the scheduling component of the application that is responsible for queueing and dispatching jobs to back-end VMs. Our goal is to derive the optimal control policy for the scheduler as a function of various system and workload parameters, including request rate, interference intensity, interference duration, etc.

As such scheduling decisions are made dynamically, and each decision affects the system evolution over time, we take a dynamic view to the problem. We consider a Markov queueing model of a multi-VM homogeneous system where any of the VMs can be independently under interference for some time.

The scheduler (or the controller) needs to decide whether Jobs should be routed to an idle under-interference VM or should be queued upon a system state change (a job arrival or a service completion); see Section III for our model formulation. We formalize the model as a Markov decision process (MDP) that can be solved to find the scheduler's optimal control policy to minimize the expected total number of job in the system; the MDP formulation is detailed in Section III-A.

After characterizing the structural properties of the MDP optimality equation, we prove the optimal scheduling policy in a system with any number of VMs is a queue length based threshold policy; that is, depending on the number of busy under-interference VMs, it is optimal to not utilize the under-interference VMs if the queue length is below a threshold, and but, it is optimal to utilize the under-interference VMs as the queue length exceeds the threshold. We show that a threshold policy is characterized by multiple thresholds, which depend on the state of the VMS (how many are under interference and how many are free/busy). For a two-VM system, we show the existence of three thresholds, and we prove the ordering among them (Section IV).

We evaluate the performance of our threshold policy via extensive numerical experiments (Section V). Our results show that, for a wide range of workload and system parameter values, the threshold policy results in much better performance compared to a non-idling policy (i.e., a threshold of essentially zero). Our results also highlight the non-trivial impact of model parameters on performance improvement. Specifically, we find that the greatest benefits from using an optimal policy (as compared to the non-idling policy) arise in settings with a modest (relatively light) level of traffic and intermediate interference event rates; implementing the optimal policy matters most in the cases far from the "extremes." We also use our numerical results to investigate the performance improvement afforded by our threshold policy for different numbers of VMs. We find that in settings with more than two VMs, the *potential* for improvement is greater than in settings with only two VMs.

Finally, to validate the practical applicability of our work, we also implement and experimentally evaluate our threshold policy for a cluster of web server VMs (Section VI). Our experimental results allow us to evaluate the efficacy of our policy under realistic processor sharing, and also allow us to compare against existing interference-mitigation strategies. We implement our policy using the open-source HAProxy [3], that is employed as a load balancer in front of Apache web server VMs (using KVM [25] in Linux) serving dynamic web user traffic and under processor interference from colocated CPU- and cache-intensive VMs. Our implementation results show that our theoretically inspired threshold policy provides substantial benefits, to the tune of 40–60%, over other dynamic baseline policies that are employed in web services.

To summarize, this paper makes the following contributions:

- We present the formulation of VM interference in clouds as a Makovian model, and the scheduling problem that arises in this setting as an MDP.
- We prove the optimality of a threshold policy and establish the ordering of the thresholds for different VM states.
- We present numerical results to illustrate the performance of our policy and to compare it against a non-idling policy.
- We implement our policy using HAProxy and experimentally evaluate its performance for a web serving cluster of VMs. We show that, even under time-varying request rates, our threshold policy reduces mean response time by 40–60% compared to other dynamic baseline scheduling policies. The improvements are even more pronounced (about 70%) at higher percentiles of response time.

II. Prior Work

The model we consider in this paper is a multi-server (or multi-VM) system where the service rate for any server independently switches between fast (interference-free) and slow (under interference). The key problem that we aim to address for this model is the decision for an incoming job or a queued job when the only available (idle) server is under interference. The problem and model we study is similar to the "slow server problem", which is an M/M/2 system where the first server has a higher service rate than the second server and both are completely reliable. Unlike our problem, however, the servers in the slow server problem have fixed service rates. Several extensions of the slow server problem have been proposed and analyzed, including the investigation of the optimal threshold policy. We now discuss these works in detail, starting from the slow server problem.

In 1981, R. L. Larsen proposed, but did not prove, in his Ph.D. thesis that a threshold policy exists for when to utilize the slow server in such a system [27]. Three years later, Lin and Kumar proved the existence of such a threshold policy on the queue length for the system using policy iteration [29], and in 1995 Koole presented a more concise proof of the threshold using value iteration [26].

Rubinovitch explored the slow server problem analytically using several models to obtain explicit results on the utilization of the slow server [40]. Specifically, the author explored whether it was beneficial to utilize the slow server at all or not. Under different scheduling policies (random, fast-server–first, etc.), traffic intensity thresholds are established, using generating functions, under which the slow server should be avoided. Rubinovitch later extended his analysis to consider the case where incoming customers decide whether to join the queue for the fast server or to go to the idle slow server [41], similar to the decision we investigate in this paper for our problem. Rubinovitch first conducted a stochastic analysis of the system and derived the steady state probabilities. Then, using these probabilities, the author derived the optimal queueing threshold values below which customers should wait in the queue for the fast server and beyond which customers should skip the queue and utilize the idle slow server.

Luh and Viniotis study an M/M/k server system with het-

erogeneous servers [31]. Each server has a lower service rate than the previous one, or in other words, $\mu_1 > \mu_2 > \ldots > \mu_k$. For this system, the paper proved the optimality of a threshold policy with multiple thresholds whereby if server i is currently being utilized, server $(i+1)$ will be utilized only if the queue length is greater than or equal to some threshold, n_i. Because of the complexity of the model, they used Linear Programming to prove the optimality of such a policy. We also consider a multi-server system, but each server in our model may be fast or slow, depending on interference.

More recently, Rykov and Efrosinin proposed an M/M/k version of the slow server problem with a finite queue [42]. The authors proved, using value iteration, that a threshold policy is optimal in this setting. Furthermore, they showed that by considering the minimum average service time at each server, the most efficient server to utilize can be chosen.

Efrosinin later considered an M/M/2 system subject to complete and partial breakdowns [16]. The author assumed that only one of the servers is subject to breakdown and the other is completely reliable. Furthermore, he allows for different levels of service based on the intensity of the breakdown. The optimal threshold policy is proved through policy iteration, along with some structural properties of the policy.

Özkan and Kharoufeh extended the slow server problem to a failure-prone server system wherein the faster server is subject to random breakdown [37]. The authors proved the existence of two thresholds for when to queue jobs for this system, one for when the fast server is operational and the other for when the fast server is unavailable due to a failure. The system is modeled as a discounted MDP and the threshold policies are proved using structural properties of the operators through the value iteration algorithm. The authors then extended the analysis to an average cost problem. Our model can be viewed as an extension of the slow server problem with failures analyzed by Özkan and Kharoufeh but with the key variation that any server can experience interference, and that under interference, the service rate decreases to a non-zero value.

Votke et al. [47] analyzed the performance of a multi-server system under interference. Specifically, the authors considered non-idling policies, including random routing and interference-free–first routing, and solved the associated Markov chain using Matrix Analytic Methods [28]. While similar to our model, the above referenced paper does not focus on the existence or analysis of optimal policies. Our goal in this paper is to specifically focus on the development of optimal policies. In addition to proving the optimality of the threshold policy for our multi-server problem, we also prove the ordering of the various thresholds that exist for our problem. Further, we establish structural properties for the optimal policy, which are critical for proving the optimality of the threshold policy.

Casale et al. also recently modeled the variability and transient behavior of interference in cloud deployments [10], [11]. In their model, the authors view interference as a *separate* process and model the overall phenomenon as a two-stage process with transitions between the stages. Furthermore, they model a cloud deployment as a *closed* queueing network with a specified number of servers at each station. Due to the intractability of exact solutions for the transient behavior of the queueing network, the authors come up with approximations of the stationary distribution.

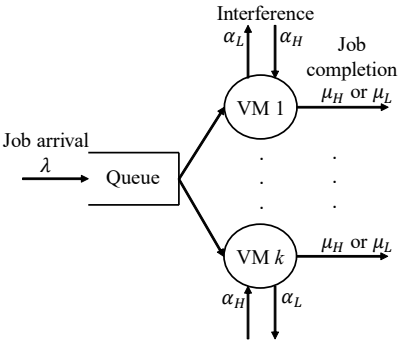


Fig. 2: Schematic system representation

III. Model and MDP Formulation

To analyze optimal scheduling policies that mitigate the impact of interference, we first develop a performance model for VMs under interference. We consider a cluster of VMs that together host an application, and model the number of outstanding jobs or tasks for this application as a Markov chain, making the required Markovian assumptions. We consider the VMs to be fed by a central queue, representing the application scheduler or load balancer, as is the case for online services [13], [21], [44]. The scheduler's job is to decide on the routing of requests from the central queue to idle VMs. Consider a k-VM deployment, with homogeneous VMs, with average service rate of μ_H jobs/s when there is no interference. Under interference, we assume that the average service rate drops to $\mu_L < \mu_H$. We can also consider multiple levels of interference, resulting in a range of lower service rates, say $\mu_{L1} < \mu_{L2} < \ldots < \mu_H$. However, this substantially complicates the state space and the decision process; we thus only consider a single level of interference in this paper. While VMs may be colocated on the same physical host and subject to correlated interference, for analytical tractability, we consider interference as an independent process that strikes a VM with rate α_H and leaves a VM with rate α_L. The independence assumption can be justified in today's large-scale public clouds as VMs are often distributed across thousands of physical hosts [1], [4]. We assume that arrivals to the system follow a Poisson process with mean arrival rate (across all VMs) of λ req/s. This assumption of Poisson arrivals mirrors prior work on leveraging queueing theory to model cloud computing [45], [47]. Fig. 2 illustrates the system of interest.

