Tail Amplification in n-Tier Systems: A Study of Transient Cross-Resource Contention Attacks

Shungeng Zhang*, Huasong Shan*§, Qingyang Wang*, Jianshu Liu*, Qiben Yan†, Jinpeng Wei‡

*Louisiana State University–Baton Rouge, †University of Nebraska–Lincoln,
‡University of North Carolina–Charlotte, §JD.com Silicon Valley R&D Center–Mountain View,

***Abstract*—Fast response time becomes increasingly important for modern web applications (e.g., e-commerce) due to intense competitive pressure. In this paper, we present a new type of Denial of Service (DoS) Attacks in the cloud, MemCA, with the goal of causing performance uncertainty (the long-tail response time problem) of the target n-tier web application while keeping stealthy. MemCA exploits the sharing nature of public cloud computing platforms by co-locating the adversary VMs with the target VMs that host the target web application, and causing intermittent and short-lived cross-resource contentions on the target VMs. We show that these short-lived cross-resource contentions can cause transient performance interferences that lead to large response time fluctuations of the target web application, due to complex resource dependencies in the system. We further model the attack scenario in n-tier systems based on queuing network theory, and analyze cross-tier queue overflow and tail response time amplification under our attacks. Through extensive benchmark experiments in both private and public clouds (e.g., Amazon EC2), we confirm that MemCA can cause significant performance uncertainty of the target n-tier system while keeping stealthy. Specifically, we show that MemCA not only bypasses the cloud elastic scaling mechanisms, but also the state-of-the-art cloud performance interference detection mechanisms.**

***Index Terms*—Performance uncertainty, n-tier systems, web attack, millibottleneck, resource contention.**

I. Introduction

Cloud computing has been predicted by Berkeley [6] as the top one opportunity to defend against DoS attacks because of its elasticity: "systems can scale easily to fit the dynamic user requirements, even to serve the attack traffic". During the past decade a large number of websites are moved into the cloud (e.g., Spotify [2] moved its core infrastructure to Google Cloud on Feb. 23, 2016). However, DoS attacks are still very active and even more severe [45], because the ever-evolving new types of DoS attacks exploit various newly discovered network or system vulnerabilities even in the cloud, bypassing not only the state-of-the-art defense mechanisms [39], [59], but also the elasticity mechanisms of cloud computing [47], [48], [54].

The new types of DoS attacks in the cloud can be categorized into two classes: external and internal attacks [7], [13]. External attacks are similar to the traditional DoS attacks that launch external attacking traffic (both application and network level) to the target services [35], [39], [59], but with different attacking approaches or goals to bypass the scaling capability of the cloud. For example, an external attack may attack third-party services (e.g., the Dyn DNS servers outside of the cloud computing platform) that the target services rely on, or cause partial denial of service (PDoS) of the target service by sending pulsating but legitimate HTTP traffic to the target system [47], [48]. In contrast, internal attacks are new-born, which are emerging simultaneously with cloud computing and become an important class of DoS attacks due to the sharing nature of public cloud [7], [13]. Internal attacks can easily mount the adversary programs in the co-located VMs [44] (on the same host with the target VM) that cause resource contention and performance interference of the target VM and cause performance uncertainty of the target service [14], [60].

Existing approaches to detect and mitigate performance interference are either provider-centric [36], [61] or user-centric [24], [32], [33]. For provider-centric approaches, cloud providers profile the infrastructure-level metrics from the hosts in the cloud. Due to the requirement of a worthwhile investment of cloud providers (e.g., profiling overhead should be under 1% [26]), cloud providers typically adopt coarse granularity monitoring (in minutes level). For example, the sampling interval of Amazon's monitoring tool CloudWatch [1] and Microsoft Azure Application Insights [37] is typically 1 minute, which is incapable of detecting short-term performance interference (e.g., $<$ 1 minute). For user-centric approaches, cloud tenants protect their applications from performance interference in their rented VMs. They may enable fine-grained monitoring (e.g., 1 second), with the cost of non-trivial monitoring overhead, to detect short-lived performance interference. However, user-centric approaches typically lack host-level information (e.g., the state of the co-located adversary VMs), thus they are unable to detect the causes of performance interference.

In this paper, we present a new type of internal attacks in the cloud, which can cause significant performance uncertainty (long-tail response time) of target web applications while bypassing the elasticity mechanisms and the interference detection mechanisms in the cloud. Concretely, an attacker mounts an adversary program in the co-located VMs with the target VM that hosts the target web application; the adversary program can cause large response time fluctuations of latency-sensitive web applications through creating very-short-lived (e.g., $<$1 second) performance interference among the co-located VMs hosted in the same physical machine in the cloud. Such an attack is considered a great threat for modern web applications that require rapid responsiveness [15], [16], [46], such as e-commerce, media streaming, and online gaming. For example, Amazon reported that every 100ms increase in

the page load decreases sales by 1% [27]; Google requires 99th percentile of its queries to finish within 500ms [12]. At the same time, the performance interference caused by the proposed attacks only lasts for every short time period (e.g., 100ms) each time; from the Sampling theory, the average system resource utilization is still at a moderate level (e.g., 50%) using coarse granularity monitoring, thus not only avoiding the typical triggering conditions (e.g., CPU usage $>$ 80%) of the cloud scaling, but also escaping the state-of-the-art detection mechanisms of performance interference in the cloud.

To make an effective internal attack, the primary challenge is to choose the target attack resource. In cloud environments, the shared hardware and software resources among the co-located VMs are essential for internal attacks, such as network bandwidth [8], [30], I/O [20], last level cache [60], memory lock [60], and CPU scheduling mechanism of the hypervisors [62]. These shared resources are usually inter-correlated with each other, causing *cross-resource contention*. For example, network traffic affects last level cache [52] and memory bandwidth affects CPU utilization. These cross-resource contentions [31] significantly raise the difficulty in tracing the cause of performance interference, increasing the stealthiness of internal attacks. In this paper, we explore one type of cross-resource internal attacks, in which an adversary program intermittently triggers memory resource contention and degrades the available memory bandwidth among the co-located VMs, which transiently saturates the CPU of the target VM. We name this attack *MemCA* (Memory Attacks on the neighbor's CPU).

In brief, this work makes the following contributions:

- Present a new type of internal DoS attacks named *MemCA*, that can cause long-tail response time of web applications with high stealthiness since the average utilization of the target system is far from saturation.
- Introduce a type of cross-resource contentions that can cause significant performance interference among the co-located VMs in the host inside the cloud.
- Model the proposed attack scenario in n-tier systems based on queuing network theory, and analyze cross-tier queue overflow and tail (response time) amplification under our attacks.
- Validate the practicality of our attacks through extensive benchmark experiments in both private and public clouds such as Amazon EC2, confirming both damages (e.g., 95th percentile response time $>$ 1 second) and stealthiness (bypassing elasticity mechanisms and interference detection mechanisms in the cloud).

