Modeling and Analysis of Performance under Interference in the Cloud

Scott Votke, Seyyed Ahmad Javadi, Anshul Gandhi
{*svotke, seyyedahmad.javadi, anshul.gandhi*}*@stonybrook.edu*
Stony Brook University

***Abstract*—One of the key performance challenges in cloud computing is the problem of interference, or resource contention, among colocated VMs. While prior work has empirically analyzed interference for specific workloads under specific settings, there is a need for a generic approach to estimate application performance under any interference condition.**

In this paper, we present an analytical model to estimate performance as a function of various workload, system, and interference conditions, including the intensity and length of interference, for single- and multi-VM systems. Comparisons with empirical results under various scenarios show that our model can provide accurate latency estimations (less than 5% error). We employ our model to analyze systems under interference, and derive useful results to aid practitioners.

1. Introduction

The cloud helps lower resource costs, provides virtually unlimited and elastic resources, and enables a geo-distributed presence, all of which are important for the success of online services. Thus, many online services [1], [2] are now provided by cloud-deployed Virtual Machines (VMs).

However, despite its many benefits, cloud computing has its shortcomings. From the perspective of a latency-sensitive user, one of the most significant drawbacks of the cloud is *interference*. Performance interference is caused by contention for physical resources, such as CPU or network, among colocated VM users. Prior studies have shown that application performance on VMs hosted on public and private clouds can degrade by *as much as 27*× [3], [4], [5] due to interference. Our own experiments, shown in Figure 1, for a web server hosted on a private cloud under contention, highlight the significant increase in application response time under interference.

Prior works on interference analysis typically study application performance under fixed workload and system conditions and specific interference scenarios, as we discuss in Section 6. However, applications often experience different loads. Likewise, the service requirements of an application can change over time due to, for example, upgrades and updates to the application. Also, the size of the system, or number of VMs employed, can change over time due to the popularity of the application. Worse, interference itself can occur with varying degrees of intensity and for different durations. Thus, the lessons learnt from analyzing interference under specific workload and interference conditions might not hold true for other conditions.

As a concrete example, if resource contention for CPU exists in a cloud environment under low application load, it might result in negligible performance impact; on the other hand, the same resource contention, but under heavy application load, might result in significant performance degradation. The increase in performance degradation as the request rate increases in Figure 1 illustrates this problem. Likewise, if interference only lasts for a very small time period, it might not result in a noticeable performance impact. However, the same interference can significantly affect performance if it lasts for a longer time, as illustrated by the different lines in Figure 1.

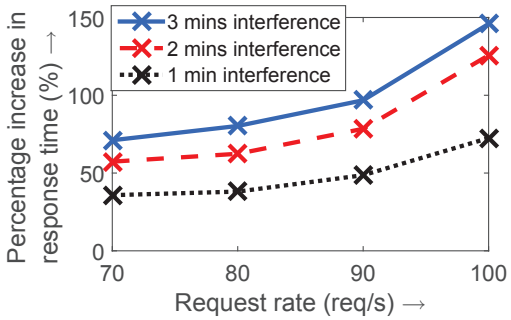
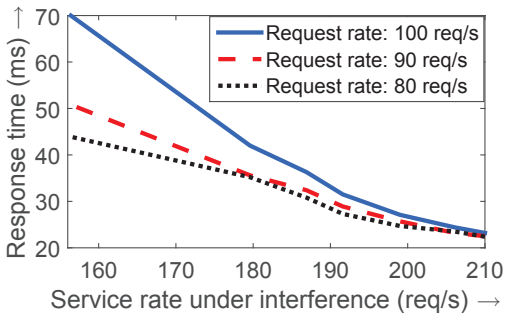


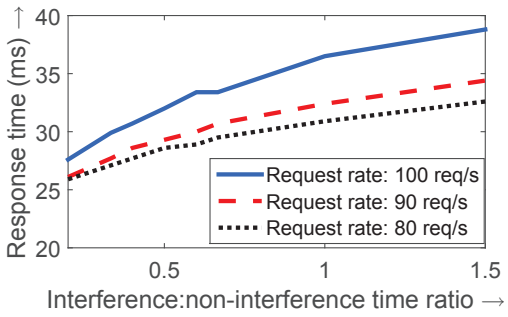
Figure 1. Empirical results for increase in response time at a web server under CPU interference as a function of the workload request rate (x-axis) for different interference lengths; here, the length of the non-interference phase is 5 minutes.

The central problem we consider in this paper is how to model application performance under interference, especially as a function of the workload and interference conditions. A key challenge in this modeling goal is that *performance under non-interference is not independent of performance under interference*. For example, during severe interference, a backlog of unserviced requests is generated, which can continue to impact performance even after interference ceases. Thus, a simple average of performance under interference and performance under non-interference cannot capture such dependencies, and would be inaccurate. Since interference is a dynamic and transient phenomenon [6], our model must track the backlog of requests to estimate the overall performance.

In this paper, we *develop a stochastic model to estimate the impact of interference on application performance as a function of the workload and interference*. We specifically focus on transactional workloads, such as online web applications, that are often hosted on the cloud [1], [2]. We start by determining workload and interference characteristics that affect application performance under interference (Section 2). We then present a performance model that takes these characteristics into account to estimate application response time (Section 3). We validate our model by comparing against empirical results from an OpenStack-deployed web server under interference; our model estimates response



(a) Impact of service rate (or peak throughput) under interference.



(b) Impact of ratio of time spent in interference to time spent in non-interference.

Figure 2. Empirical results for response time as a function of different characteristics that define interference: (a) the service rate under interference, and (b) the ratio of time spent in interference to time spent in non-interference. Workload characteristics, such as request rate, continue to affect response time.

time with *less than 5% error* across various workload and interference conditions including different workload request rates, different interference lengths, and different interference intensities (Section 4).

Using our model, we analyze the impact of workload and interference parameters on the performance of single- and multi-VM applications (Section 5). Our analysis is complementary to existing efforts that focus on interference detection and mitigation [7], [8]. For example, we use our model in Section 5.2 to evaluate the performance of various scheduling policies for a cluster of VMs under interference and assess when each policy is effective. Likewise, our model can help identify interference parameters that have a significant impact on system backlog. Thus, a model that can estimate performance as a function of workload and interference conditions can be valuable to performance management efforts as well.

To summarize, this paper makes the following contributions:

- We examine the impact of workload and interference characteristics on application performance.
- Based on this study, we develop an analytic model that estimates performance under interference. Validation against empirical results show that our model can accurately predict performance under different scenarios.
- Using our model, we analyze scheduling policies for a multi-VM system under interference; our findings can help guide the design of an interference-aware scheduler for cloud-deployed applications.

2. Workload and Interference Characteristics That Affect Application Performance

In order to build an interference-aware performance model, we first study the various factors that impact application performance under interference in the cloud. We then present our model, in Section 3, that leverages these findings to estimate performance.

2.1. Characterizing the workload

Workload characterization is a topic of research in itself. For our purposes, we focus on high-level workload characteristics that dictate the load of transactional services, such as web applications. We borrow concepts from performance modeling [9], [10] and characterize the workload using two parameters: (i) the *average workload request rate*, and (ii) the *average service time*. The former is simply the mean arrival rate of requests at the VM, in units of req/s. The latter is the mean time to complete a request, in seconds, under negligible load; the inverse of average service time is referred to as average service rate, and represents the peak throughput of the VM.

The product of the average request rate and average service time, which represents the rate of work coming into the VM, can be used as a proxy for the system load [11]. Note that an increase in value of either parameter increases the system load. Both parameters described above assume no interference; the next subsection deals with parameters under interference.