Based on the above, we consider a continuous-time Markov chain (CTMC) on the state space S in which a state s is a four-tuple of the form (s_1, s_2, s_3, s_4), where:

- $s_1 \in \{0, 1, \ldots\}$ represents the number of jobs in the system (system size),

- $s_2 \in \{0, 1, \dots\}$ represents the number of jobs in the queue (queue size),
- $s_3 \in \{0, \dots, k\}$ represents the number of VMs under interference, and
- $s_4 \in \{0, \dots, s_3\}$ represents the number of busy VMs under interference.

These four state variables fully define the state of the system. We define the following *auxiliary* state variables based on the above main state variables:

- Number of idle VMs: $a_1 = k - s_1 + s_2$,
- Number of idle VMs under interference: $a_2 = s_3 - s_4$,
- Number of idle, interference-free VMs: $a_3 = a_1 - a_2$, and
- Number of busy, interference-free VMs: $a_4 = s_1 - s_2 - s_4$.

We use the terms VM and *server* interchangeably in the rest of the paper as VMs are the servers in our queueing system.

A. MDP formulation

We model the system as a Markov decision process with two *decision epochs*: (1) when a job arrives and there is at least one idle under-interference server, and (2) a service completion from an under-interference server when the queue size is not empty. The *actions* available to the controller in each decision epoch are whether to utilize an under-interference server by routing one of the jobs in the queue (or the arriving job to the system) to that server, or not utilize the server (not route any job from the queue to the server).

Let $\mathbb{I}_E$ be an indicator function for condition E; i.e., $\mathbb{I}_E = 1$ if E holds, and $\mathbb{I}_E = 0$ otherwise, Also, let $\Lambda = \lambda + k(\mu_H + \mu_L + \alpha_H + \alpha_L)$ be the maximum output rate of any state; i.e., $1/\Lambda$ is the minimum expected time for a possible *event* occurrence (job arrival, job departure, and interference occurrence or departure) in the system. We define $(\mathbf{s} \pm \mathbf{e_i})$ to show an increase or decease by one in the i^{th} variable of state $\mathbf{s} = (s_1, s_2, s_3, s_4)$. We consider the objective function of minimizing the expected total number of jobs in the system. We employ the Uniformization method to convert the continuous-time model to its equivalent discrete-time version, and we express the MDP optimality equation as follows[1]:

$$\nu(\mathbf{s}) = \frac{r(\mathbf{s})}{\Lambda} + T\nu(\mathbf{s}), \tag{1}$$

where the reward in state $\mathbf{s}$ is s_1 ($r(\mathbf{s}) = s_1$), and we define the operator T and the minimization operators T_a and T_s as follows:

$$\begin{aligned} T\nu(\mathbf{s}) = \frac{1}{\Lambda}\Big(& \\ & \lambda\left(\mathbb{I}_{a_1=0}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) + \mathbb{I}_{a_3>0}\nu(\mathbf{s}+\mathbf{e_1}) + \mathbb{I}_{a_3=0}\mathbb{I}_{a_2>0}T_a\nu(\mathbf{s})\right) \\ & + a_4\mu_H\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_2}\mathbb{I}_{s_2>0}) \\ & + s_4\mu_L\left(\mathbb{I}_{s_2=0}\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}) + \mathbb{I}_{s_2>0}T_s\nu(\mathbf{s})\right) \\ & + \alpha_H\left(a_3\nu(\mathbf{s}+\mathbf{e_3}) + a_4\nu(\mathbf{s}+\mathbf{e_3}+\mathbf{e_4})\right) \\ & + \alpha_L\left(a_2\nu(\mathbf{s}-\mathbf{e_3}-\mathbf{e_2}\mathbb{I}_{s_2>0}) + s_4\nu(\mathbf{s}-\mathbf{e_3}-\mathbf{e_4})\right) + \delta(\mathbf{s})\nu(\mathbf{s})\Big). \end{aligned} \tag{2}$$

$$T_a\nu(\mathbf{s}) = \min\{\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}), \nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_4})\}, \tag{3}$$

$$T_s\nu(\mathbf{s}) = \min\{\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}), \nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_2})\}, \tag{4}$$

and we define $\delta(\mathbf{s})$ as the rate to remain in state $\mathbf{s}$:

$$\delta(\mathbf{s}) = \Lambda - \lambda - a_2\alpha_H - a_3\alpha_L - a_4\left(\mu_H + \alpha_H\right) - s_4\left(\mu_L + \alpha_L\right).$$

The minimization operators T_a and T_s (Eq. (3) and Eq. (4)) represent the controllers' decisions at the job arrival and service completion decision epochs, respectively. The first argument inside the minimization function in both T_a and T_s corresponds to the decision of not utilizing an idle under-interference server in state $\mathbf{s}$, and the second argument corresponds to the decision of utilizing an idle under-interference server in state $\mathbf{s}$. In Eq. (1), all the main and auxiliary state variables are related to state $\mathbf{s}$.

Given that the system size at the beginning of a period is s_1, the value function at the end of the period (of length $1/\Lambda$) is evaluated as the value contributed by the s_1 jobs over the duration of the period (s_1/Λ) plus the optimal value function evaluated at the end of the previous period ($T\nu(\mathbf{s})$). We have structured the expressions for $T\nu(\mathbf{s})$ in (2) based on the possible events occurring in each time interval (*period*) with length $1/\Lambda$ of the discrete-time MDP; each line in (1) corresponds to an event in a period as we explain below:

- A job arrives with probability λ/Λ: (1) If $a_1 = 0$, the controller has to queue the job (transition to $\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$); (2) If $a_3 > 0$, the controller sends the job to an idle interference-free server (transition to $\mathbf{s}+\mathbf{e_1}$); (3) If $a_3 = 0$ and $a_2 > 0$, the controller decides whether to not utilize an idle under-interference server and queue the job (transition to $\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$) or utilize an under-interference server (transition to $\mathbf{s}+\mathbf{e_1}+\mathbf{e_4}$).
- A job departs an interference-free server with probability $a_4\mu_H/\Lambda$: (1) If $s_2 = 0$, the interference-free server that has recently completed service remains idle (transition to $\mathbf{s}-\mathbf{e_1}$); (2) If $s_2 > 0$, the controller sends the job to the idle interference-free server (transition to $\mathbf{s}-\mathbf{e_1}-\mathbf{e_2}$).
- A job departs an under-interference server with probability $s_4\mu_L/\Lambda$: (1) If $s_2 = 0$, no follow-up event or decision is available (transition to $\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}$); If $s_2 > 0$, the controller decides whether to not utilize the idle under-interference server (transition to $\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}$) or utilize the under-interference server by routing one of the jobs in the queue to the server (transition to $\mathbf{s}-\mathbf{e_1}-\mathbf{e_2}$).
- Interference occurrence: (1) To an idle interference-free server with probability $a_3\alpha_H/\Lambda$ (transition to $\mathbf{s}+\mathbf{e_3}$); (2)

[1]For the objective function of minimizing the discounted expected total number of jobs in the system, Λ should be replaced with $\Lambda + \beta$, where $\beta > 0$ denotes the discount parameter.

To a busy interference-free server with probability $a_4\alpha_H/\Lambda$ (transition to $\mathbf{s}+\mathbf{e_3}+\mathbf{e_4}$).

- Interference departure: (1) From an idle under-interference server with probability $a_2\alpha_L/\Lambda$: if $s_2=0$, no follow-up event or decision is available (transition to $\mathbf{s}-\mathbf{e_3}$), and if $s_2>0$, the controller routes a job from the queue to the new interference-free server (transition to $\mathbf{s}-\mathbf{e_2}-\mathbf{e_3}$)[2]; (2) From a busy under-interference server with probability $s_2\alpha_L/\Lambda$, in which case no follow-up event or decision is available (transition to $\mathbf{s}-\mathbf{e_3}-\mathbf{e_4}$).
- No event occurs and the state remains in $\mathbf{s}$ with probability $\delta(\mathbf{s})/\Lambda$.

IV. The Optimal Policy and Structural Properties

In this section, we first prove several structural properties of the value function. Then, we employ the structural properties to prove that the optimal policy for utilizing the under-interference servers is threshold-based, defined on the queue length. We show that for each system we can realize different thresholds on the queue length depending on the number of idle under-interference VMs beyond which utilizing the idle under-interference VMs is optimal. We prove the thresholds ordering for a two-server system.