We outline the rest of this paper as follows. Section II discusses the background of memory resources and describes our attack scenario and the practical impact of tail amplification. Section III investigates two types of memory attacks: saturating memory bus and triggering memory lock. Section IV designs and implements MemCA, models the attack scenario using queuing network theory, and analyzes the attack impacts (e.g., tail amplification). Section V evaluates MemCA from two aspects: damage and stealthiness. Section VI presents related work and Section VII concludes the paper.

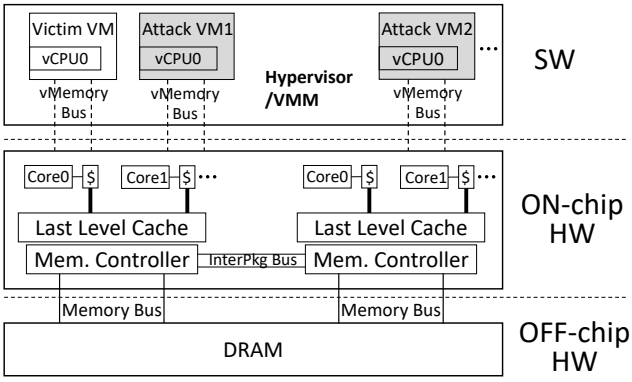


Fig. 1: **On-chip resources shared by co-located VMs.**

II. Background and Motivations

A. Shared On-Chip Resources and Resource Contention

In modern CPU architecture (see Figure 1), such as Intel Xeon family which is widely used by today's IaaS providers, there exist multiple processor sockets splitting on-chip resources into different packages. In the same processor package, except for core-private L1/L2 caches, last level cache (LLC) and memory scheduling components (e.g., memory controller bus, bank scheduler, and channel scheduler) are all shared by co-located VMs. The commercial cloud providers allow users to choose VM instances with different memory size and CPU cores [4], [38], but it is insufficient to isolate all other on-chip memory resources, leading to potential resource contention between co-located VMs, such as memory bandwidth, or even *cross-resource contention*. For example, RFA [52] investigated performance degradation on LLC caused by the network; LLC contention causes co-located VMs to require more memory bandwidth, creating a memory bandwidth contention.

In this work, we focus on one type of cross-resource contentions: shared on-chip memory resource contention on CPU among the co-located VMs – *MemCA*. This type of cross-resource contentions is hard to detect because the cause and the result are indirectly correlated (e.g., CPU saturation does not mean CPU is the bottleneck, but the limited memory bandwidth is), which is a big challenge for current detection mechanisms [31]. In addition, typical memory metrics supported by monitoring tools (e.g., sar and collectl) cannot cover all the memory attack cases (e.g., memory lock).

B. Threat Model and Assumptions

We consider a MemCA attack scenario, in which the adversary frequently creates shared on-chip memory resource contentions of the victim VMs in the cloud hosting the target web system by intermittently saturating those shared on-chip memory resources (e.g., memory bandwidth) without being detected. Today's software-based VMM (e.g., Xen, KVM, VMware vSphere, and Microsoft Hyper-V) can only guarantee secure access to virtual and physical memory pages, but do

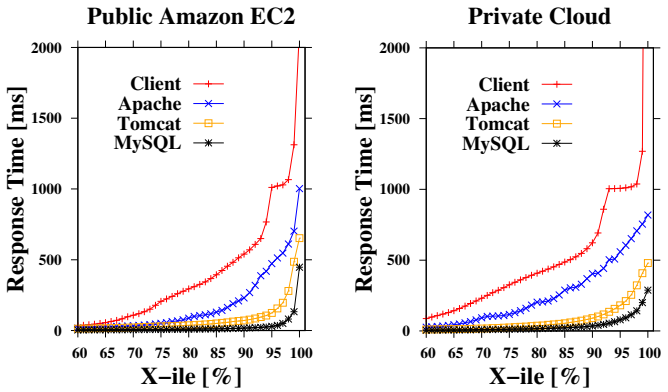


Fig. 2: **Measured tail (percentile) response time in each tier of a 3-tier system under our MemCA attack. We observed significant tail response time amplification from the back-end MySQL to the front-end Apache, and eventually to the clients. For example, obvious long-tail response time was observed by the clients in both our private cloud and the public Amazon EC2 environment.**

not isolate on-chip memory resources shared by VMs [17]. In this case, a MemCA attacker can increase CPU consumption of the target VM by memory attacks in co-located adversary VMs, even though vCPUs are isolated and protected by the hypervisor. Finally, such an attack is able to cause performance uncertainty of the target web system, especially cause the long-tail response time problem in the long run.

To effectively launch a MemCA attack, we assume that the adversary VM is able to share the same host with the target VMs. Many previous research efforts already provide solutions [23], [53], [58], and are orthogonal to our research. Ristenpart's team [53] reported the cost of achieving co-located VMs in public IaaS cloud (average cost is from $0.137 to $5.304), and the successful rate of VM co-location (from 0.6 to 0.89). We also assume that the attacker can fully control the rented adversary VMs, in other words, they can run any attacking programs in the rented VMs [29], as is the case for today's public IaaS cloud. In addition, unlike traditional DDoS attacks which usually require a large amount of bot machines, a MemCA attack only requires one or a few adversary VMs that are co-located with any component VMs that are in the critical path of the target web system, so such an attack is economically feasible considering the potential damage it can bring to the target website.

C. Experimental Illustration of Attack Impact

We conduct concrete experiments to show the damaging impact caused by MemCA using a representative 3-tier benchmark RUBBoS [42], deployed in both our private cloud and the public Amazon EC2 cloud. Here we just show the attack impact; more experimental details and explanations are available in Section V. Without our MemCA attack, the benchmark web application responds to every client request within 100ms. With our MemCA attach, we show the results in Figure 2. We can see a clear tail response time amplification in each tier of the 3-tier system, in other words, the response time of each tier has a nonlinear tail trend as percentile increases. Meanwhile, the tail response time of each tier amplifies from the back-end tier (MySQL) to all the front-end tiers (Tomcat and Apache), and eventually to the clients. For example, the 95th and 98th percentile response time observed by the clients is longer than 1 and 2 second(s), respectively, which is considered as severe performance damage by most e-commerce websites [12], [15], [16], [27].

III. Memory Attack Measurements

In this section, we investigate the performance impact of memory attacks among the co-located VMs in a representative host widely used by current cloud platforms.

Experiment Methodology. In our private cloud, we use 3 machines to build our profiling environment, each of which equips with a 12-core, 2-package 1.60 GHz Intel Xeon CPU E5-2603 v3 with 15MB LLC per package and 16GB of main memory, a type of processors in Intel Xeon family that is widely used to host VM instances by Amazon EC2 [4]. Our private cloud is managed by OpenStack Ocata; each VM, running CentOS release 6.7, is managed by KVM [28].