2.2. Characterizing interference

Interference is caused by contention among colocated VMs for shared physical resources (such as CPU, network, and cache [12], [13]) on the underlying host. Thus, to characterize interference, we consider the *dominant* resource under contention, which is the resource at the VM under interference that is closest to saturation. The dominant resource can be identified either via resource usage monitoring (at the VM) or via existing source-of-interference detection techniques such as CRE [14] or CPI^2 [15].

2.2.1. Service rate under interference

We again focus on high-level characteristics that help define the impact of interference on application performance. The first characteristic we consider is the *intensity* of interference or resource contention; quantitatively, we define this as the *reduced service rate* of the VM under interference, which is also the peak system throughput under interference. Similar parameters have also been used to define interference in prior works, such as the "pressure" term in Cuanta [16] and Bubble-Up [17].

Figure 2(a) shows the mean response time of our cloud-deployed web server as a function of the service rate under interference (see Section 4.1 for details of our experimental setup). We create different interference levels here by generating CPU load on colocated VMs. We see that, from right

to left, response time increases consistently as service rate under interference decreases.

Workload characteristics, such as request rate, continue to affect mean response time under interference, but their impact is amplified. In Figure 2(a), for example, response time goes up, for a given service rate under interference, as the request rate goes up. However, when interference is low (right of the graph), the difference in response time is negligible. From a queueing-theoretic perspective, interference reduces the service rate of the VM. Thus, for the same request rate, response time will be (non-linearly) higher under interference than under non-interference since the VM's service capacity has decreased.

2.2.2. Interference length

The second parameter we consider is the duration of time for which interference lasts. If we assume that interference is periodic, we can also consider the ratio of the time spent by the VM under interference to that under non-interference.

Figure 2(b) shows the mean response time of our cloud-deployed web server as a function of the ratio of interference to non-interference length. We see that response time increases, somewhat sub-linearly, as this ratio increases.

Again, the impact of workload characteristics under interference is amplified. In Figure 2(b), we see that, for a given ratio, response time is higher for higher request rates. However, when interference is low (left of the graph), the difference in response time is almost negligible.

3. Analytic Model for Estimating Application Performance Under Interference

We now present out analytic model for estimating performance under interference. We will make use of all parameters introduced in Section 2 to define our model.

3.1. High-level idea

Our basic approach is to model the system as a Markov chain, making the required Markovian assumptions. The advantage of a Markov chain is that it can track the backlog in the system as the VM (or server, used interchangeably) goes in and out of interference; this is important since interference is transient [6]. The resulting Markov chains are complex, especially for the multi-server system case, as we show in Section 3.4. We thus use Matrix Analytic Methods [18] to solve the resulting Markov chains for the distribution of number of requests in the system; this then gives us mean response time under interference.

3.2. Model setup

We consider a system with k VMs and an average request rate of λ req/s into the system. We assume that the k servers are homogeneous and belong to the same tier, with a central scheduler responsible for dispatching incoming requests to servers (we discuss this further in Section 3.4). Denote the average service rate under no interference by μ_H (H here refers to "high") and the average service rate under interference by μ_L (L refers to "low"); we can, of course, have multiple service rates under interference depending on the intensity of interference, in which case we can think of μ_L as a vector. We consider interference to be periodic, with average interference length denoted by $1/\alpha_L$ (in seconds) and average non-interference length denoted by $1/\alpha_H$ (in seconds). Thus, the rate of leaving the interference phase is α_L and that of leaving the non-interference phase is α_H, *for each server*. That is, we consider the interference at each VM to be an independent process; this makes sense since VMs are often distributed in the cloud and need not be on the same physical host, especially for large public clouds, such as AWS and Azure, that have thousands of physical hosts [19], [20]. The variables used in our analysis are summarized in Table 1.

Variable	Meaning
k	Number of VMs
λ	Request rate (req/s)
μ_H	Service rate under no interference (req/s)
μ_L	Service rate under interference (req/s)
$1/\alpha_H$	Length of non-interference phase (s)
$1/\alpha_L$	Length of interference phase (s)

TABLE 1. DESCRIPTION OF THE VARIABLES USED IN OUR MODEL.

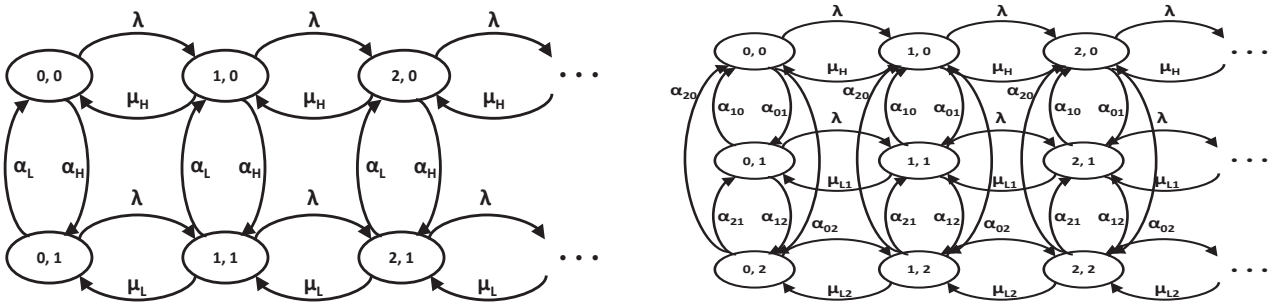
Based on the above notation, we can define system load [11] as the ratio of average request rate to average system service rate: $\lambda / \left(k \cdot \frac{\mu_H/\alpha_H + \mu_L/\alpha_L}{1/\alpha_H + 1/\alpha_L}\right)$. For stability, we assume the load is less than one.

We use Markov chains to track performance as the system moves between interference and non-interference. To this end, we assume that the inter-arrival time and all service times are exponentially distributed, as is common practice when employing Markov chains [11], [18]. Further, we assume that the time spent in the interference phase and the non-interference phase is exponentially distributed. We refer to the resulting Markov chains as M/M/k/int chains. While we require these assumptions for tractability, we show, in Section 4, that our model estimates are fairly accurate even when such assumptions do not hold.

3.3. M/M/1/int: Markov chain for a single-server system under interference

For a single-server system, our Markov chain tracks the number of requests in the system, i, and the number of servers under interference, j; note that j is either 0 or 1. Requests at the server can either be served in a first-come-first-server manner or via processor sharing. In case of the former, the remaining $(i-1)$ requests wait in a queue. The resulting Markov chain, with states (i, j), is shown in Figure 3(a) for the case of a single interference intensity resulting in service rate of μ_L under interference. Here, the rate of going from (i, j) to $(i+1, j)$ is λ, and the rate of going from $(i+1, j)$ to (i, j) is μ_L or μ_H depending on whether the system is under interference ($j = 1$) or not ($j = 0$). Finally, the rate of going from non-interference to interference, or $(i, 0)$ to $(i, 1)$, is α_H, and that of going from interference to non-interference, or $(i, 1)$ to $(i, 0)$, is α_L. Note that the chain (the structure of the states and the transitions in and out of states) repeats itself indefinitely.

We can extend this model to consider different intensities of interference, as shown in Figure 3(b), where μ_{L1} and



(a) M/M/1 with 1 level of interference.

(b) M/M/1 with 2 levels of interference.

Figure 3. Markov chain for an M/M/1 with (a) 1 level of interference, and (b) 2 levels of interference. The state space is denoted by (i, j), where i is the number of requests at the system and j is the level of interference; $j = 0$ indicates no interference.