A. Structural properties of the optimal policy

Let's define the first difference operator $D_{ij}\nu(\mathbf{s}) = \nu(\mathbf{s}+\mathbf{e_i}+\mathbf{e_j})-\nu(\mathbf{s})$, and the second difference operators $D^2_{ij}\nu(\mathbf{s}) = D_{ij}\nu(\mathbf{s}+\mathbf{e_i}+\mathbf{e_j})-D_{ij}\nu(\mathbf{s})$ and $D_{ij}D_{ik} = D_{ik}D_{ij}$. The first difference operators measure the marginal increase/decrease in the value function, compared to its value in state $\mathbf{s}$, if an under-interfere server is not utilized ($D_{12}\nu(\mathbf{s})$) or utilized ($D_{14}\nu(\mathbf{s})$). The second difference operators measure the marginal increase/decrease in the value function due to the utilize/not utilize decisions in two different states ($\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$ vs. $\mathbf{s}$, or $\mathbf{s}+\mathbf{e_1}+\mathbf{e_4}$ vs. $\mathbf{s}$).

Let Θ be the set of functions ν defined on the state space S that have the following properties:

P1. Non-decreasing in $\mathbf{e_1}$ and $\mathbf{e_2}$: $D_{12}\nu\geq 0$.
P2. Non-decreasing in $\mathbf{e_1}$ and $\mathbf{e_4}$: $D_{14}\nu\geq 0$.
P3. Supermodularity: $D_{12}D_{14}\nu\geq 0$.
P4. Diagonal dominance: $D^2_{12}\nu\geq D_{14}D_{12}\nu$.
P5. Convexity in $\mathbf{e_1}$ and $\mathbf{e_2}$: $D^2_{12}\nu\geq 0$.
P6. Convexity in $\mathbf{e_1}$ and $\mathbf{e_4}$: $D^2_{14}\nu\geq 0$.

Lemma IV.1 shows that properties **P1**-**P6** are preserved under the operators T_a (defined in Eq. (3)), T_s (defined in Eq. (4)), and T (defined in Eq. (2)), and the optimal value function ν. Before presenting the Lemma, we first explain properties **P1**-**P6** below.

The first two properties state that queueing an arriving job (**P1**) and routing it to an idle under-interference server (**P2**) both weakly increase the value function; in total, **P1** and **P2** state that the value function is weakly increasing in the system size. Property **P3** states that the marginal impact of queueing (not utilizing an idle under-interference server) on the value function weakly increases with the number of busy under-interference servers, or the marginal impact of utilizing an idle under-interference server weakly increases with the queue length. Property **P4** states that given that an under-interference server is not utilized upon a decision epoch, the marginal impact of not utilizing the under-interference server in the next decision epoch is weakly greater than the impact of utilizing it; i.e., as the queue length increases, the benefit of utilizing eventually outweighs the benefit of queueing. Property **P5** states that the impact of queueing an arriving job on the value function increases convexly with the queue length, and Property **P6** states that the impact of routing an arriving job to an idle under-interference server increases with more servers being utilized; in total, **P5** and **P6** state that the value function is convex in the system size.

Lemma IV.1. *Let τ be a real valued function defined on state S. If $\tau\in\Theta$, then (a) $T_a\tau\in\Theta$, (b) $T_s\tau\in\theta$, and (c) $T\tau\in\Theta$. Furthermore, (d) the optimal value function $\nu\in\Theta$.*

Our proof strategy for Lemma IV.1 is similar to the proof strategy in [14]. We omit the proofs for Lemmas IV.1.a-IV.1.b (available online [46]) and present the proof for Lemmas IV.1.c-IV.1.d below.

Proof for Lemma IV.1.c. It is easy to verify that Θ is closed under convex combinations; if $\tau_1,\tau_2\in\Theta$, then $\alpha\tau_1+(1-\alpha)\tau_2\in\Theta$. It is obvious that T is a convex combination of τ, T_a, and T_s, as the summation of the coefficients in (2) equals one. By the assumption in Lemma IV.1, $\tau\in\Theta$, and by parts (a) and (b) of Lemma IV.1, $T_a\in\Theta$ and $T_s\in\Theta$. Therefore, we can conclude $T\in\Theta$. □

Proof for Lemma IV.1.d. The proof is by induction and the direct application of the value iteration algorithm. Initializing the algorithm with $\nu_0(\mathbf{s})=0$, it is clear that $\nu_0(\mathbf{s})\in\Theta$. For the induction assumption in period t, let's assume that $\nu_t(\mathbf{s})\in\Theta$. By the definition of T, $\nu_{t+1}(\mathbf{s})=r(\mathbf{s})/\Lambda+T\nu_t(\mathbf{s})$. By Lemma IV.1.c, $T\nu_t(\mathbf{s})\in\Theta$ as by the induction assumption $\nu_t(\mathbf{s})\in\Theta$. The term s_1/Λ, which is added to $\nu_t(\mathbf{s})$ to evaluate $\nu_{t+1}(\mathbf{s})$, does not violate **P1**-**P6**, and therefore $\nu_{t+1}(\mathbf{s})$ preserves **P1**-**P6**. Furthermore, $\nu_{t+1}(\mathbf{s})=r(\mathbf{s})/\Lambda+T^{t+1}\nu_0(\mathbf{s})$; as $t\to\infty$ by the value iteration $\nu_t(\mathbf{s})\to\nu(\mathbf{s})$; thus, the optimal value function is in Θ. □

Lemma IV.1 enables us to characterize the optimal policy in the next section.

B. Characterizing the optimal policy

Proposition IV.2. *The optimal policy is a threshold-based policy on the queue length; i.e:*

- *If it is optimal to utilize an idle under-interference server in state $\mathbf{s}$, then it is optimal to utilize an idle under-interference server in state $\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$.*
- *If it is optimal to not utilize an idle under-interference*

[2]This means we are considering non-idling policies for interference-free servers in our model. Non-idling policies are proved to be optimal in many systems, and it is natural to focus on such policies in our model as we focus on minimizing the number of jobs in the system.

server in state $\mathbf{s}$*, then it is optimal to not utilize an idle under-interference server in state* $\mathbf{s}-\mathbf{e_1}-\mathbf{e_2}$ *(when* $s_2>1$*).*

Proof. We use the structural properties **P1-P6** to prove the proposition separately for the two decision epochs: job arrival and service completion.

Job arrival epoch. We need to prove if it is optimal to utilize in $\mathbf{s}$ upon an arrival, it is also optimal to utilize in $\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$:

$$\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_4}) \leq \nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) \implies \nu(\mathbf{s}+\mathbf{2e_1}+\mathbf{e_2}+\mathbf{e_4}) \leq \nu(\mathbf{s}+\mathbf{2e_1}+\mathbf{2e_2}),$$

or in terms of the difference operators:

$$D_{14}\nu(\mathbf{s}) - D_{12}\nu(\mathbf{s}) \leq 0 \implies D_{14}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) - D_{12}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) \leq 0. \quad (5)$$

Beginning with the consequent, we have

$$\begin{aligned} &D_{14}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) - D_{12}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) \\ &\quad = D_{14}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) - D_{12}\nu(\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}) \\ &\quad\quad + D_{14}\nu(\mathbf{s}) - D_{14}\nu(\mathbf{s}) + D_{12}\nu(\mathbf{s}) - D_{12}\nu(\mathbf{s}) \\ &\quad = D_{12}D_{14}\nu(\mathbf{s}) - D_{12}^2\nu(\mathbf{s}) + D_{14}\nu(\mathbf{s}) - D_{12}\nu(\mathbf{s}) \leq 0, \end{aligned}$$

because $D_{12}D_{14}\nu_t(\mathbf{s}) \leq D_{12}^2\nu_t(\mathbf{s})$, due to the diagonal dominance property of ν, and $D_{14}\nu_t(\mathbf{s}) \leq D_{12}\nu_t(\mathbf{s})$, because of the antecedent in (5); thus, $D_{14}\nu_t(\mathbf{s}) - D_{12}\nu_t(\mathbf{s}) \leq 0$.

Service completion epoch. We need to prove if it is optimal to utilize in $\mathbf{s}$ upon a service completion from an under-interference server, it is also optimal to utilize in $\mathbf{s}+\mathbf{e_1}+\mathbf{e_2}$:

$$\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_2}) \leq \nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}) \implies \nu(\mathbf{s}) \leq \nu(\mathbf{s}+\mathbf{e_2}-\mathbf{e_4}),$$

or in terms of the difference operators:

$$D_{14}\nu(\mathbf{s}-\mathbf{2e_1}-\mathbf{e_2}-\mathbf{e_4}) - D_{12}\nu(\mathbf{s}-\mathbf{2e_1}-\mathbf{e_2}-\mathbf{e_4}) \leq 0 \implies D_{14}\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}) - D_{12}\nu(\mathbf{s}-\mathbf{e_1}-\mathbf{e_4}) \leq 0. \quad (6)$$

Note that (6) is the same implication as (5) just evaluated at a different state. Thus, the implication holds, which means that the threshold is the same upon an arrival and upon a service completion. □

For a specific system, there are multiple optimal thresholds on the queue length depending on the number of under-interference servers and whether they are idle or busy. To specify the values of the thresholds, one can solve the optimality equation using, for example, the value iteration algorithm. Though the value iteration algorithm is computationally expensive, especially for larger systems, the model needs to be solved only once for a specific set of system parameters to obtain the optimal thresholds. After obtaining the thresholds, implementing the optimal policy requires minimum effort. For a two-server system, three thresholds exist:

- Q_i: Optimal queue length threshold for when there is one busy interference-free server and one idle under-interference server.
- Q_{ib}: Optimal queue length threshold for when there is one busy under-interference server and one idle under-interference server.
- Q_{ii}: Optimal queue length threshold for when there are two idle under-interference servers.