To measure the performance impact of memory attacks among the co-located VMs, we exploit two approaches to launch memory attacks: (1) saturating memory bus through memory benchmarking tools [24], and (2) locking memory access through unaligned atomic operations [60]. At the same time, we measure available memory bandwidth which can be used by each co-located VM as an evaluation metric to assess performance interference caused by memory attacks.

The program to measure memory bandwidth of the host is RAMspeed, a cache and memory benchmarking tool [19]. As we have 6 cores per package in our host, we deploy 6 co-located VMs, each with one vCPU. To comprehensively understand the impact of memory attacks on shared memory bandwidth in modern CPU architecture (see Figure 1), we execute attack programs and measure memory bandwidth in two scenarios:

1) Same package, 6 VMs each pinned to a separate core on the same package, which share last-level cache and memory bandwidth per package.
2) Random package, 6 VMs floating over 12 cores on two packages, which share last-level cache and memory bandwidth of both packages. This case represents the common practice in a real cloud computing platform aiming to increase the level of sharing resources.

Results of Memory Bandwidth Contention among Co-located VMs. Figure 3 depicts measured available memory bandwidth used by each co-located VM in the same and random package under memory attacks through either saturating memory bus or locking memory access. We summarize several main results for building MemCA.

1) One VM running the attack program does not saturate the memory bus, since memory bandwidth in modern processors is high enough to host two co-located VMs.

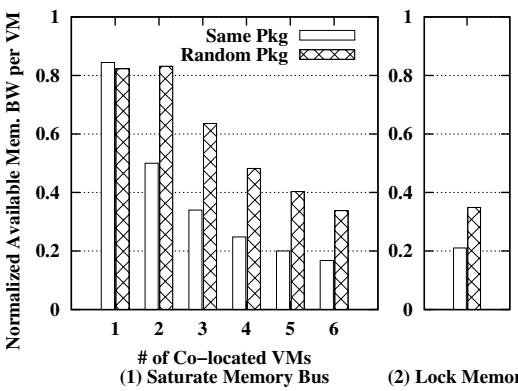


Fig. 3: **Memory bandwidth degradation under two types of memory attacks.** (1) As co-located VMs increases, available memory bandwidth used by each VM decreases. (2) Locking memory (launched by 1 VM) is more effective in degrading available memory bandwidth than saturating memory bus.

2) As co-located VMs increases, available memory bandwidth used by each VM decreases in same package case. In the random package case, the trend is similar, but the degradation level is lower, since the capacity of memory bus in two packages is twice as that in the same package.
3) Locking memory access is more effective to degrade available memory bandwidth than saturating memory bus since memory access from other applications are completely blocked until the locked action is done.

The above memory attack experiments are all conducted in the KVM platform. To further confirm that the effectiveness of these memory attacks in co-located VMs is not related to any specific hypervisor, we also conduct similar experiments in different platforms managed by different popular hypervisors (e.g., Xen, VMware, and Hyper-V). We get similar results under the same memory attacks as shown in Figure 3.

IV. MemCA

A. MemCA Overview

Through memory attack experiments in Section III, we profile the capacity of the target host and determine the attack intensity of the adversary program. A recent research effort [60] has introduced the performance impact of brute force memory attacks for co-located VMs. However, such attacks can be detected by sophisticated state-of-the-art detection mechanisms [32], [33], [61]. Here we exploit these previous brute force memory attack techniques and design a much more stealthy internal attack, MemCA.

The main idea of MemCA is to saturate or degrade the shared resources (e.g., memory bandwidth) quickly within a very short time period (e.g., tens to hundreds of milliseconds) while bypassing the typical coarse-grained monitoring based detection mechanisms (e.g., second or minute) [32], [33]. Then MemCA intermittently repeats this attack process to cause the long-lasting threat to the target system performance. In other words, MemCA creates very short resource contention

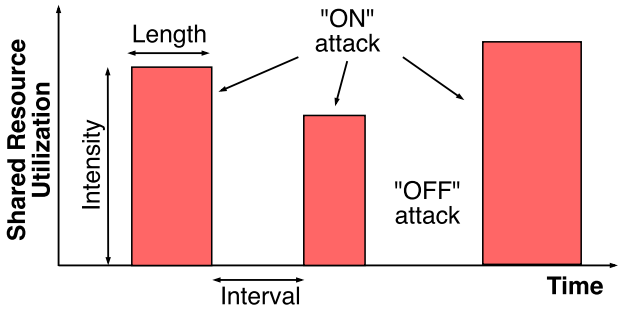


Fig. 4: **An illustration of MemCA bursts**.

bursts in an ON-OFF style (Figure 4) to create performance uncertainty. We formally propose MemCA as follows:

$$Effect = \mathbb{A}(R, L, I) \quad (1)$$

where,

- $Effect$ is the measurement of the attack impact. We use percentile response time as the metric to measure the tail response time of the target system (e.g., 95th percentile response time $>$ 1s).
- R is the intensity of resource consumption per interference attack burst. R should be large enough to temporarily saturate or degrade shared resource (e.g., memory bandwidth) in the servers of the cloud.
- L is the lasting period (length) of each interference burst. L should be short enough (e.g., $<$ 1s) to guarantee that the interference burst is not captured by the interference detection mechanisms [32], [33], [61].
- I is the time interval between every two consecutive interference bursts. I infers the frequency of attack bursts. I should be short enough so that the attacker can generate interference bursts frequent enough to cause significant performance damage on the target system. On the other hand, too short I makes the attack similar to traditional flooding DDoS attacks, which can be easily detected.

Given this high-level design of MemCA, we need to tackle two key challenges to make MemCA practical. They are addressed in the following sections.

- How can we determine the relationship between the attack impact and the attack parameters? Section IV-B will use queuing network theory to model a typical n-tier system, generalize the attack scenarios of MemCA, and analyze the phenomenon of cross-tier queue overflow [55] and response time amplification under MemCA.
- How can an attacker get the optimal attack parameters without the knowledge of performance parameters of the target n-tier system, which is a typical case in a public cloud environment? Section IV-C will exploit feedback control theory to dynamically adjust attack parameters for better attack effectiveness and stealthiness, even without knowing the performance parameters of the target systems (e.g., the service time and resource utilization).