μ_{L2} are two different service rates under interference corresponding to different interference intensities. Note that we are using j in this specific example to refer to the intensity or level of interference, and not the number of servers under interference. There are different rates of entering and leaving these different interference intensities, along with the possibility of going between them. Thus, the model can be extended to handle an arbitrary range of interferences, albeit at the cost of increased model complexity. For simplicity, we only consider the case of a single interference.

3.4. M/M/k/int: Markov chain for a multi-server system under interference

The multi-server case is complicated by the fact that we now have more than 1 server (out of k) that can be under interference. One might think that the Markov chain in Figure 3(a) can be extended down to multiple rows to account for the multiple servers under interference, but this is not completely true. There are two challenges that must be addressed before we can model a multi-server system faithfully using Markov chains. First, *how can we track the number of requests that are at interference servers versus not-interference servers?* For example, consider the case where we have 2 requests at the system ($i = 2$) and $k = 3$. In this case, if only one server is under interference ($j = 1$), then the 2 requests can either both be at the 2 non-interference servers, or one each at an interference and non-interference server, respectively. Thus, the state $(i = 2,\ j = 1)$ does not provide complete information about the system anymore for the multi-server case. Second, *how does the system decide which VM (interference versus non-interference) to send the next incoming request to?* For example, if we are in state $(i = 0,\ j = 1)$ and $k = 2$, then the scheduler, or load balancer, can send the request to the $(j = 1)$ interference server or to the non-interference server. While one might think the scheduler should prefer non-interference VMs, keep in mind that the scheduler might not be aware of the state of the VMs, especially in a public cloud where users are not aware of the VM to host mapping, and thus cannot predict interference [7], [12], [21]. Thus, in the above $(i = 0,\ j = 1)$ example, an interference-aware scheduler would pick the non-interference VM for the incoming request, whereas an interference-oblivious scheduler would have a $1/2$ probability of picking either server.

3.4.1. 3-dimensional state space

To address the first issue, we consider a 3-dimensional state, (i, l, j), where i and j are the number of requests in the system and number of servers (out of k) under interference, respectively, similar to the M/M/1/int, and l is the number of requests at interference servers. Thus, $l \leq i$ and $l \leq j$; here, we assume that each server only serves one request at a time and unassigned requests wait in a queue. Using this state space, we can now address the $k = 3$ and $(i = 2,\ j = 1)$ example above and define two states, $(2, 0, 1)$ and $(2, 1, 1)$, to represent the cases where both jobs are at non-interference servers or one job each is at a non-interference and interference server, respectively. Note that l does not always start from 0. For example, if $j = k$, meaning that all servers are under interference, then $l = \min(i, k)$.

3.4.2. Scheduler-aware model

To address the second issue, we explicitly consider the scheduling policy employed by the application in our model. We consider three different representative scheduling policies for multi-VM systems under interference:

1) *RandomSched:* This is an interference-oblivious policy that does not know which VMs are under interference. Thus, of the available (free) y VMs, if x are under interference, the probability of routing an incoming request to an interference VM is x/y.
2) *SmartSched:* This is an interference-aware policy, inspired by recent scheduler designs (e.g., Quasar [8] and ICE [7]), that knows which servers are under contention. Each incoming request is assigned to an available non-interference VM, if it exists, else to an available interference VM. If no VMs are available, the request is queued.
3) *OptSched:* This is an unrealistic but near-optimal policy that knows which servers are under interference and can instantaneously migrate, without any penalty, requests from interference servers to available non-interference servers. Thus, if a non-interference VM becomes available after serving a request, an existing request at an interference VM is immediately migrated to it.

M/M/k/int under RandomSched:

Figure 4 shows the 2-dimensional Markov chain for the multi-server system under interference employing the random scheduler. The state space is denoted by (i, l, j), where i, l, and j are as defined in Section 3.4.1. $q_{i,l,j} = \frac{j-l}{k-i}$ and

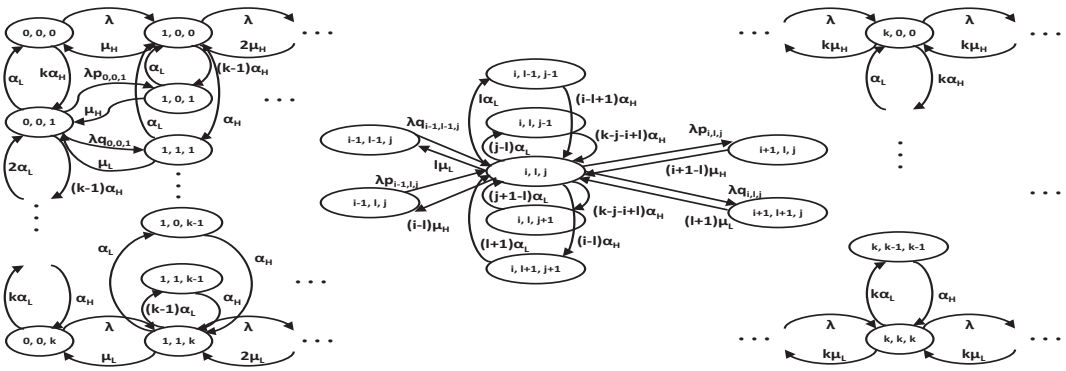


Figure 4. Markov chain for an M/M/k/int under the random and smart schedulers. The state space is denoted by (i, l, j), where i is the number of requests at the system, j is the number of servers under interference, and l is the number of requests at interference servers. For simplicity, in the figure, we only show some states; however, we show all transitions into and out of states on the boundary and the generic state (i, l, j). The chain repeats for $i \geq k$.

$p_{i,l,j} = 1 - q_{i,l,j}$, for $i < k$, are the probabilities that an incoming request is assigned to an available interference server and non-interference server, respectively. The chain repeats for $i \geq k$. In the non-repeating portion ($i \leq k$), we arrange all (i, l, j) states for a given i and j together vertically, enabling a 2-dimensional chain despite the 3-dimensional state space. Note that while the number of rows in the repeating portion of the chain is $(k+1)$, the number of rows in each column of the non-repeating portion is not the same. The 1st column has $(k+1)$ rows, but the 2nd column has $1 \cdot 2 + 2 \cdot (k-1) = 2k$ rows, since each of the $(k-1)$ pairs of $(i = 1,\ 1 < j < k)$ have two possibilities for l, 0 or 1. Likewise, the 3rd column has $1 \cdot 2 + 2 \cdot 2 + 3 \cdot (k-3) = 3k-3$ rows. Note that, in the non-repeating portion, the structure of the Markov chain is symmetric around the mid-point in the horizontal and vertical directions.

For transitions, consider the generic (i, l, j) state shown at the center of Figure 4. An incoming request to this state will be routed either to an available non-interference server with rate $\lambda p_{i,l,j}$, thus transitioning to $(i+1, l, j)$, or to an available interference server with rate $\lambda q_{i,l,j}$, thus transitioning to $(i+1, l+1, j)$. Of the i requests in the system at (i, l, j), l of them are at interference servers, thus departing with rate $l\mu_L$ to state $(i-1, l-1, j)$, and $(i-l)$ of them are at non-interference servers, thus departing with rate $(i-l)\mu_H$ to state $(i-1, l, j)$. Now, in terms of interference striking any of the $(k-j)$ non-interference servers, we consider the two cases where the non-interference server was either serving a request or not. In case of the former, there are $(i-l)$ such servers, assuming $(i-l) \leq (k-j)$, and if interference strikes such a server, state (i, l, j) will transition to $(i, l+1, j+1)$ with rate $(i-l)\alpha_H$ since the request at this server will now contribute to l. In case of the latter, the remaining $(k-j-(i-l))$ non-interference servers will transition to $(i, l, j+1)$, each with rate α_H, for a total rate of $(k-j-i+l)\alpha_H$. Finally, interference at any of the j servers can cease, resulting in a transition to $(i, l, j-1)$ with rate $(j-l)\alpha_L$ if the server is free, or a transition to $(i, l-1, j-1)$ with rate $l\alpha_L$ if the server is serving a request (since the request at that server will no longer contribute to l). Similarly, we can define the rates into (i, l, j).