Once reached from below, the above queue length thresholds suggest utilizing an idle under-interference server, and once passed over from above, they suggest not utilizing. For example, $Q_{ii}=5$ indicates that we should queue incoming jobs (and not send to any idle under-interference server) when the number in queue is less than 5, but should utilize the under-interference server once queue length reaches 5. It also implies that if the queue length drops below 5, we should stop utilizing idle under-interference servers and instead queue the jobs.

The above thresholds can be mapped to the thresholds on the system size, N, as $N_i = Q_i + 1$, $N_{ib} = Q_{ib} + 1$, and $N_{ii} = Q_{ii}$. Proposition IV.3 shows the ordering of the three thresholds; the proof of the proposition is omitted (available online [46]).

Proposition IV.3. *The order of the system size based thresholds follow* $N_i \geq N_{ib} \geq N_{ii}$.

It is intuitive that N_i could be larger than the other two thresholds especially when $\mu_H >> \mu_L$ and $\mu_H >> \alpha_L$; the controller would be better off not utilizing an idle under-interference server for some time, and let the queue build up in the anticipation that the interference-free server could serve all current jobs in the queue sooner than the time it takes for interference to leave the under-interference server.

When both servers are under interference, the threshold depends on the status of the servers. If none of the under-interference servers are being utilized, then the system size is increasing at rate λ. If one of the under-interference servers is busy and the other is idle, then system size is (only) increasing at rate $(\lambda - \mu_L)$. This suggests that in the latter case, the penalty of waiting for interference to leave one of the servers is smaller, and so the controller may risk a larger threshold ($N_{ib} \geq N_{ii}$).

Note that the N_{ii} implies that it is better to wait for interference to leave an under-interference server than to utilize an under-interference server, but only while the queue length is shorter than $Q_{ii} = N_{ii}$. Thus, once the system size reaches N_{ii}, when the next arrival occurs we utilize one of the under-interference servers, resulting in the same queue length, N_{ii}. At this point, if a new job arrives, the controller will face the same decision of whether to route to an under-interference server or to wait for interference to leave an under-interference server, thus resulting in the same optimal threshold on the queue length but with one more job in the system. Therefore, N_{ii} and N_{ib} differ by at most one.

V. Numerical Results

In this section we examine a variety of problem settings—exploring a range of arrival (λ), service (μ_H, μ_L), and in-

terference rate (α_H, α_L) parameters—in both the two-VM and multiple VM settings. In particular, for each setting, we use numerical techniques to determine the optimal decision policy and we report the performance of the optimal policy. To facilitate the reporting of the results, we use the term offered load, ρ, to refer to the quantity $\lambda / \left(k \cdot \frac{\mu_H/\alpha_H + \mu_L/\alpha_L}{1/\alpha_H + 1/\alpha_L}\right)$. In many cases, we also measure the percentage improvement of the optimal policy over the *non-idling* policy (described below), which serves as a benchmark. The numerical results presented in this section allow us to extract insights regarding what the optimal policies "look like" across a variety of parameter settings, and identify which settings offer the greatest opportunities for improvement when one adopts a policy more sophisticated than our benchmark policy. As not all parameters are practical, in Section V-E, we describe problem parameters that are expected to appear in real-world systems. Our key findings in this section include:

- An intermediate traffic intensity allows for the greatest improvements upon the non-idling policy, as in this range the optimal policy will make less use of VMs under interference (see Section V-C).
- In time-scales where VMs alternate between the interference-free and interference states at an intermediate rate, there is substantial room for improvement over the non-idling policy; there is not much room for improvement when a VM is likely to switch between the two states several times while serving a job, or on the other extreme where the VM will persist in interference for many job services (see Section V-D).
- In the "sweet spot" where the two intermediate cases mentioned above overlap, one can achieve improvements of up to 20% over the non-idling policy in the two-VM setting (see Section V-E).
- The potential for improvement is even higher when there are more than two VMs (see Section V-F).

A. Determining the optimal policy

In order to compute optimal decision policies we use the *value iteration procedure* on the MDP model corresponding to the given parameters (see Sections III and IV for details). The value iteration procedure, which is outlined in [38], requires truncating the MDP model's underlying state space order to examine a finite state space, and iterative computing value functions over this state space until a convergence criterion is satisfied (e.g., the value function changed less than a specified small amount in the most recent iteration). With the convergence criteria reached, the actual value function is known (modulo a small approximation error), enabling one to extract the optimal decision policy (and evaluate its performance), so long as a sufficiently large state-space and a sufficiently strict convergence criterion was considered.

The value iteration technique is powerful, and can in theory accommodate an arbitrary number of VMs, k. However, some parameter sets allow for the extraction of the correct optimal policy only when a very large state space is considered. That is, for these parameter sets, one must choose a "truncation point" that is "far" into the chain's state space in order to compute the optimal policy correctly. In our computational experiments, such issues arose primarily in 2-VM settings with a heavy load and low rates of entering and leaving interference (i.e., $\rho \approx 1$ and $\alpha_H, \alpha_L \approx 0$). In these settings, the value iteration approach would require very long computation runtimes, so we opted to use an alternative approach in these settings. Specifically, we used *matrix analytic methods* (MAM)—as detailed in [28] and Chapter 21 of [20]—to evaluate the performance of threshold policies for an expansive range of thresholds, and selected the best-performing of these as the optimal policy. We validated our MAM-driven approach by comparing the optimal policies it found with those found by value iteration approach across a number of parameter settings. In all cases where the value-iteration approach converged, the two approaches agreed on the same optimal policy. While the MAM-driven approach is particularly computationally efficient in the two-VM setting, we found the value-iteration approach to be less sensitive to the curse of dimensionality in terms of computational efficiency.

λ	μ_H	μ_L	α_H	α_L	k
{1,2,3,...,50}	50	1	0.3	0.3	2

TABLE I: Parameters for *Environment 1*.

B. The non-idling policy, a benchmark

In order to measure the benefit of using a sophisticated MDP framework to extract and implement the optimal policy, we compare the performance of said optimal policy to a very simple benchmark: the *non-idling* policy. The non-idling policy utilizes any available idle under-interference server when there are no idle interference free servers *regardless of the current queue length*. For a system with 2 VMs this corresponds to using the thresholds $(Q_i, Q_{ib}, Q_{ii} = (0, 0, 0)$ (or equivalently, the system occupancy thresholds $(N_i, N_{ib}, N_{ii}) = (1, 1, 0))$. Such thresholds imply that as long as there is even a single unassigned job in the system and there is an idle server available, we assign the job to the idle server (of course we make use of idle servers that are interference-free over those under interference, when possible). In the subsections that follow, we will be measuring the improvement (with respect to average number in system, or equivalently mean response time) for our optimal policies by comparing them to the non-idling policy.

C. The impact of the arrival rate, λ

We first study the impact of the arrival rate, λ, on both the structure of the optimal policy and on the benefit offered by the optimal policy over the benchmark non-idling policy. In this study, we restrict attention to the parameter settings in *Environment 1*, detailed in Table I. In this environment, there are two VMs alternating between service rates of $\mu_H = 50$ and $\mu_L = 1$, spending equal amounts of time free from and under interference. We allow λ to take any integer value from 1 to 50 inclusive, which in this case corresponds to a range

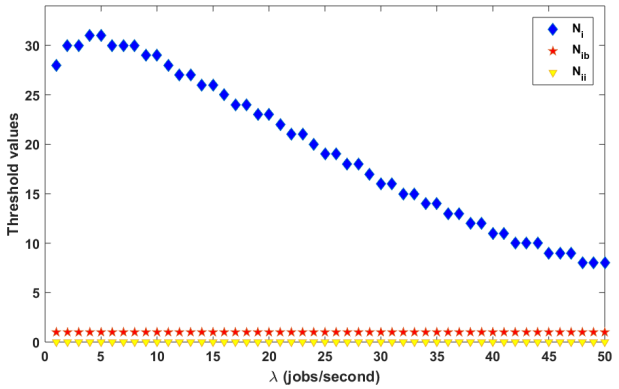


Fig. 3: Optimal thresholds for parameters of *Environment 1* as a function of the arrival rate.

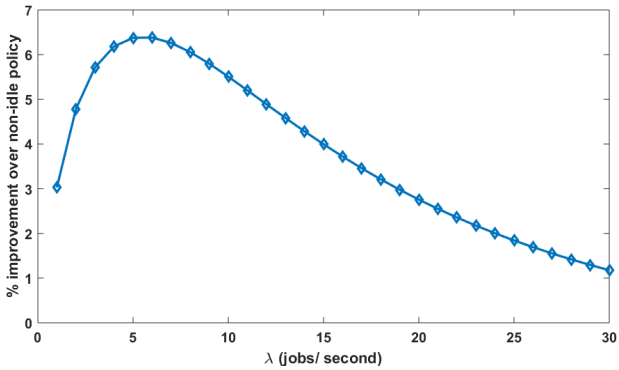


Fig. 4: Improvement in the average system size over the non-idling policy for *Environment 1* parameters.

of offered loads, ρ, from approximately 2–98%.