B. MemCA Modeling

System Model and Problem Statement. Previous study [18], [57] shows that queueing theory has been used on quantitative

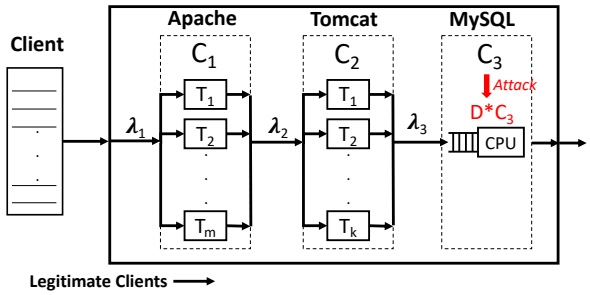


Fig. 5: **MemCA scenario and system model.**

Param.	Description
Q_i	the queue size for the *i*th tier
$C_{i,OFF}$	the capacity for the *i*th tier during OFF bursts
$C_{i,ON}$	the capacity for the *i*th tier during ON bursts
λ_i	the legitimate request rate arriving in the *i*th tier
D	the degradation index of the capacity of the *n*th tier
$l_{i,UP}$	the time to fill up the queue of the *i*th tier per burst
$l_{i,DOWN}$	the time to drop down the queue of the *i*th tier per burst
P_D	the damage period of the target VM during a burst
P_{MB}	the millibottleneck period of the target VM during a burst
ρ	overall percentile response time under MemCA

TABLE I: **Model and system parameters**

analysis for DoS attacks in computer systems. In this section, we start our system model with a tandem queue, which is usually exploited to model complex computer systems such as n-tier systems. In an n-tier system, the client requests are offered to the first queue, the output of which are fed to the second queue, and so on. Each tier has an arrival rate of a Poisson process and a service rate that follows an exponential distribution. In a Tandem Queue model, the service rate is independent among different tiers in the system.

To realistically model the target n-tier system, we also assume the arrival rate of each tier is a Poisson process, and the capacity of each tier is an exponential distribution [49]. Figure 5 depicts a MemCA attack scenario and a classic 3-tier web applications model based on our RUBBoS environment (i.e., 1 Apache, 1 Tomcat, and 1 MySQL). The queue size of each tier denotes the size of concurrency control resources (e.g., server threads or connections) of each tier in the system. For example, Figure 5 shows there are *m* and *k* threads in Apache and Tomcat, respectively. In practice, both *m* and *k* cannot be very large (from tens to a few hundred) because of the well-known high multithreading overhead [55]. In our RUBBoS experiments, MySQL CPU is the critical (bottleneck) resource in the system. Since n-tier systems usually adopt synchronous RPC-style request/response for inter-tier communication, a queued request in the back-end MySQL will also cause a thread pending for response in every upstream tier such as Tomcat and Apache. Thus, queued requests in MySQL can easily exhaust the limited threads in upstream tiers, which is a key insight that motivates us to design the MemCA attacks. Table I summarizes the notation and description of the parameters used in our model.

The purpose of MemCA attacks is to cause the long-tail response time problem (maximize the attack impact $Effect$) while keeping stealthy. To analyze percentile response time of the target n-tier system caused by our attacks, we should pinpoint where the major time is consumed inside the n-tier system. Thus, we analyze the percentile response time observed in each tier. Meanwhile, we also need to quantify the stealthiness of our attacks. In other words, we should quantify the period of each interference burst (millibottleneck); shorter period of each interference burst means the attack is stealthier.

Capacity Degradation under Attack. In MemCA attacks, the attacker can degrade the capacity of the *n-th* tier through adversary programs executing in co-located VMs, blocking the process of requests in the *n-th* tier, which leads to queue overflow propagation from the *n-th* tier to all the upstream tiers. In a concrete workload environment, each host should have a peak capacity (service rate), R_{max}. Thus, during each MemCA attack burst, we define the degradation index D as:

$$D = \frac{R_{max} - R}{R_{max}} \tag{2}$$

where R is the attack intensity of each burst.

We also define $C_{n,ON}$ and $C_{n,OFF}$ to differentiate the degraded capacity of the *n-th* tier under the attack scenario from the normal capacity. Specifically,

$$C_{n,ON} = D * C_{n,OFF} \tag{3}$$

We note that during the ON attack periods, the MemCA attacker quickly saturates the queues in each tier of the system, making each tier working in a high concurrency mode. Previous research results [10], [56] show that the capacity of a server degrades as job concurrency increases due to multithreading overhead. To simplify our analysis, we disregard the capacity degradation caused by high concurrency; we only count the capacity degradation caused by MemCA attack bursts as shown in Equation 3.

Quantifying Attack Impact. To quantify the damaging impact of MemCA, we divide the queueing status of the system under attack into three stages: *build-up, hold-on, and fade-off*. We analyze the period of each stage in the following.

During the *build-up* stage, the queue will be filled up from the bottleneck tier to each upstream tier, until the front-most tier. In reality, the database tier in an n-tier system is very likely to be the bottleneck tier, which is the last-most tier. *If the queue size satisfies*

(Condition 1) $Q_1 > Q_2 > ... > Q_{n-1} > Q_n$

and the degradation index during build-up stage satisfies

(Condition 2) $\lambda_n > C_{n,ON}$

for all i=1,...,n, then the time needed to fill up the queue for the i-th server during a burst is approximately

$$l_{n,UP} = \frac{Q_n}{(\lambda_n - C_{n,ON})} \tag{4}$$

$$l_{n-1,UP} = \frac{(Q_{n-1} - Q_n)}{(\lambda_{n-1} + \lambda_n - C_{n,ON})} \tag{5}$$

$$\dots$$

$$l_{1,UP} = \frac{(Q_1 - Q_2)}{(\sum_{i=1}^{n} \lambda_i - C_{n,ON})} \tag{6}$$

Equation 4 denotes the time needed to fill up the queue in the *n-th* tier, where the adversary VMs co-locate with the

target VM. *Condition 2* is to make sure that the MemCA attack intensity is high enough so that the queue in the *n-th* tier is able to fill up, and the queue fill-up rate is $(\lambda_n - C_{n,ON})$. Once the *n-th* tier queue fills up, requests starts to queue in the *(n-1)-th* tier, and so on. We derive Equation 5 and 6 based on the characteristic of typical n-tier web systems, which usually adopt the RPC-style synchronous communication between consecutive tiers; one queued request in a downstream server holds a thread in every upstream server; so the available queue slots in the *(n-1)-th* tier is $(Q_{n-1} - Q_n)$ when the *n-th* tier queue fills up. In addition, all the traffic arriving to a downstream tier needs to go through every upstream tier, thus the request rate for *(n-1)-th* tier is $(\lambda_{n-1} + \lambda_n)$. The same reasoning process can be applied to every upstream tier including the front-most tier as shown in Equation 6.

During the *hold-on* stage, the biggest performance damage occurs because the queue in every tier is full. Two factors contribute to the damage. First, every request encounters the maximum queuing time in the system; second, new coming requests from clients start to drop by the front-most tier, leading to TCP retransmissions. Since the minimum TCP retransmission time-out is 1 second [21], clients will observe requests with very long response time. We call the hold-on stage as the damage period P_D caused by our MemCA attacks, which can be calculated as follows:

$$P_D = L - \sum_{i=1}^{n} l_{i,UP} \tag{7}$$

where L is the burst length (interference period) caused by adversary VMs, including both the build-up and hold-on stage.