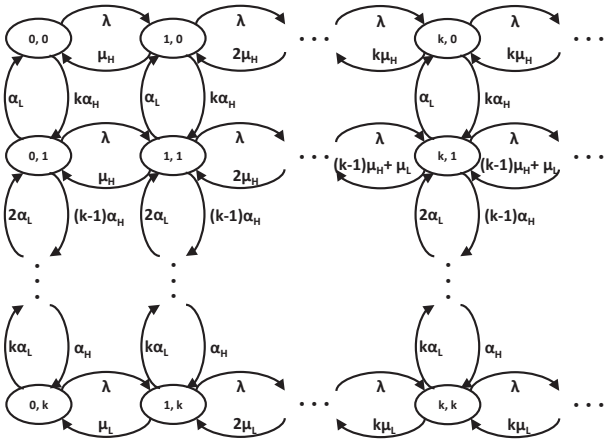
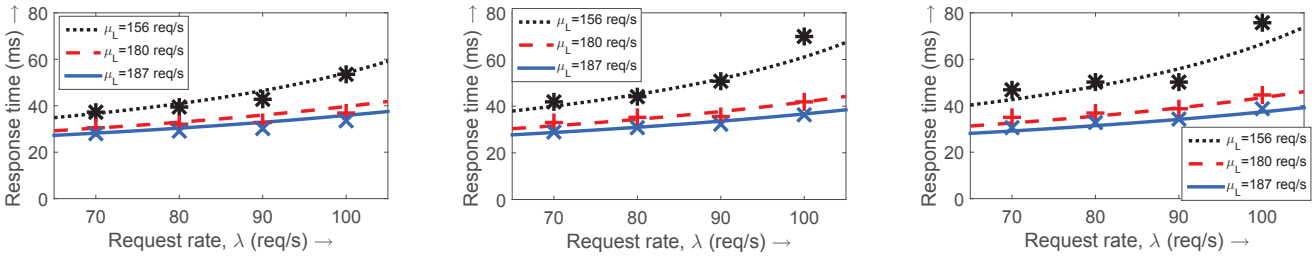


Figure 5. Markov chain for an M/M/k with interference under the optimal scheduler (with instantaneous migration). The state space is denoted by (i, j), where i is the number of requests at the system and j is the number of servers (out of k) that are under interference.

M/M/k/int under SmartSched:
The Markov chain for the M/M/k/int under SmartSched is exactly the same as that under RandomSched, as shown in Figure 4, except for the $p_{i,l,j}$ and $q_{i,l,j}$ probabilities, which can only take on values 0 or 1 under SmartSched. Specifically, $p_{i,l,j} = 1$ if $(i-l) < (k-j)$, and 0 otherwise; $q_{i,l,j} = 1 - p_{i,l,j}$. However, we still require the same number of states, even though some of them will not have any λ transitions into them. For example, consider the state $(1, 1, 1)$ under $k = 2$ (top left of Figure 4). There is no λ transition into this state for SmartSched since $q_{0,0,1} = 0$ as $i < (k-j)$ for $(0, 0, 1)$. Yet, we can get to $(1, 1, 1)$ from $(1, 0, 0)$ with rate α_H, from $(1, 1, 2)$ with rate α_L, and from $(2, 1, 1)$ with rate μ_H.

M/M/k/int under OptSched:
Figure 5 shows the 2-dimensional Markov chain for the multi-server system under interference employing the opt scheduler. The state space is (i, j), where i is the number of requests at the system and j is the number of servers (out of k) that are under interference. Fortunately, in this case, we do not need to track l, the number of requests at non-interference servers, since, under OptSched,



(a) No interference for $1/\alpha_H = 5$ minutes followed by interference for $1/\alpha_L = 3$ minutes.

(b) No interference for $1/\alpha_H = 3$ minutes followed by interference for $1/\alpha_L = 3$ minutes.

(c) No interference for $1/\alpha_H = 2$ minutes followed by interference for $1/\alpha_L = 3$ minutes.

Figure 6. Model validation results: empirical (markers) versus theoretical (lines) response times as a function of request rate (λ) for various interference service rates (μ_L) with an interference phase length (α_L^{-1}) of 3 minutes and non-interference phase length (α_H^{-1}) of (a) 5 minutes, (b) 3 minutes, and (c) 2 minutes. Modeling error ranges from 1.21% to 9.19% with average modeling error of 4.58%.

$l = \min(j, i-(k-j))$ if $i < (k-j)$, and $l = 0$ otherwise. This is because, at all times, requests are first routed/migrated to the $(k-j)$ non-interference servers, and the remaining requests, if any, are sent to the j interference servers. We thus drop l from the state space and denote a state as (i, j), similar to M/M/1/int, resulting in the Markov chain shown in Figure 5; the chain repeats for $i \geq k$. Here, the different rows illustrate the $(k+1)$ different values that j can take, from 0 to k, representing the number of servers under interference. In terms of transitions, for a generic (i, j) in the non-repeating portion ($i < k$), the rate to $(i-1, j)$ is $i\mu_H$ if $i < (k-j)$, and $(k-j)\mu_H + (i-(k-j))\mu_L$ otherwise. With rate $j\alpha_L$, we transition up to $(i, j-1)$ and with rate $(k-j)\alpha_H$, we transition down to $(i, j+1)$. Finally, with rate λ, we transition forward to $(i+1, j)$. For the repeating portion ($i \geq k$), note that it is the same as that for M/M/k/int under OptSched since, in the repeating portion, all new requests queue up and do not have to be immediately assigned to a server.

3.5. Solving the Markov chains

Given the complexity of the Markov chains, and the fact that we have transitions between several states, we resort to Matrix Analytic Methods to numerically solve the Markov chains and derive the limiting probabilities of being in each state; for more details, we refer the reader to [18]. After the probabilities of each state, s, referred to as π_s, have been derived, we then compute the probability of having x requests in the system, π_x. Based on the Markov chain in question, we have $\pi_x = \sum_{i=x} \pi_{(i,l,j)}$ (for M/M/k/int under RandomSched and SmartSched) or $\pi_x = \sum_{i=x} \pi_{(i,j)}$ (for M/M/1/int and M/M/k/int under OptSched). Note that the probabilities for states in the repeating portion are related to the probabilities of the states in the finite non-repeating portion in a simple manner, allowing infinite sums to collapse into simple expressions, as is typically the case under Matrix Analytic Methods [18]. From the π_x, we derive the mean number of requests in the system, $E[N] = \sum_x x \cdot \pi_x$; the mean response time is then derived via Little's Law [22] as $E[T] = E[N]/\lambda$. Note that π_x is the distribution of number of requests in the system, from which the distribution of the backlog, or queued requests, can also be obtained.

4. Model Validation

We now validate our analytical model by comparing its response time estimates with empirical measurements obtained from an experimental testbed. We also validated our model by comparing against known results for the M/M/k [11] when $\mu_H = \mu_L$, but we only discuss empirical validation results here.