Fig. 3 shows the thresholds (N_i, N_{ib}, N_{ii}) associated with the optimal policy as a function of the arrival rate, $\lambda \in \{1, 2, 3, \ldots, 50\}$. A cursory examination reveals that the thresholds $N_{ib} = 1$ and $N_{ii} = 0$ for all values of λ examined. Hence, these thresholds attain their minimum possible value, implying that in *Environment 1* it is optimal to assign jobs to idle VMs whenever both VMs are under interference. This result can be explained as follows: the service rate of each VM in this setting is $\mu_L = 1$, while the rate at which one of the two VMs becomes interference-free is $2\alpha_L = 0.6$, so we are better off making use of our idle servers—rather than waiting for one to become interference-free—as it is more likely than we can finish serving a job (even with the speed reduction) before either VM becomes interference-free. On the other hand, in the presence of an interference-free VM, we should be more conservative in making use of a server under interference, as $\mu_F = 50 \gg \mu_S = 1$, so we will often be better off relying on our interference-free VM to eventually serve our entire queue, in the hopes that this server will remain interference-free. Specifically, when exactly one VM is under interference, we will only make use of this VM when the system occupancy is at least N_i, where as λ increases the optimal value of N_i rises to a peak of 31 at $\lambda \in \{4, 5\}$, and steadily drops thereafter.

The most intriguing feature of the results in Fig. 3 is that the optimal threshold value of N_i is highest at an *intermediate* value of λ, rather than at either extreme. This observation suggests the presence of a tradeoff. As λ grows we are simultaneously encouraged to be *more aggressive* with using our slower VM, because the expected waiting time externality imposed on future arrivals due to maintaining a longer queue grows with λ. On the other hand there must be a more subtle counteracting effect where as λ grows, we are encouraged to be *more conservative* with using our slower VM; for higher values of λ this second effect is weaker than the first, hence the eventual decline in the optimal threshold.

λ	μ_H	μ_L	α_H=α_L	k
$\{1, 2, 3, \ldots, 50\}$	50	1	$\{0.3, 3, 30\}$	2

TABLE II: Parameters for *Environment 2*.

The above discussion concerns the structure of the optimal policy in *Environment 1*, but we would also like to know how optimal policies perform in comparison to our non-idling benchmark policy. We address this line of inquiry in Fig. 4, where it is evident that the optimal policy outperforms the benchmark by about 3% at $\lambda = 1$, with this edge increasing to a peak of over 6% when $\lambda = 5$, after which the edge declines down to about 1% at $\lambda = 50$. A modest improvement in light-traffic ($\rho, \lambda \approx 0$) is to be expected because when the arrival rate vanishes, the improvement should also vanish. This is because there will very rarely ever be more than one job in the system when the arrival rare is essentially zero, and thus (under these parameter settings) all policies *perform* like the non-idling policy in this setting; even if the non-idling policy is structurally very different from the non-idling policy, the disagreements between these two policies are occur only at states which seldom arise. On the other side, we also expect the improvement to vanish under heavy traffic ($\rho \approx 1$), as in such a setting a policy that maintains system stability can very rarely afford to be idle, so any such policy—and the optimal policy in particular—will be making the same decisions as the non-idling policy the vast majority of the time.

D. The impact of interference rates, α_H and α_L

Next, we study the impact of the rates at which a VM enters and leaves interference, α_H and α_L, respectively. We initiate this study by focusing on *Environment 2*, with exact parameter details given in Table II. As in the previous environment, here we again assume that $\alpha_H = \alpha_L$, but this time we allow these parameters to take on different values, namely 0.3, 3, and 30. In fact *Environment 1* is a special case of *Environment 2* where $\alpha_H = \alpha_L = 0.3$, as we again consider a 2-VM setting with $\mu_H = 50$ and $\mu_L = 1$, and still vary λ from 1 to 50.
The performance of the optimal threshold policy relative to that of the non-idling benchmark policy is plotted in Fig. 5. The thick dashed curve corresponds to the same curve seen in Fig. 4. Notice that unlike thick dashed curve, the solid and thin dashed curves (corresponding to the more extreme α-values) do not vanish at light traffic, this is because in both of these cases the α-values are sufficiently high enough, that even when one anticipates no future jobs, when one is dealing with two VMs both under interference, it is best to suspend

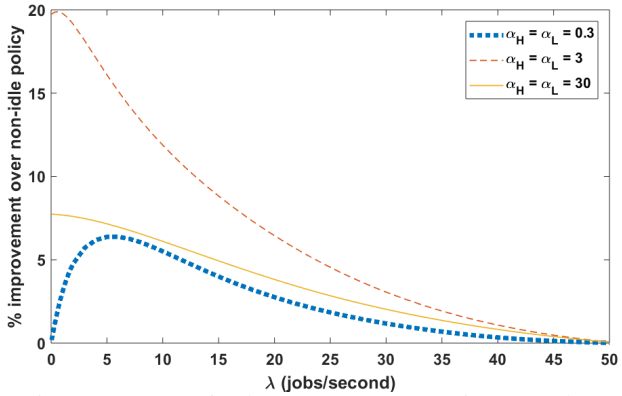


Fig. 5: Improvement in the average system size over the non-idling policy for parameters of *Environment 2* as a function of λ under three different values of $\alpha_H = \alpha_L$.

the decision of where to send the job until one of the VMs leaves interference.

Intriguingly, we find that for all values of λ examined the percentage improvement in the setting where setting with intermediate α-values ($\alpha_H = \alpha_L = 3$) dominates the percentage improvement in the other two cases. To investigate this phenomenon further, we fix the value of λ to 10 and let $\alpha_H = \alpha_L$ vary across a wider spectrum from 0.03–300, giving rise to *Environment 3*, in which $\alpha_H = \alpha_L \in 0.03 \cdot 10^x$, where x is an index parameter taking on values in the set $\{0, 0.1, 0.2, \ldots, 4\}$ (see Table III).

The improvements associated with the optimal threshold policy (again, relative to the non-idling benchmark) are presented as a "lin-log" plot in Fig. 6. From this plot, we see that improvements are modest for values of $\alpha_H = \alpha_L$ on the low and high end of the plotted range, but are considerable for intermediate values in this range, peaking at around 14% when $\alpha_H = \alpha_L \approx 4.75$ (corresponding to $x = 2.2$). Let us first address the lackluster performance improvements when the α-values are very large. In such a setting, each VM is rapidly changing between two speed states, and essentially performs like a VM with service rate $(\mu_H + \mu_L)/2 = 25.5$; since it is not unlikely for a VM to go in and out of interference multiple times while serving the same job, routing decisions become increasingly inconsequential as the α-values grow large. The case of modest improvements under very low α-values is more subtle. To better understand this regime, it is helpful to examine the thresholds, which are plotted in Fig. 7. With low α-values, when both VMs are under interference, we make use of these slow VMs liberally (i.e., $N_{ib} = 1$ and $N_{ii} = 0$), because a VM is not likely to leave interference anytime soon; this is similar to an observation we made regarding optimal policies in *Environment 1*. While the optimal policy is more reserved in making use of a slow VM when one VM is interference-free (as exhibited by the high optimal N_i values), it turns out making suboptimal decisions in such cases are fairly inconsequential, as it typically only hurts the prematurely assigned job, which then occupies the VM under

λ	μ_H	μ_L	$\alpha_H = \alpha_L$	k
10	50	1	0.03×10^x	2

TABLE III: Parameters for *Environment 3*; $x \in \{0, 0.1, 0.2, \ldots, 4\}$.

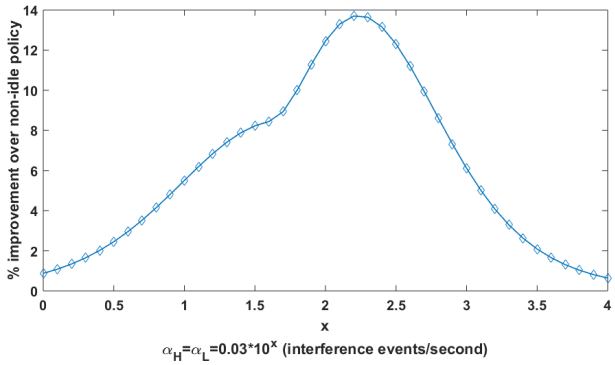


Fig. 6: Improvement in the average system size over the non-idling policy for the parameters of *Environment 3* as a function of the interference rates.

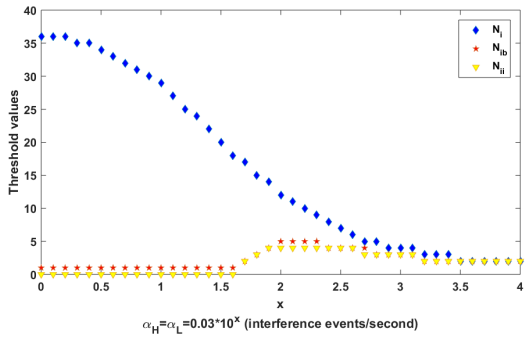


Fig. 7: Optimal thresholds for the parameters of *Environment 3* as a function of the index parameter x where $\alpha_H = \alpha_L = 10^x$.

interference for a long time. In other words, performance is not very sensitive to the choice of N_i in this setting, and the non-idling policy performs quite well. Moreover, we observe that under low α-values the system will be under temporary overload for long periods of time when both VMs are under interference, leading to a very large backlog of jobs that will persist for a substantial duration of time even after one VM comes out of interference, potentially exacerbating the aforementioned issue. See Fig. 8, which demonstrates how the average number in system is very large under low α-values.