To enable significant performance damage of our MemCA attacks, P_D for each attack burst should be the longer the better. We can quantify the impact of our MemCA attack on the target n-tier system as follows:

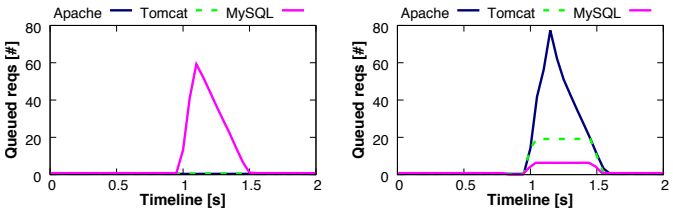
$$\rho = \frac{P_D}{I} \tag{8}$$

where I is the interval between every two consecutive bursts. For example, to achieve the attacking goal that 95th percentile response time longer than 1 second, the damage period P_D should be at least 100ms for a 2-second attacking interval.

Quantifying Attack Stealthiness. Although the damage period caused by an attack burst is shorter than the burst length itself (Equation 7), the millibottleneck period caused by each attack burst is beyond the burst length. This is because even if an attack burst stops, the system will still be busy in processing queued requests accumulated during the build-up and hold-on stage. We refer the post hold-on stage as the *fade-off* stage.

During the fade-off stage, the service rate of the *n-th* tier recovers to its full capacity $C_{n,OFF}$ because of removed interference, thus the queued requests in each tier will start to drain. Here we only consider the queue drain period in the *n-th* tier because this is the tier where resource contention occurs. The drain rate of the *n-th* tier queue is $(C_{n,OFF} - \lambda_n)$, thus this queue drain period can be approximated as follows:

$$l_{n,DOWN} = \frac{Q_n}{(C_{n,OFF} - \lambda_n)} \tag{9}$$



(a) In tandem queue, all the requests are queued in last tier.

(b) In our attack model, queue overflow propagates in all tiers.

Fig. 6: **Cross-tier queue overflow under MemCA (Figure 6b), comparing to tandem queue model (Figure 6a).**

We note that during the queue drain period, the critical resource (e.g., CPU) of the *n-th* tier will be fully utilized to process the queued requests; thus the overall millibottleneck period caused by an attack burst is as follows:

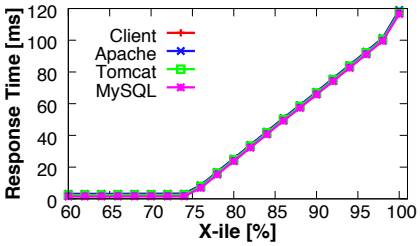
$$P_{n,MB} = L + l_{n,DOWN} \tag{10}$$

To guarantee the stealthiness of our MemCA attack, we should keep $P_{n,MB}$ as short as possible, for example, within sub-second to escape the detection of typical monitoring tools that normally use seconds or even minutes time granularity.

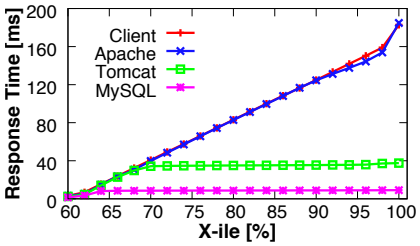
Simulation Analysis. To further illustrate the proposed model and quantify the attack impact, we adopt simulation using Java Model Tools (JMT) [9]. JMT is an open source suite for modeling Queuing Network computer systems. It is widely used in the research area of performance evaluation, capacity planning in n-tier systems. Thus, it is very suitable to evaluate the impact of our attacks on n-tier systems. We modify the JMT code and control the degradation index of service rate during an attack burst to simulate our MemCA attacks. To simplify the analysis, we fix the burst interval I and burst length L as 2 seconds and 100 milliseconds, respectively. With regard to the JMT simulation of a 3-tier system, we estimate the constant parameters (λ_i, C_i) of each tier measured from our real RUBBoS experiments.

Cross-Tier Queue Overflow. In Figure 6, we compare our system model with the classic tandem queue model under the same MemCA attack case (e.g., D = 0.1). Obviously, Figure 6b shows the process of cross-tier queue overflow in our system model, involving queue fill-up, hold-on, and fade-off, since new coming requests will be queued in an upstream tier once the queue of its consecutive downstream tier is full. On the contrary, in the tandem queue case in Figure 6a, all the requests are queued in the last tier.

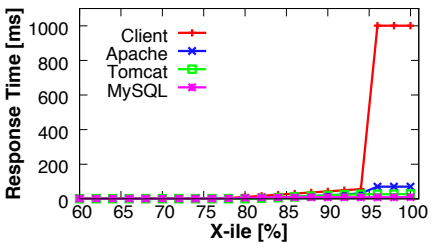
Tail Response Time Amplification. In Figure 7, we use the same attack parameters (e.g., D = 0.1, I = 2s, L = 100ms) to launch our MemCA attack, compare three cases, including a tandem queue model, our attack model with infinite Apache queue (not drop requests), and our attack model with finite Apache queue (can drop requests once all queues are full). Figure 7a shows the tandem queue case with infinite MySQL queue, percentile response time observed by all the tiers and client nearly overlap and continuously increase due to increased queueing time in MySQL. All the requests are queued



(a) Tandem queue case with infinite MySQL queue size. The percentile response time observed by all tiers nearly overlap. All requests are queued in MySQL (see Figure 6a).



(b) Our attack model case with infinite queue in Apache but finite queue in other tiers. Percentile response time of Apache and clients amplifies due to cross-tier queue overflow.



(c) Our attack model case with finite queue of each tier. Client perceives much longer peak response time than that in Figure 7b because of dropped requests and TCP retransmissions.

Fig. 7: **Tail response time amplification under MemCA.** (a) and (b) compares the tandem queue case with our model with infinite Apache queue. And (c) shows performance of our model with finite queues (the realistic settings in n-tier systems).

in MySQL under the attack (see Figure 6a). Figure 7b shows our attack model case with infinite Apache queue, percentile response time observed by Apache and clients is amplified due to cross-tier queue overflow. In practice, the Apache queue size has to be limited due to the high concurrency overhead as introduced previously. Figure 7c shows our attack model case with the finite queue of each tier; in this case the clients perceive much longer peak response time than that in Figure 7b and Figure 7a, since requests are dropped once all the queues in the 3-tier system are full, leading to TCP retransmission (the minimum timeout of TCP retransmission is 1 second).

Relationship between Attack Parameters and Impact. MemCA has two attack goals: causing high damage (e.g., 95th percentile response time $>$ 1s) while keeping stealthiness (e.g., millibottleneck length $<$ 1s). Through equations 8 and 10, we can calculate the damaging impact and the millibottleneck length if we know system parameters (See Table I) and attack parameters. On the contrary, based on the predefined attack goals, we can also calculate attack parameters if we know system parameters.