4.1. Experimental setup

We set up an experimental testbed in an OpenStack private cloud with a web server VM under CPU contention, driven by a client VM. In particular, our setup consists of two physical hosts (Dell C6100), each with two 6-core CPUs and 48 GB memory, which are part of a larger OpenStack private cloud setup. The hosts are connected to a network switch via a 1Gb Ethernet cable. On the first host, which is our target host, we launch 4 VMs, each with 4 vCPUs and 4GB memory. Note that the total vCPU request is 16, which exceeds the core count of 12 on the physical host, thus resulting in CPU oversubscription (overcommit is enabled by default in OpenStack). One of the 4 VMs acts as the web server, and the remaining three colocated VMs create CPU contention. On the second host, we run a client VM which sends requests to the web VM on the first host.

The web server VM on the first host uses the Apache HTTP server (version 2.4) [23], along with PHP 5.5, to host a CPU intensive php script. The remaining three colocated VMs on this host employ the stress-ng tool [24] to create controllable CPU contention. The client server VM uses httperf [25] as the load generator to issue requests for the php script at a controllable rate.

In our experiments, the inter-request time and php service time distributions are deterministic. We create periodic contention intervals such that the length of the contention and non-contention intervals are deterministic with different lengths. Note that this is different from the Markovian assumptions in our model. In each experiment, we set a request rate, and create alternating phases of contention and non-contention. We log the response time of each request, and compute the mean response time for each experiment over its entire duration. Finally, to derive the service rate under interference and non-interference, we run a separate experiment where we stress the web server with increasing

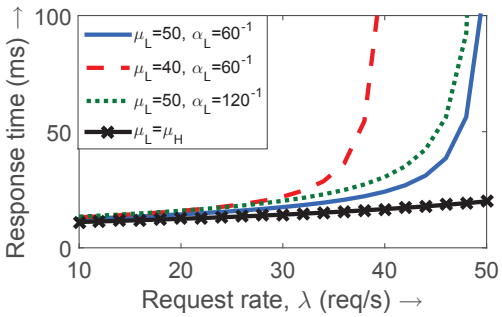


Figure 7. Response time for the M/M/1/int as a function of the request rate (λ) for various interference scenarios (μ_L and α_L). Here, $\mu_H = 100$ req/s and $\alpha_H^{-1} = 10$ minutes. Also shown, for comparison, is the response time of the M/M/1 without interference ($\mu_L = \mu_H$).

request rates, and record the peak throughput at which the VM saturates. Using this methodology, we find that our service rate under non-interference (μ_H) is about 220 req/s.

4.2. Comparing empirical and model results

To validate our model, we run several experiments with different values of request rate (λ), length of non-interference phase (α_H^{-1}), length of interference phase (α_L^{-1}), and contention levels (that result in different μ_L values).

Figure 6 shows our validation results, where we compare the model-predicted values (shown as lines) with empirical measurements (shown as markers). For each figure, the interference phase length is set to 3 minutes, and the request rate is varied for different interference levels. We vary the non-interference phase length between the figures from 5 minutes to 2 minutes. Since we have 4-vCPU VMs, in our model we consider the request rate and service rate at each vCPU to be one-fourth of the total request and service rate.

We see that the model-predicted mean response time increases with (i) an increase in request rate, or (ii) an increase in interference level (or a decrease in service rate under interference), or (iii) a decrease in the length of the non-interference phase. Throughout, the empirical measurements of response time are in close agreement with the model estimates. The average modeling error across all experiments is 4.58%, with per-experiment error ranging from 1.21% to 9.19%.

5. Analysis and Evaluation

In this section, we employ our model to analyze and evaluate the application performance of single-server and multi-server systems under interference. Where applicable, we highlight important observations and derive helpful rules-of-thumb to aid practitioners.

5.1. Analyzing a single VM under interference

It is instructive to first analyze the performance under interference for a single VM before considering multiple VMs. Further, most of the prior work on interference focuses on a single VM [15], [16], [17], thus motivating this subsection.

5.1.1. Impact of interference intensity

Figure 7 shows response time as a function of request rate for different interference scenarios with $\mu_H = 100$ req/s and $\alpha_H^{-1} = 10$ minutes. For comparison, we also show the response time under the M/M/1 without interference (black line with markers). Starting with the solid blue line which denotes the case of 1-minute long interference phases and a 50% reduction in service rate, we see that interference can significantly increase response time, especially under moderate to high request rates, when compared to the M/M/1 without interference. In terms of load (see Section 3.2), the 10–50 req/s corresponds to roughly 10% – 50% load. Response time worsens if we increase the length of the interference phase (green dotted line) or the intensity of interference (dashed red line). Importantly, we observe that:

Observation 1. Response time is much more sensitive to the intensity of interference than the length of interference.

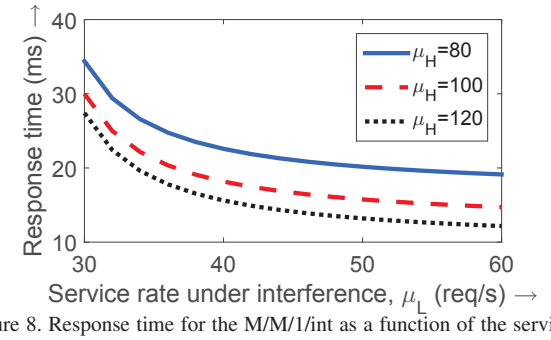


Figure 8. Response time for the M/M/1/int as a function of the service rate under interference (μ_L) for various non-interference service rates (μ_H) with $\lambda = 25$ req/s, $\alpha_H^{-1} = 10$ minutes, and $\alpha_L^{-1} = 1$ minute.

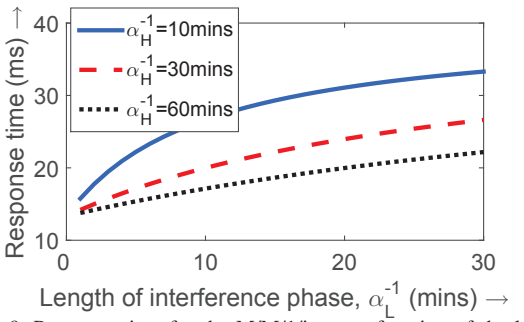


Figure 9. Response time for the M/M/1/int as a function of the length of the interference phase (α_L^{-1}) for various non-interference phase lengths (α_H^{-1}) with $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 50$ req/s.

Figure 8 looks more closely at the effect of interference and non-interference service rates on response time. Here, $\lambda = 25$ req/s, $\alpha_H^{-1} = 10$ minutes, and $\alpha_L^{-1} = 1$ minute. We see that response time is very sensitive to the intensity (service rate) under interference. Further, response time quickly increases with either a decrease in the service rate under interference (μ_L) or a decrease in the non-interference service rate (μ_H).

Observation 2. Response time increases super-linearly with a decrease in service rate under interference.

5.1.2. Impact of interference & non-interference lengths

Figure 9 looks at the effect of interference and non-interference phase lengths. Here, $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 50$ req/s. While response time increases with the length of the interference phase and decreases

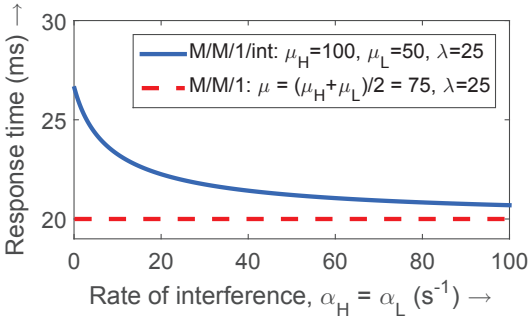


Figure 10. Response time for the M/M/1/int as a function of interference rate ($\alpha_H = \alpha_L$) with $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 50$ req/s. Also shown (dashed line) is the response time of an M/M/1 without interference with service rate equal to average service rate of the M/M/1/int.