We also note that Fig. 7 shows that the optimal N_b and N_i values are nontrivial for higher α-values. This further explains why the curves associated with higher α-values in Fig. 5 do not vanish in light traffic.

E. Additional observations on two-server systems

We experimented with numerous parameter settings to analyze the threshold values of the optimal dispatching policy and its improvement upon the non-idling policy for a two-server system. This analysis yielded several interesting observations, which we present in brief in this section; in the interest of brevity, we omit the full set of results (these are available in the online Appendix [46]).

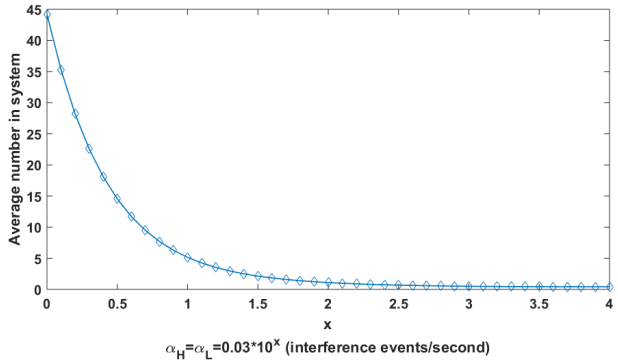


Fig. 8: Average system size for the parameters of *Environment 3* as a function of x where $\alpha_H = \alpha_L = 10^x$.

Parameter set	ρ	μ_H	μ_L	k	α_L	α_H/α_L
1	.25	100	2	2	.01	{.01,.02,.03... 1}
2	.1	100	1	2	1	{.01,.02,.03... 1}
3	.1	100	1	2	10	{.01,.02,.03... 1}
4	.75	100	1	2	10	{.01,.02,.03... 1}

TABLE IV: Parameters for *Environment 4*

By fixing the problem parameters μ_H, μ_L, α_H and α_L and increasing load ρ (essentially increasing λ), we find that the thresholds N_i, N_{ib} and N_{ii} and the improvement percentage with respect to the non-idling policy are non-increasing. Intuitively, as the load increases it would be best to decrease number of idling servers, which moves the optimal threshold toward that of the non-idling policy. This intuition also holds when we fix problem parameters (μ_H, α_H, α_L, ρ) and vary only μ_L — the thresholds are non-increasing with increases in μ_L. Specifically, as μ_L increases, there is a decline in the benefit associated with idling a server under interference and waiting for an interference free server. With the same reasoning we also expect that thresholds are monotonically decreasing with increasing μ_L keeping λ fixed (instead of ρ).

Our experiments show that there does not exist any consistent monotone increasing or decreasing structure of the thresholds when fixing ρ, μ_H, μ_L, α_L and varying α_H. For a better picture, consider *Environment 4* given in Table IV. Fig. 9 plots the threshold values of the optimal dispatching policy with increase in α_H for the different parameter sets of *Environment 4*. We observe that for two parameter sets the threshold values increase monotonically with α_H and for the other two sets the threshold values decrease with α_H.

As stated in Proposition IV.3, we observe in all our numerical examples that $N_i \geq N_{ib} \geq N_{ii}$ and that the difference between N_{ib} and N_{ii} is at most one.

Finally, the biggest takeaway from our examination of two-VM settings is that improvements tend to be greatest when traffic is not extremely heavy, and when interference is neither extremely frequent nor infrequent. It is in these intermediate regimes where the greatest improvement potentials lie. Our experiments show that in these regimes improvements on the order of 5–20% are not atypical.

It should be noted that the settings discussed above are not necessarily in line with those found in practice [23], [24]. In reality, (i) servers only occasionally experience relatively short sojourns of interference (e.g., $\alpha_H/\alpha_L \leq 0.3$), (ii) servers under interference generally function at less than half of their normal speed (e.g., $\mu_L/\mu_H \leq 0.5$), and (iii) interference sojourns are considerably longer than processing times (e.g., $\alpha_L/\mu_H \leq 0.01$). Nevertheless, these realistic settings still frequently allow for the aforementioned improvements that are on the order of 5–20%.

F. Systems with more than two VMs

Just as the optimal policy for two-VM systems is defined by three threshold values, that of a k-VM system is given by $k(k+1)/2$ thresholds. Therefore, the MAM solution technique is computationally inefficient under large VM counts. In many-VM settings, the MDP value-iteration approach is superior as it automatically recognizes different thresholds without prior definitions. That said, even with the MDP approach, the time to convergence grows with the number of VMs.

Our numerical experiments show that for systems with more than two VMs, there is greater potential for improvement as compared to the two-VM case. This is illustrated in Fig. 10. One possible explanation for this phenomenon is that the number of thresholds required to define a policy increases with the VM count, allowing for greater sophisticated decision-making opportunities as compared to the non-idling policy.

VI. Implementation and Experimental Evaluation

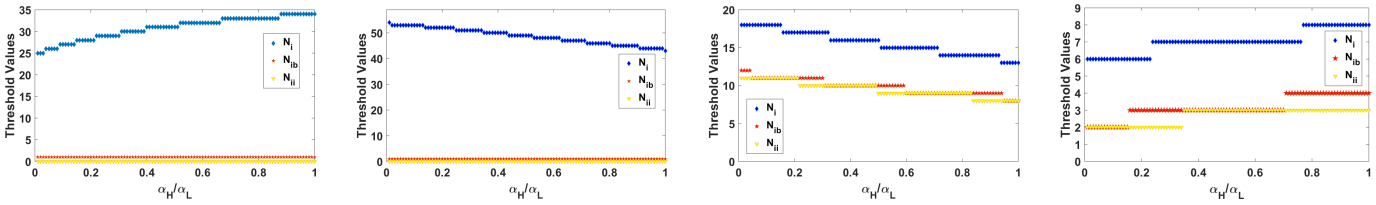
To evaluate our threshold policy in a realistic setting, we consider an experimental testbed with multiple (KVM) VMs hosting a web application and under interference with colocated CPU-intensive microbenchmarks. We implement our threshold policy at the HAProxy [3] load balancer that distributes incoming requests among the web server VMs. The implementation experiments, despite the simplistic workload, allow us to evaluate our policy under realistic processor sharing at the VMs and under realistic interference intensity induced by CPU-intensive workloads.

A. Experimental setup

Testbed: We employ two 1-core VMs as web servers (we use VM and server interchangeably). The VMs run Ubuntu Linux 14.04 and use Apache web server (version 2.4.18) [43]. The web server is configured to serve a web page which encrypts 15 KB of data for each request.

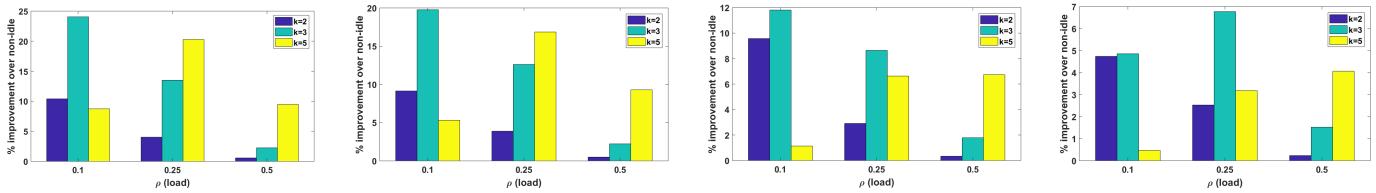
We employ the httperf [35] load generator to drive our workload using exponential inter-arrival times with different means. httperf outputs the mean and tail response times at the end of each experiment, which we report as our performance metrics. Each of our experiments is run for 2 minutes; we report average results based on 3 runs of each experiment.

We use the HAProxy load balancer to distribute incoming web requests between the VMs using the default round robin policy. The HAProxy load balancer uses a maxconn feature, which restricts the number of requests sent to a server to avoid overwhelming the server [5]. In our experiments, a maxconn setting of 5 works well to provide a timely response without overloading the VM CPU. Note that while our MDP model as-



(a) Parameter set 1 (b) Parameter set 2 (c) Parameter set 3 (d) Parameter set 4

Fig. 9: Numerical results showing how threshold values vary as we increase α_H.



(a) $\mu_L = 1$ (b) $\mu_L = 2$ (c) $\mu_L = 10$ (d) $\mu_L = 20$

Fig. 10: Improvement over non-idling policy for $2, 3$, and 5-VM systems; $\mu_H = 100, \alpha_L = 1/10, \alpha_H = 1$, $\mu_L \in \{1, 2, 10, 20\}$.

sumes a first-come-first-serve setting, most real-world servers exhibit a processor sharing or limited processor sharing [36], [52] type of service policy.