C. MemCA Implementation

It is difficult to accurately know the various parameters of the target n-tier system for a MemCA attacker. However, given the proposed model and simulation analysis, we understand the relationship between the attack parameters and their impact on the target system. So we exploit some advanced feedback control tools (e.g., Kalman filter [25]) to dynamically tune the attack parameters to fit the dynamic system state [47] and achieve our attacking goal. We implement a control framework which includes two components: MemCA frontend (MemCA-FE) and MemCA backend (MemCA-BE) (Figure 8). MemCA-FE executes the attack program in the co-located adversary VMs and reports the shared resource consumption. MemCA-BE consists of two components: a prober which periodically sends lightweight HTTP requests to the target web system and monitors the response time of the target web application, and a commander which dynamically controls the attack parameters of the adversary VMs based on the performance and the resource utilization metrics of historical attack bursts.

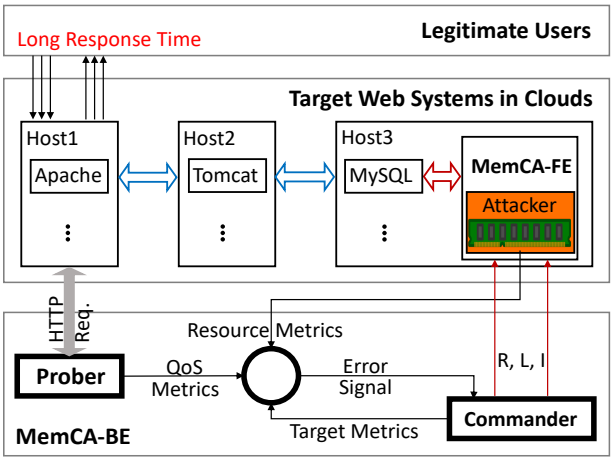


Fig. 8: **MemCA framework and a 3-tier sample topology.**

Estimate Critical Resource Utilization (MemCA's damage). In MemCA-FE, we record the critical resource utilization consumed by the adversary VMs through the attack program (e.g., RAMspeed [19]). In our MemCA attack case, the critical resource is memory bandwidth of the physical machine hosting both the adversary VMs and the target VM. The maximum memory bandwidth of the target machine is fixed and can be easily profiled by running some memory intensive benchmark in the adversary VMs.

In our attack control framework, we measure the percentile response time of the system through the prober in MemCA-BE. Due to the positive correlation between the system response time and resource utilization, we tune the attack intensity R of the adversary VMs using feedback control technology, to control the target resource utilization. Through adjusting the attack burst length L and the burst interval I, we can achieve the expected attack goal (e.g., 95th percentile response time $>$ 1 second).

Estimate Millibottleneck Length (MemCA's stealthiness). We record the execution time of the attack program (e.g., the RAMspeed benchmark) in the adversary VMs in MemCA-FE. During the execution time of the attack program, the target bottleneck resource is supposed to be busy and saturated.

Thus, we can exploit the execution time to estimate the millibottleneck length of the critical resource, further control the attack parameters (e.g., burst length L) to achieve the stealthiness goal. In practice, this approach is very conservative because the target bottleneck resource may not always be 100% utilized during the execution time of the attack program, thus the actual attack is stealthier than estimated using the execution time as the millibottleneck length.

V. MemCA Evaluation

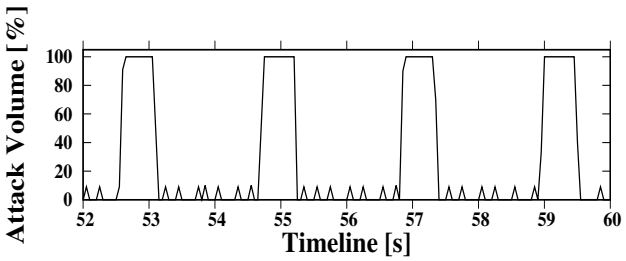
Here, we use the RUBBoS web application benchmark to evaluate the damage and the stealthiness of MemCA attacks.

A. MemCA Damage in RUBBoS

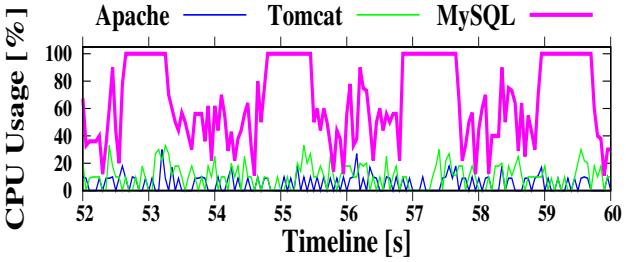
Experiment Methodology. We adopt RUBBoS [42], a representative n-tier web application benchmark modeled after the popular news website Slashdot. We configure RUBBoS using the typical 3-tier architecture (see the 3-tier system in Figure 8) in the Amazon EC2 platform, each tier is deployed in an EC2 c3.large instance hosted by a dedicated node (equipped with two ten-core Intel Xeon CPU E5-2680 and 64GB of main memory). Each instance has two vCPUs and runs Red Hat 7.3.1. We configure 3500 concurrent legitimate users using the default RUBBoS workload generator to interact with the target benchmark website. Each user follows a Markov chain model to navigate among different web pages, with averagely 7-second think time between every two consecutive requests. Since recent works [23], [53], [58] already provide solutions to co-locate VMs with a victim in Amazon EC2, the co-location step is orthogonal to our research. We co-locate VMs on a dedicated EC2 node to perform our experiments.

The adversary VMs are co-located with the MySQL VM of the target 3-tier benchmark website. As for the attack parameters, we fix the burst interval I to be 2 seconds and the burst length L to be 500 milliseconds, with the goal of making the average utilization of the critical resource to be moderate, thus bypassing the state-of-the-art detection of performance interference. Given the knowledge in Section III, we saturate the memory bandwidth of the target host through triggering memory lock during each attack burst, which is more effective than saturating the memory bus. Our attack damage goal is to cause the 95th percentile response time of the target benchmark website $>$ 1 second.