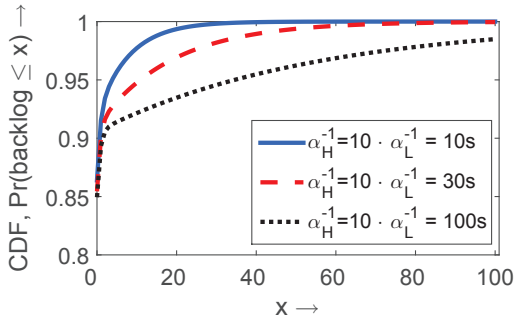


Figure 11. CDF of request backlog (queued requests) for different non-interference lengths and a fixed ratio (0.1) of interference to non-interference length. $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 20$ req/s.

with the length of the non-interference phase, we see an interesting behavior:

Observation 3. Response time increases sub-linearly with an increase in length of the interference phase.

Based on the above three observations, we suggest:

Rule-of-thumb 1. The intensity of interference is more significant than its length for a single VM. Hence, interference mitigation approaches should focus more on reducing interference, even if it takes longer to do so.

5.1.3. Impact of rate (or frequency) of interference

Figure 10 investigates the impact of rate of interference on response time. Here, we fix $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 50$ req/s, and set $\alpha_H = \alpha_L$. Then, we vary α_H and α_L together to see the impact on response time. Interestingly, as the rate (length) of interference increases (decreases), the response time decreases, and approaches the response time under an M/M/1 with service rate equal to the average of service rates under interference and non-interference. We summarize this finding as:

Observation 4. The ratio of length of interference and length of non-interference is not a good metric when evaluating performance under interference.

Thus, while Figure 2(b) in Section 2 is informative, it will change depending on the absolute lengths. We believe the reason behind this observation is that since response time increases quickly under interference, the longer the interference period, higher will be the backlog, resulting in non-linearly higher response times.

We verify this by plotting the CDF of the backlog, or the number of queued requests (obtained via the π estimates determined in Section 3.5), in Figure 11. Here, we set $\lambda = 25$ req/s, $\mu_H = 100$ req/s, and $\mu_L = 20$ req/s; we then fix a ratio, 0.1, of interference length to non-interference length, and vary the non-interference length. We see that the backlog is worse for longer lengths, despite the same interference intensity (μ_L) and ratio of interference length to non-interference length. Note that the value at $x = 0$ corresponds to no backlog (no queueing). Thus, a single long interference period is worse than several shorter periods.

Rule-of-thumb 2. Frequent but short interference periods are less harmful than infrequent but long periods, assuming the interference intensity is the same.

Practitioners may use this result, for example, to prefer colocation with transactional workloads over batch workloads. Note that Rules-of-thumb 1 and 2 are different – #1 focuses on interference intensity (or μ_L) versus length, whereas #2 focuses on frequency of interference, given fixed intensity.

5.2. Multi-VM systems under interference

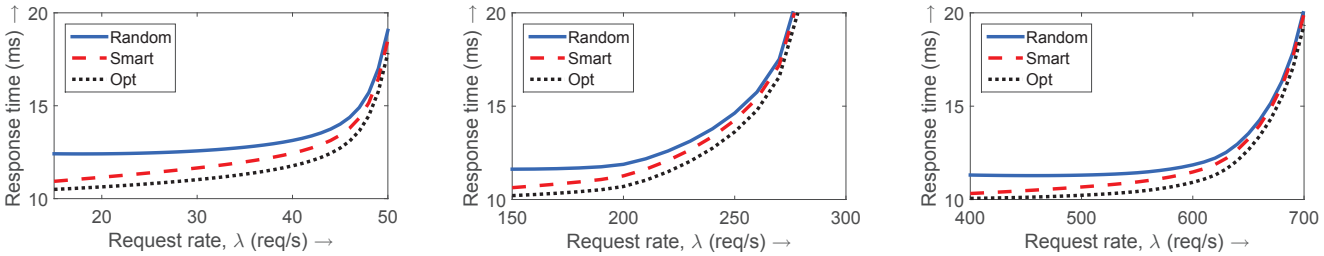
We now analyze the multi-VM M/M/k/int Markov chains under RandomSched, SmartSched, and OptSched. Recall, from Section 3.4.2, that RandomSched is the interference-oblivious policy that randomly routes requests to available VMs. SmartSched is interference-aware and preferably routes requests to available non-interference VMs. OptSched is the unrealistic policy that is interference-aware, but can also immediately migrate requests from interference to non-interference VMs, when possible.

The goal of our analysis is to understand the impact of the various factors, including the scheduling or load-balancing policy, on application performance under interference. While interference-aware schedulers have been proposed [7], [8], they are typically empirically evaluated only under specific workload and interference conditions.

5.2.1. Impact of #VMs and request rate

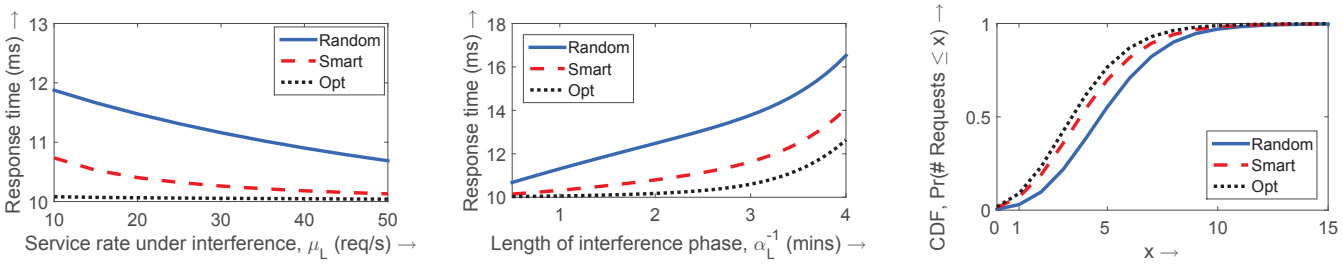
Figure 12 shows response time as a function of request rate for different values of k under all three schedulers. Here, $\mu_H = 100$ req/s, $\mu_L = 25$ req/s, $\alpha_H^{-1} = 10$ mins, and $\alpha_L^{-1} = 1$ min. The request rate range in each case is chosen to highlight the difference in performance between policies. In terms of load, the request rate range in Figures 12(a), 12(b), and 12(c) roughly corresponds to 0.1 – 0.3, 0.3 – 0.6, and 0.4 – 0.75, respectively.

In general, we see that the performance difference between policies is greatest under low to moderate request rates. Specifically, the performance under RandomSched is about 10–20% worse than OptSched at low request rates (left of the graphs); note that the y-axis starts at 10ms (and not 0) to better highlight the results. SmartSched is also worse than OptSched, as expected, but the difference is usually about 5% or so. At high request rates, the performance under all policies starts to converge. This is



(a) $k = 2$ VMs. (b) $k = 5$ VMs. (c) $k = 10$ VMs.