Interference: We use the last level cache (LLC) microbenchmark, dcopy [2], to create interference for the VMs. dcopy copies vectors repeatedly to stress the cache; to control the induced cache load, we add a sleep timer to the dcopy microbenchmark. In our experiments, we pin both web server VMs to a physical core on different sockets. We run the dcopy process outside the VMs but on the same core and socket as one of the VMs, thus creating CPU and LLC interference.

B. Implementation of the threshold policy

We implement our threshold policy on the HAProxy load balancer. HAProxy allows us to remotely enable or disable, for load dispatching, any of the back-end VMs. We use this feature to restrict the VMs that serve incoming requests.

HAProxy also provides detailed logging about the number of requests at different stages – accepted, at front-end (load balancer), at back-end (all the VMs), and at a particular server. The maximum number of requests that are simultaneously sent to a server is restricted by the maxconn (= 5) parameter. If the number of requests received by the load balancer is more than *number-of-active-servers* $\times$ *maxconn*, the additional requests are queued at the load balancer; note that requests at the back-end VM are also counted as being on the load balancer, per HAProxy. By periodically (1ms) monitoring the HAProxy logs, we determine the queue length. Note that HAProxy is a scalable load balancer (used by popular, high-traffic websites such as Instagram, Twitter, Alibaba [19]), and hence queue length monitoring at the HAProxy load balancer is an effective and scalable solution.

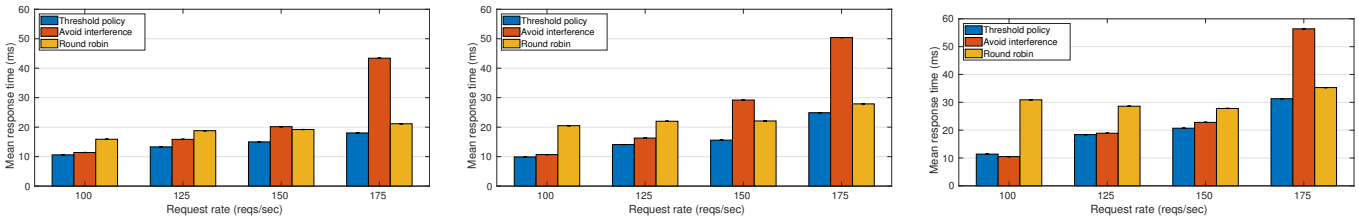
When interference is detected at the VMs, we monitor the queue length to determine when to utilize the VM under interference, as suggested by the threshold of our policy; we use the numerical results for the queue thresholds from Section V to obtain our thresholds. Interference detection has been investigated by prior works [9], [32], [33], and is orthogonal to the focus of this work. For our experiments, we assume that the load balancer is aware of interference.

C. Experimental evaluation results

In our experiments, we set a deterministic $\alpha_L = \alpha_H = 1/30\ s^{-1}$. For deriving the service rates, we stress a single VM under different levels of interference, corresponding to the load induced by the dcopy benchmark, and find the peak throughput. Without interference, we find that $\mu_H = 125$ req/s. Under different levels of interference, we find that $\mu_L = 50$ req/s (low intensity), $\mu_L = 30$ req/s (medium intensity), and $\mu_L = 15$ req/s (high intensity). For this set of results, we only consider 1 VM under interference, and employ the numerically optimal Q_i thresholds given by our MDP model. Note that the thresholds, or even the threshold policy, need not be optimal under our realistic experiments since we have deterministic α_L, α_H, μ_L, and μ_H, in addition to allowing multiple requests to execute simultaneously at a VM. However, we anticipate that the benefits of the threshold policy still hold for real-world settings, as we evaluate next.

Fig. 11 shows the mean response time for the web server application for different request rate and interference intensity settings. The request rate range was chosen to maintain the VM CPU utilization in the realistic 30–80% regime [7], [18]. We compare three policies in all cases:

- The "Threshold policy" employs the theoretically optimal queueing thresholds obtained from our MDP model.
- The "Avoid interference" policy always ignores a VM if it is under interference; this is equivalent to setting a threshold of ∞. Under high interference, such a policy is known to perform well [32].
- The "Round robin" policy is similar to the non-idling policy from Section V, and can be thought of as an interference-oblivious policy.



(a) Low interference intensity. (b) Medium interference intensity. (c) High interference intensity.

Fig. 11: Experimental results showing application mean response time under different policies for various settings.

Across all twelve settings, our threshold policy lowers mean response time by about 24%, on average, compared to the Avoid interference policy, and by as much 58%. Likewise, compared to Round robin, our threshold policy lowers mean response time by about 30%, on average, and by as much 63%.

In general, compared to the Avoid interference policy, our threshold policy's benefits increase with request rate, since the Avoid interference policy utilizes only 1 VM at all loads; the average improvement in mean response time afforded by our threshold policy over the Avoid interference policy increases from 7.7% at 100 req/s to 51.2% at 175 req/s. Further, compared to the Avoid interference policy, our benefits decrease with interference intensity, since it is increasingly better to avoid the under-interference VM at higher intensities; the average improvement in response time afforded by our policy over the Avoid interference policy decreases from 26.8% at low intensity to 16.2% at high intensity.

By contrast, compared to the Round robin policy, our threshold policy's benefits decrease with request rate since at high request rates the optimal threshold is lower, resulting in both VMs being utilized and our policy more closely resembling Round robin; the average improvement in mean response time afforded by our threshold policy over the Round robin policy decreases from 49.4% at 100 req/s to 12.3% at 175 req/s. Further, compared to Round-robin, our benefits increase with interference intensity, since Round robin will suffer more under increased interference; the average improvement afforded by our policy over Round robin increases from 24.8% at low intensity to 33.9% at high intensity.

We also experiment with time-varying request rates, to assess the applicability of our threshold policy for real-world workloads. We use two digitized traces from Facebook [6], appropriately scaled for our setup. The APP trace has a steep rise, followed by a relatively constant request rate, and ends with a steep drop. The ETC trace is similar to APP, but is more bursty. The peak-to-min request rate ratio in both is about 2.

Each experiment (and arrival trace) is run for 2 minutes, and results are reported as averages based on 3 runs. As before, we set a deterministic $\alpha_L = \alpha_H = 1/30\ s^{-1}$. We experiment with medium interference intensity, with $\mu_L = 30$ req/s, and $\mu_H = 125$ req/s. For the APP trace, our threshold policy (using dynamically generated request-rate–specific thresholds) provides a moderate 9% improvement over the Avoid interference policy, but a more substantial 45% improvement over the Round robin policy. For the ETC trace, we see greater benefits, owing to the bursty nature of the trace. Our threshold policy lowers mean response time by 48% and 55% compared to the Avoid interference and Round robin policies, respectively.

The improvement is more pronounced for tail response times. Compared to the Avoid interference policy, our threshold policy reduces the 95%ile response time by about 40% and 73% for the APP and ETC traces, respectively. Compared to the Round robin policy, the corresponding improvements are about 78% and 79%, respectively, for APP and ETC traces.

VII. Conclusion

We considered the dynamic control of jobs in a queueing system with a central queue and a controller in which servers (VMs in the cloud) could temporarily experience interference independent of each other. The interference downgrades the service rate of the servers, and leaves the servers after an exponential time. The question we answered is when to utilize the under-interference servers. We proved the optimality of a threshold control policy based on the number of jobs in the central queue; to minimize the expected total number of jobs in the system, the controller should route a job to an under-interfere server only when the queue size exceeds a threshold. This threshold increases with the number of busy interference-free servers and the number of busy under-interference servers.

Our numerical experiments enable us to extract several insights regarding the nature of optimal policies and the settings with the greatest potential for improvement over the non-idling benchmark policy. We find improvements tend to be greatest under intermediate traffic and interference rates, particularly when traffic is relatively (but not extremely light). Moreover, systems with more than two VMs allow for greater potential improvements. Under relatively light traffic, improvements on the order of 5–20% over the non-idling policy are not atypical.

We also implemented our threshold policy on a testbed with two Web server VMs subjected to real-world interference under different request rates, including time-varying request rates. While our theoretical optimality results need not hold in such complex, real-world scenarios, our experimental results show substantial performance improvement, as much as 40–60%, compared to other baseline scheduling policies.

Acknowledgment

This work was supported by NSF CNS grants 1617046, 1717588, and 1750109.

REFERENCES

[1] A Rare Peek Into The Massive Scale of AWS. https://www.enterprisetech.com/2014/11/14/rare-peek-massive-scale-aws, 2014.

[2] DCOPY (part of BLAS). http://www.netlib.org/blas, 2017.

[3] The Reliable, High Performance TCP/HTTP Load Balancer. http://www.haproxy.org, 2018.

[4] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R. H., KONWINSKI, A., LEE, G., PATTERSON, D. A., RABKIN, A., STOICA, I., AND ZAHARIA, M. Above the clouds: A Berkeley view of cloud computing. Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[5] ASSMANN, B. How to play with maxconn to avoid server slowness or crash. https://www.haproxy.com/blog/play_with_maxconn_avoid_server_slowness_or_crash, 2011.

[6] ATIKOGLU, B., XU, Y., FRACHTENBERG, E., JIANG, S., AND PALECZNY, M. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK, 2012), SIGMETRICS '12, pp. 53–64.