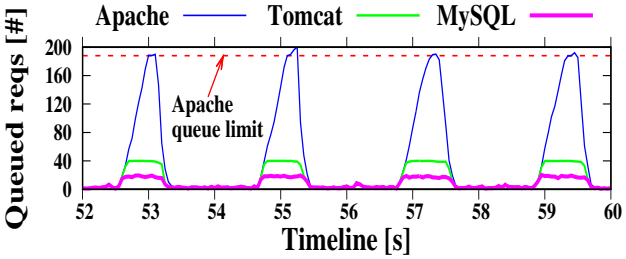
Results. We have conducted 3-minute RUBBoS experiments under MemCA attacks launched by our control framework. Figure 2(a) shows the corresponding percentile response time observed from each tier, suggesting that our attack achieved the damage goal. Figure 9 captures a representative 8-second snapshot with each metric monitored using fine-grained (50 ms) monitoring tools. This figure illustrates how MemCA attack bursts can cause long response time. Figure 9a shows a MemCA attack burst occurs in every 2 seconds. Each burst (launched by the attacking program) lasts for about 500ms, triggering memory lock in order to degrade available memory bandwidth of the host. The attack bursts in the adversary VM cause transient CPU saturations of the co-located MySQL VM in Figure 9b. Each transient CPU saturation creates a millibottleneck (less than 1 second) and causes requests to queue in MySQL; MySQL local queue soon fills up during the millibottleneck period, pushing requests to queue in the upstream Tomcat and Apache as shown in Figure 9c. Once the queued requests in the front-most Apache exceed its queue limit, new requests from legitimate users are dropped, resulting in TCP retransmissions. The minimum timeout of TCP retransmission [21] is 1 second. Thus, end users perceive very long response time as we observed in Figure 9d.



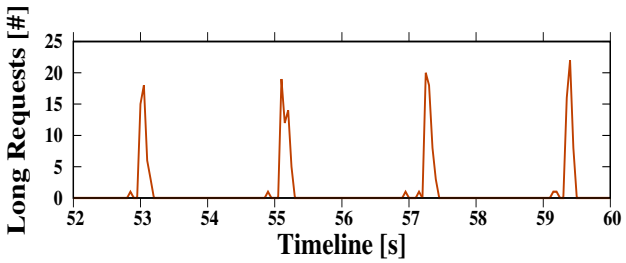
(a) 4 attack bursts with 2 seconds interval launched in the co-located adversary VM. Each burst lasts for about 500ms, triggering memory lock by adversary program run in an attack VM to degrade available memory bandwidth in the host.



(b) Transient CPU saturations of the target MySQL VM casued by the attack bursts in the co-located attack VM.



(c) Quick queue propagation in 3 tiers during each burst.



(d) Very long response time (e.g., $>$ 1 second) perceived by the end users due to MemCA bursts.

Fig. 9: MemCA Damage in RUBBoS.

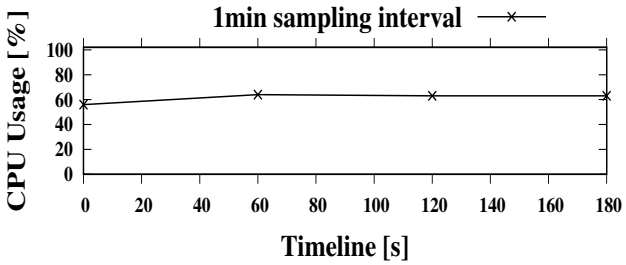
B. MemCA Stealthiness under Cloud Elasticity and Cloud Detection of Performance Interference

Evaluate MemCA under Cloud Elasticity. Amazon AWS provides Elastic Load Balancing in its EC2 platform to guarantee scalability, performance, and security for deployed applications [5]. It can elastically scale to fit the application demand because of the ability to trigger Auto Scaling [3] in the available Amazon EC2 instance fleet. The trigger mechanism of Auto Scaling is based on a coarse-grained monitoring tool, Amazon CloudWatch [1], whose sampling period is 1 minute. For example, an application can scale out more instances once the average CPU utilization of any instance of the system exceeds a preconfigured threshold (e.g., 85%) during a 1-minute sampling period. Thus, to validate whether Cloud Elasticity can mitigate our attack, we need to verify whether our attack can trigger the threshold of Auto Scaling. Here, in our experiments, we assume the preconfigured threshold of scaling out more instances is 85% of the average CPU utilization, which is usually a feasible and simple solution for system administrators [11], [43].

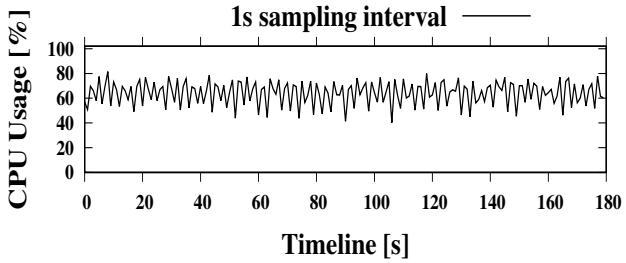
Results. Figure 10 shows the CPU utilization of the bottleneck tier MySQL in the previous 3-minute MemCA attacks experiments using different sampling granularity. Figure 10a, 10b, and 10c adopt 1-minute, 1-second, and 50-millisecond monitoring granularity, respectively. The average utilization in the 1-minute case is flat and moderate, which obviously will not trigger the condition of Auto Scaling. Using 1-second monitoring, the CPU utilization exhibits a little bit fluctuation, which will also fail to trigger Auto Scaling. Only using more fine-grained granularity 50 milliseconds, we can observe transient CPU saturations. These transient CPU saturations will also not trigger Auto Scaling due to the coarse monitoring granularity of CloudWatch.

There are two reasons why AWS Auto Scaling relies on coarse granularity monitoring provided by CloudWatch. First, the coarse granularity can significantly reduce monitoring overhead, especially if we consider there are tens of thousands of machines in a typical datacenter; each machine may also have hundreds of system metrics to monitor. Second, Auto Scaling wants to avoid over-sensitive to system state changes (e.g., CPU utilization fluctuation) in order to keep the stability of the target system. However, such coarse granularity monitoring provides a perfect opportunity for our MemCA attacks, which allows the MemCA attacks to effectively bypass the state-of-the-art cloud elasticity mechanisms as shown in Figure 10a and 10b.

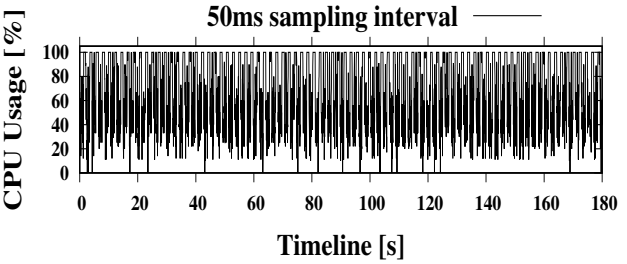
Evaluate MemCA under Cloud Detection of Performance Interference. The detection of performance interference inside the cloud typically requires monitoring low-level metrics such as cache misses and I/O counts, which cannot be measured inside the customers' VMs. Thus, most detection approaches of performance interference are based on the low-level monitoring tools such as Xentrace [34] and OProfile [51] deployed in the host level. Since Amazon EC2 does not allow host level monitoring for normal users, we repeat our above experiments



(a) Using 1-minute monitoring, observing flat CPU usage.



(b) Using 1-second monitoring, showing a little bit fluctuation.