Figure 12. Response time as a function of request rate for different k values. Here, $\mu_H = 100$ req/s, $\mu_L = 25$ req/s, $\alpha_H^{-1} = 10$ mins, and $\alpha_L^{-1} = 1$ min.



(a) Impact of interference intensity. (b) Impact of interference length. (c) Build up of requests under interference.

Figure 13. Impact of interference on the performance of multi-VM systems. Here, $k = 10$, $\lambda = 400$ req/s, $\mu_H = 100$ req/s, and $\alpha_H^{-1} = 10$ mins.

because, at high load, incoming requests will be queued with high probability, thus obviating the scheduling decision. In other words, at high request rates, workload characteristics start dominating over the scheduling decision. Further, note that all Markov chains are similar in the repeating portion ($i \geq k$), which is where most of the time is spent at high load.

Likewise, we find that the difference between performance under different policies decreases as k increases; for $k > 10$, we find that the difference in performance across all policies is typically less than 10% for the parameter values in Figure 12. Even for the M/M/k without interference, it is known that, for a given load, performance significantly improves with k [11]. This is because the probability that an incoming request finds all VMs busy decreases quickly with an increase in k. Note that $\mu_H^{-1} = 10$ms is the (theoretical) lowest achievable response time.

Observation 5. For multi-VM scheduling, the performance gap between policies tends to decrease with an increase in request rate and/or an increase in the number of VMs.

5.2.2. Impact of interference characteristics

Figure 13(a) shows response time as a function of interference intensity (represented by the service rate under interference, μ_L) for different schedulers with $k = 10$, $\lambda = 400$ req/s, $\mu_H = 100$ req/s, $\alpha_H^{-1} = 10$ mins, and $\alpha_L^{-1} = 1$ min. We see that response time, and the difference in response times, for the various schedulers increases as interference intensity increases (right to left).

Figure 13(b) shows response time as a function of interference length (α_L^{-1}) for different schedulers with the same parameters as above with $\mu_L = 25$ req/s. Again, response time, and the difference in response times, increases as the interference length increases.

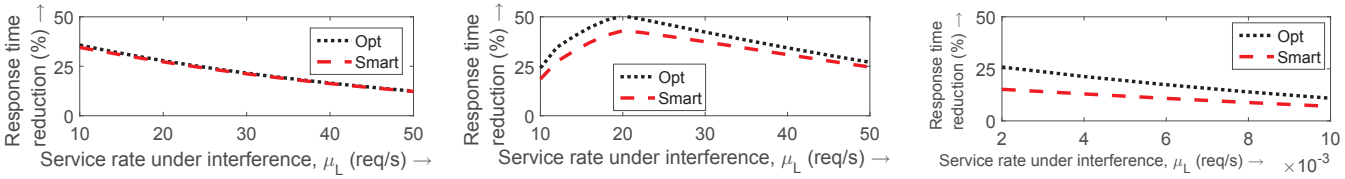
However, comparing Figures 13(a) and 13(b) (note the difference in y-limits), we see that response time, and the difference in response time under different schedulers, is more sensitive to interference length than interference intensity. This is in contrast to our Rule-of-thumb 1 for a single VM. The reason for this difference in behavior under single-VM and multi-VM is that, for multi-VM, even if the intensity of interference is high, the probability that *several* servers are simultaneously under interference is low; thus, the impact of interference intensity across all requests in the system is amortized. For a single VM, the probability of being under interference is simply $\alpha_L^{-1}/(\alpha_L^{-1} + \alpha_H^{-1})$; since there is only 1 VM, requests in the system are directly affected by interference intensity.

Rule-of-thumb 3. For multi-VM interference-aware scheduling, interference length is more critical than its intensity.

We investigate this issue further in Figure 13(c) which shows the probability distribution of number of requests in the system (obtained via the πs from Section 3.5). Here, $x > k = 10$ implies queueing. Since SmartSched and OptSched avoid VMs under interference, they tend to have fewer outstanding requests, resulting in better performance. RandomSched treats all VMs equally and thus has a worse distribution of requests in system.

5.2.3. Evaluating interference-aware scheduling

We now investigate the improvement in performance afforded by interference-aware scheduling policies, such as OptSched and SmartSched. Figures 14(a) and 14(b) show the improvement over RandomSched for different interference conditions with $\alpha_L^{-1} = 2$ mins and $\alpha_L^{-1} = 8$ mins, respectively, and $k = 10$, $\lambda = 200$ req/s, $\mu_H = 100$ req/s, and $\alpha_H^{-1} = 10$ mins; these values were chosen to highlight the difference in behavior between policies. We see that both OptSched and SmartSched provide significant improve-



(a) Interference length (α_L^{-1}) = 2mins. (b) Interference length (α_L^{-1}) = 8mins. (c) Interference = 2mins but with longer requests.

Figure 14. Reduction in response time afforded by OptSched and SmartSched over RandomSched for various interference conditions. Here, $k = 10$ VMs.

ments over RandomSched, to the tune of 50%. However, at high interference (lower values of μ_L in Figure 14(b)), the improvements are not that great because VMs are often under interference (since $\alpha_L^{-1} = 8$ mins) and there is less room for improvement via interference-aware scheduling. The improvements also decrease with an increase in request rate and k, as mentioned in Observation 5. Based on these figures, it might seem that OptSched, which exploits migration, does not provide significant benefits over SmartSched.

Figure 14(c) considers longer requests to highlight the importance of migration. We keep $\alpha_H^{-1} = 10$ mins and $\alpha_L^{-1} = 2$ mins, but vary other parameters to emulate longer service times. We set $\lambda = 0.04$ req/s, $\mu_H = 0.02$ req/s (or, 50s request length), and μ_L is varied accordingly. This time, we see that OptSched provides bigger improvements (almost $2\times$) over SmartSched, suggesting that migration is more helpful for longer requests. In summary:

Rule-of-thumb 4. Interference-aware schedulers that detect and avoid interference can significantly improve performance in multi-VM settings, especially under moderate interference conditions. The ability to migrate VMs is also beneficial, but only for longer requests.

While the above conclusion might not seem non-trivial, it does help to further validate our model. Importantly, our model can suggest exact parameter thresholds or conditions under which migration and interference-aware scheduling will be most beneficial, thus justifying their use. Finally, our model can also be extended to analyze interference-aware scheduling when the detection accuracy is not perfect, which is often the case in practical settings [26], [27]. We can model this by assigning a small probability ($q_{i,l,j}$, see Section 3.4.2) to route an incoming request to an interference VM even if a non-interference VM is present. We can also model the realistic case where migration incurs overheads [28] by incorporating setup times [29] for migration.

6. Related Work

Matrix Geometric/Analytic Methods [30], [31] are often widely employed to analyze the performance of single-server queues with phase-type distributions [11]. Of these, our model is closest to queueing systems with modulated service rates. Eisen [32] was among the first to consider queues with modulated service, but the analysis, using Generating Functions, was limited to an M/M/1 queue with slowdowns. In a follow-up work [33], Eisen analyzed the M/M/1 system via difference equations.

More recently, Zhou and Gans [34] analyzed an M/M/1 queue where the service rate changes at the end of service according to some probabilities. Mahabhashyam and Gautam [35] extended this work to allow service rates to change while in service. In our work, we focus on interference, and extend our model to M/M/k queues with various scheduling policies. While we employ Matrix Analytic Methods to solve our modeled Markov chains, we believe that Generating functions (see, for example [36]) or combinatorial techniques (that explicitly derive the rate matrix [37], [38]) could also apply. Recent works, such as RRR [39], have also proposed techniques to solve M/M/k-based chains that have a specific repeating structure; unfortunately, our complex M/M/k/int chains are not amenable to such methods.