[7] BARROSO, L. A., AND HÖLZLE, U. The Case for Energy-Proportional Computing. *IEEE Computer 40*, 12 (2007), 33–37.

[8] BU, X., RAO, J., AND XU, C.-Z. Interference and Locality-Aware Task Scheduling for MapReduce Applications in Virtual Clusters. In *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing* (New York, NY, USA, 2013), HPDC '13, pp. 227–238.

[9] CASALE, G., RAGUSA, C., AND PARPAS, P. A Feasibility Study of Host-level Contention Detection by Guest Virtual Machines. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on* (2013), vol. 2, IEEE, pp. 152–157.

[10] CASALE, G., AND TRIBASTONE, M. Modelling exogenous variability in cloud deployments. 73 – 82.

[11] CASALE, G., TRIBASTONE, M., AND HARRISON, P. Blending randomness in closed queueing network models. 15–38.

[12] CIANCUTTI, J. 5 Lessons We've Learned Using AWS. http://techblog.netflix.com/2010/12/5-lessons-weve-learned-using-aws.html, 2010.

[13] DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA, 2007), SOSP '07, pp. 205–220.

[14] DELASAY, M., KOLFAL, B., AND INGOLFSSON, A. Maximizing throughput in finite-source parallel queue systems. *European Journal of Operational Research 217*, 3 (2012), 554–559.

[15] DELIMITROU, C., AND KOZYRAKIS, C. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, TX, USA, 2013), ASPLOS '13, pp. 77–88.

[16] EFROSININ, D. Queueing model of a hybrid channel with faster link subject to partial and complete failures. *Annals of Operations Research 202*, 1 (2013), 75–102.

[17] AWS Case Study: Expedia. https://aws.amazon.com/solutions/case-studies/expedia, 2017.

[18] FAN, X., WEBER, W.-D., AND BARROSO, L. A. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture* (San Diego, CA, USA, 2007), ISCA '07, pp. 13–23.

[19] HAPROXY. They use it! https://www.haproxy.org/they-use-it.html, 2019.

[20] HARCHOL-BALTER, M. *Performance modeling and design of computer systems : queueing theory in action*. Cambridge University Press, New York, 2013.

[21] IM, J.-F., GOPALAKRISHNA, K., SUBRAMANIAM, S., SHRIVASTAVA, M., TUMBDE, A., JIANG, X., DAI, J., LEE, S., PAWAR, N., LI, J., AND ARINGUNRAM, R. Pinot: Realtime OLAP for 530 Million Users. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA, 2018), SIGMOD '18, pp. 583–594.

[22] IORGULESCU, C., AZIMI, R., KWON, Y., ELNIKETY, S., SYAMALA, M., NARASAYYA, V., HERODOTOU, H., TOMITA, P., CHEN, A., ZHANG, J., AND WANG, J. PerfIso: Performance Isolation for Commercial Latency-Sensitive Services. In *Proceedings of the 2018 USENIX Annual Technical Conference* (Boston, MA, USA, 2018), pp. 519–532.

[23] JAVADI, A., AND GANDHI, A. DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing. In *Proceedings of the 14th IEEE Internation Conference on Autonomic Computing* (Columbus, OH, USA, 2017), ICAC '17.

[24] JAVADI, A., MEHRA, S., VANGOOR, B., AND GANDHI, A. UIE: User-centric Interference Estimation for Cloud Applications. In *Proceedings of the 2016 IEEE Internation Conference on Cloud Engineering (Work-in-Progress track)* (Berlin, Germany, 2016), IC2E '16.

[25] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. KVM: the Linux Virtual Machine Monitor. In *Proceedings of the 2007 Ottawa Linux Symposium* (2007), pp. 225–230.

[26] KOOLE, G. A Simple Proof of the Optimality of a Threshold Policy in a Two-server Queueing System. *Systems & Control Letters 26*, 5 (1995), 301–303.

[27] LARSEN, R. L. *Control of Multiple Exponential Servers with Application to Computer Systems*. PhD thesis, College Park, MD, USA, 1981.

[28] LATOUCHE, G., AND RAMASWAMI, V. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, Philadelphia, PA, USA, 1999.

[29] LIN, W., AND KUMAR, P. R. Optimal Control of a Queueing System with Two Heterogeneous Servers. *IEEE Transactions on Automatic Control 29*, 8 (1984), 696–703.

[30] LO, D., CHENG, L., GOVINDARAJU, R., RANGANATHAN, P., AND KOZYRAKIS, C. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture* (Portland, OR, USA, 2015), ISCA '15, pp. 450–462.

[31] LUH, H. P., AND VINIOTIS, I. Threshold control policies for heterogeneous server systems. *Mathematical Methods of Operations Research 55*, 1 (2002), 121–142.

[32] MAJI, A., MITRA, S., AND BAGCHI, S. ICE: An Integrated Configuration Engine for Interference Mitigation in Cloud Services. In *Proceedings of the 2015 IEEE International Conference on Autonomic Computing* (Grenoble, France, 2015), pp. 91–100.

[33] MAJI, A. K., MITRA, S., ZHOU, B., BAGCHI, S., AND VERMA, A. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Middleware Conference* (Bordeaux, France, 2014), Middleware '14, pp. 277–288.

[34] MOGHADDAM, F. F., CHERIET, M., AND NGUYEN, K. K. Low Carbon Virtual Private Clouds. In *Proceedings of the 2011 IEEE International Conference on Cloud Computing* (Washington, D.C., USA, 2011), pp. 259–266.

[35] MOSBERGER, D., AND JIN, T. httperf—A Tool for Measuring Web Server Performance. *ACM Sigmetrics: Performance Evaluation Review 26*, 3 (1998), 31–37.

[36] NAIR, J., WIERMAN, A., AND ZWART, B. Tail-robust scheduling via limited processor sharing. *Performance Evaluation 67*, 11 (2010), 978 – 995.

[37] ÖZKAN, E., AND KHAROUFEH, J. P. Optimal Control of a Two-Server Queueing System with Failures. *Probability in the Engineering and Informational Sciences 28*, 4 (2014), 489–527.

[38] PUTERMAN, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[39] REEVE, R., AND NEUMANN, S. Cloud Computing – The IT Solution for the 21st Century. https://www.cdp.net/Documents/archive/2011/Cloud-Computing-The-IT-Solution-for-the-21st-Century.pdf, 2011. Carbon Disclosure Project.

[40] RUBINOVITCH, M. The Slow Server Problem. *Journal of Applied Probability 22*, 1 (1985), 205–213.

[41] RUBINOVITCH, M. The Slow Server Problem: A Queue with Stalling. *Journal of Applied Probability 22*, 4 (1985), 879–892.

[42] RYKOV, V. V., AND EFROSININ, D. V. On the slow server problem. *Automation and Remote Control 70*, 12 (2009), 2013–2023.

[43] THE APACHE SOFTWARE FOUNDATION. The Apache HTTP Server Project. https://httpd.apache.org, 2019.

[44] THERESKA, E., DONNELLY, A., AND NARAYANAN, D. Sierra: practical power-proportionality for data center storage. In *Proceedings of the 6th European Conference on Computer Systems* (Salzburg, Austria, 2011), EuroSys '11, pp. 169–182.

[45] VETHA, S., AND DEVI, K. V. Dynamic resource allocation in cloud using queueing model. *Journal of Industrial Pollution Control 33*, 2 (2017), 1547–1554.

[46] VOTKE, S., JALEEL, J. A., SURESH, A., DELASAY, M., DOROUDI, S., AND GANDHI, A. Online Appendix for the Mascots 2019 publication titled Optimal Markovian Dynamic Control of Interference-Prone Server Farms. https://tr.cs.stonybrook.edu/tr/sbcs-tr-2019-10, 2019.

[47] VOTKE, S., JAVADI, S. A., AND GANDHI, A. Modeling and Analysis of Performance Under Interference in the Cloud. In *Proceedings of the 25th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (2017), pp. 232–243.

[48] WANG, C., URGAONKAR, B., GUPTA, A., CHEN, L. Y., BIRKE, R., AND KESIDIS, G. Effective Capacity Modulation as an Explicit Control Knob for Public Cloud Profitability. In *Proceedings of the 13th IEEE International Conference on Autonomic Computing* (Wurzburg, Germany, 2016).

[49] WHITNEY, J., AND DELFORGE, P. Data Center Efficiency Assessment. https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf, 2014. The Natural Resources Defense Council (NRDC).

[50] XU, C., RAJAMANI, K., FERREIRA, A., FELTER, W., RUBIO, J., AND LI, Y. dCat: Dynamic Cache Management for Efficient, Performance-sensitive Infrastructure-as-a-service. In *Proceedings of the 13th EuroSys Conference* (Porto, Portugal, 2018), EuroSys '18, pp. 14:1–14:13.

[51] XU, Y., BAILEY, M., NOBLE, B., AND JAHANIAN, F. Small is Better: Avoiding Latency Traps in Virtualized Data Centers. In *Proceedings of the 4th Annual Symposium on Cloud Computing* (Santa Clara, CA, USA, 2013), SOCC '13.

[52] ZHANG, J., AND ZWART, B. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems 60*, 3 (2008), 227–246.