There are also prior experimental works on analyzing interference. Cuanta [16] and Bubble-Up [17] use controlled experiments to assess the impact of different intensities of contention on performance; Cuanta focuses on on-chip contention and Bubble-Up on memory contention. Paragon [27], Quasar [8], and ICE [7] use machine learning to predict interference, and then leverage their findings to design interference-aware schedulers. CloudScope uses a discrete-time Markov chain to track the resource usage of VMs, and then designs an interference-aware migration scheme. Casale et al. [40] and DIAL [41] develop simple queueing models to estimate response time under interference, but do not consider the transient nature of interference. While effective, the analyses in the above works do not take workload characteristics and interference length into account.

7. Conclusion, Limitations, and Future Work

This paper presents a stochastic performance model for cloud-deployed applications under interference. Our model leverages Markov chains to track the system state as a function of various workload and interference characteristics, and provides estimates of request backlog and response time. Comparisons with an OpenStack-deployed web server under CPU contention highlight the accuracy of our model (average error of less than 5%).

The assumption of exponentially distributed variables is a significant limitation of our Markov chain-based modeling approach. We will investigate Phase-type distributions as part of future work to somewhat relax this assumption. We will also expand our validation efforts and consider different, possibly simultaneous, resource contentions.

Acknowledgements

This work was partially supported by NSF grants CNS-1617046 and CNS-1464151.

References

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

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

[3] D. Ghoshal, R. S. Canon, and L. Ramakrishnan, "I/O Performance of Virtualized Cloud Environments," in *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*, Seattle, WA, USA, 2011, pp. 71–80.

[4] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *Proceedings of the 29th IEEE International Conference on Computer Communications*, San Diego, CA, USA, 2010, pp. 1163–1171.

[5] A. Javadi, S. Mehra, B. Vangoor, and A. Gandhi, "UIE: User-centric Interference Estimation for Cloud Applications," in *Proceedings of the 2016 IEEE International Conference on Cloud Engineering (Work-in-Progress track)*, ser. IC2E '16, Berlin, Germany, 2016.

[6] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*, ser. Middleware '14, Bordeaux, France, 2014, pp. 277–288.

[7] A. Maji, S. Mitra, and S. Bagchi, "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.

[8] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14, Salt Lake City, UT, USA, 2014, pp. 127–144.

[9] L. Kleinrock, *Queueing Systems, Volume I: Theory*. Wiley-Interscience, 1975.

[10] ——, *Queueing Systems, Volume 2*. New York: Wiley-Interscience, 1976.

[11] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[12] A. Gandhi, P. Dube, A. Karve, A. Kochut, and H. Ellanti, "The Unobservability Problem in Clouds," in *Proceedings of the 2015 IEEE International Conference on Cloud and Autonomic Computing*, Cambridge, MA, USA, 2015.

[13] C. Delimitrou and C. Kozyrakis, "iBench: Quantifying interference for datacenter applications," in *Proceedings of the 2013 IEEE International Symposium on Workload Characterization*, 2013, pp. 23–33.

[14] Y. Amannejad, D. Krishnamurthy, and B. Far, "Detecting Performance Interference in Cloud-based Web Services," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 423–431.

[15] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI 2: CPU Performance Isolation for Shared Compute Clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 2013, pp. 379–391.

[16] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ser. SOCC '11, Cascais, Portugal, 2011, pp. 1–14.

[17] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '11, Porto Alegre, Brazil, 2011, pp. 248–259.

[18] G. Latouche and V. Ramaswami, *Introduction to Matrix Analytic Methods in Stochastic Modeling*. Philadelphia, PA, USA: ASA-SIAM, 1999.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A Berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

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

[21] D. Novakovi, N. Vasi, S. Novakovi, D. Kosti, and R. Bianchini, "Deepdive: Transparently identifying and managing performance interference in virtualized environments," in *Proceedings of 2013 USENIX Annual Technical Conference*, ser. ATC '13, San Jose, CA, USA, 2013, pp. 219–230.

[22] J. Little, "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research*, vol. 9, pp. 383–387, 1961.

[23] "The Apache HTTP Server Project," https://httpd.apache.org.

[24] "Stress-ng," http://kernel.ubuntu.com/~cking/stress-ng.

[25] D. Mosberger and T. Jin, "httperf—A Tool for Measuring Web Server Performance," *ACM Sigmetrics: Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[26] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt, "CloudScope: Diagnosing and Managing Performance Interference in Multi-tenant Clouds," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, Atlanta, GA, USA, 2015, pp. 164–173.

[27] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13, Houston, TX, USA, 2013, pp. 77–88.

[28] S. Nathan, U. Bellur, and P. Kulkarni, "Towards a Comprehensive Performance Model of Virtual Machine Live Migration," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15, Kohala Coast, HI, USA, 2015, pp. 288–301.

[29] A. Gandhi, M. Harchol-Balter, and I. Adan, "Server farms with setup costs," *Performance Evaluation*, vol. 67, pp. 1123–1138, 2010.

[30] M. F. Neuts, "Markov Chains with Applications in Queueing Theory, Which Have a Matrix-Geometric Invariant Probability Vector," *Advances in Applied Probability*, vol. 10, no. 1, pp. 185–212, 1978.

[31] ——, "Matrix-analytic methods in queuing theory," *European Journal of Operational Research*, vol. 15, no. 1, pp. 2 – 12, 1984.

[32] M. Eisen, "Effects of Slow-Downs and Failure on Stochastic Service Systems," *Technometrics*, vol. 5, no. 3, pp. 385–392, 1963.

[33] ——, "An Approximate Method for a Queuing Process with a Randomly Deteriorating Server," *Operations Research*, vol. 11, no. 6, pp. 996–1000, 1963.

[34] Y.-P. Zhou and N. Gans, "A Single-Server Queue with Markov Modulated Service Times," Wharton School Center for Financial Institutions, University of Pennsylvania, Tech. Rep. 99-40, Oct. 1999.

[35] S. R. Mahabhashyam and N. Gautam, "On Queues with Markov Modulated Service Rates," *Queueing Systems: Theory and Applications*, vol. 51, no. 1-2, pp. 89–113, 2005.

[36] I. Adan and J. Resing, "A class of Markov processes on a semi-infinite strip," Eindhoven University of Technology, Department of Mathematics and Computing Sciences, Tech. Rep. 99-03, 1999.

[37] J. Van Leeuwaarden and E. Winands, "Quasi-birth-and-death processes with an explicit rate matrix," *Stochastic models*, vol. 22, no. 1, pp. 77–98, 2006.

[38] B. Van Houdt and J. van Leeuwaarden, "Triangular M/G/1-Type and Tree-Like Quasi-Birth-Death Markov Chains," *INFORMS Journal on Computing*, vol. 23, no. 1, pp. 165–171, 2011.

[39] A. Gandhi, S. Doroudi, M. Harchol-Balter, and A. Scheller-Wolf, "Exact Analysis of the M/M/k/setup Class of Markov Chains via Recursive Renewal Reward," in *Proceedings of the 2013 ACM International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '13, Pittsburgh, PA, USA, 2013, pp. 153–166.

[40] G. Casale, C. Ragusa, and P. Parpas, "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*, vol. 2. IEEE, 2013, pp. 152–157.

[41] A. Javadi and A. Gandhi, "DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing," in *Proceedings of the 14^{th} IEEE Internation Conference on Autonomic Computing*, ser. ICAC '17, Columbus, OH, USA, 2017.