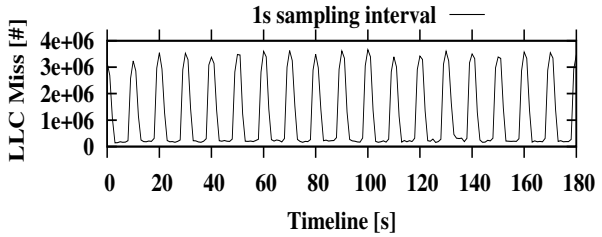
(c) Using 50-millisecond monitoring, observing frequent and transient saturations of CPU uitilization.

Fig. 10: MemCA Stealthiness under Cloud Elasticity.

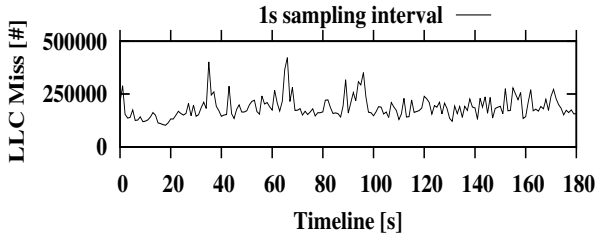
in our private cloud. The setup of our private cloud is introduced in Section III. Figure 2b shows that we are able to achieve similar attack impact in our private cloud as that in the public Amazon EC2 (Figure 2a). Here, we use OProfile to measure LLC cache misses, since we try to intermittently degrade the memory bandwidth or saturate the shared memory bus of the host, and LLC cache misses is the most relevant low level metric with our attacks.

Results. Figure 11 shows LLC cache misses of the physical machine hosting MySQL and its co-located VMs (Host3 in Figure 8) using two memory attack approaches (i.e., saturating memory bus and triggering memory lock) as described in Section III. Due to the on-off pulsating attack styles of MemCA, we can observe periodic LLC misses in the MySQL VM in Figure 11a, which is caused by intermittently saturating the memory bus in the co-located adversary VMs. However, when we adopt the approach of triggering memory lock for launching MemCA, we are not able to observe any obvious pattern of LLC misses in the MySQL VM as shown in Figure 11b, even though the attacker creates periodic attack bursts as shown in Figure 9a.

The above results also reveal the complexity of an effective detection approach. On the one hand, monitoring the wrong metrics will not give us any useful information. On the other



(a) Periodic LLC misses in MySQL VM due to attack bursts of saturating memory bus in co-located VMs.



(b) No obvious pattern of LLC misses in MySQL VM under periodic bursts of triggering memory lock in co-located VMs.

Fig. 11: MemCA Stealthiness under Cloud Detection of Performance Interference.

hand, it is very challenging to pinpoint the right metrics to monitor considering there are only a few hardware performance counters supported by modern CPUs (e.g., Our Xeon E5-2603 CPU only support 4) while the candidate events can reach to hundreds. In addition, using different monitoring granularity to sample the chosen metrics affects the detection effectiveness significantly; for example, in the case of LLC cache miss monitoring, we get similar results as we have observed in Figure 10: coarse granularity monitoring significantly blurs the metric variation while fine-grained monitoring may cause non-trivial overhead that current cloud platforms want to avoid in the first place (e.g., Amazon CloudWatch [1]). Overall, designing effective detection/defense mechanisms again MemCA attacks in the cloud requires significant future research.

VI. Related Work

In this section, we review the most relevant work with regard to memory performance attacks and its corresponding solutions in today's cloud platforms.

Memory Performance Attacks. Memory is one of the key computer resources that is frequently exploited by attackers to degrade the performance of the target system. For example, Moscibroda et al. [40] study the memory performance attacks in multi-core systems where an adversary program can maliciously degrade the memory-related performance of another application running on the same socket. Zhang et al. [60] exploit two types of memory contentions, storage-based contention (LLC cleansing attack) and scheduling-based contention (exotic atomic locking attack), to degrade the performance of applications deployed in clouds. Mehmet et al. [22] attack memory bus/bandwidth, focusing on mobile devices launched by an attack App. Compared to previous brute-force memory attacks, we investigate transient MemCA-based cross-resource attacks (e.g., the tangible damage is on CPU while the attack is on memory). In addition, our attacks focus on n-tier systems, in which the complex dependencies among the distributed nodes make the attacks stealthy while the performance damage is severe, due to the tail response time amplification.

Cross-Resource Contention. Mitigating resource contention in the cloud has been widely investigated [41], [61]. Cross-resource contention (e.g., the relationship between memory bandwidth and CPU in MemCA) is a more difficult problem than the traditional single-resource contention. On the solution side, Heracles [31] exploit multiple resource isolation schemes to avoid/reduce shared resource contention while satisfying their latency critical workloads. Their solution, however, may fail in general cases due to a large variety of resource contentions, especially the cross-resource contentions. Finally, after continuous efforts to achieve high utilization, low interference, and fast response time in cloud data centers, engineers in Google admit that it is almost impossible to achieve all the three at the same time [50]. Our MemCA attacks also comply with such an observation: by imposing transient resource interferences, MemCA attacks are likely to degrade the target system performance, especially the response time.

VII. Conclusion

In this paper, we described MemCA attacks, a new type of low-volume internal DoS attacks in the cloud. Such attacks exploit the sharing nature of public cloud computing platforms by co-locating one or a few adversary VMs with the target VMs that deploy response time sensitive web applications. We showed that MemCA is able to cause intermittent and short-lived cross-resource contentions that lead to performance uncertainty of the target web application (Section II-C). We modeled the attack scenario in n-tier systems based on queuing network theory and analyzed cross-tier queue overflow and response time amplification under MemCA (Section IV-B). To validate the practicality of our attacks, we evaluated our attacks through extensive benchmark experiments in both private cloud and the public Amazon EC2 (Section V), and confirmed the significant damage and the high stealthiness of our attacks. In general, MemCA attacks make an important contribution towards understanding the emerging low-footprint and stealthy DoS attacks in the cloud era.

Acknowledgment

This research has been partially funded by National Science Foundation by CISE's CNS-1566443, CNS-1566388, CNS-1717898, Louisiana Board of Regents under grant LEQSF(2015-18)-RD-A-11, and gifts or grants from Fujitsu. Any opinions, findings, and conclusions are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding agencies and companies mentioned above.

REFERENCES

[1] *Amazon CloudWatch Concepts*. " http://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html".

[2] *Spotify Moves to Google Cloud Platform*. "https://www.infoq.com/news/2016/02/Spotify-Google-Cloud", 2016.

[3] Amazon. *Amazon Auto Scaling*. " https://aws.amazon.com/documentation/autoscaling", 2017.

[4] Amazon. *Amazon EC2 Instance Types*. " https://aws.amazon.com/ec2/instance-types/", 2017.

[5] Amazone. *AWS Elastic Load Balancing*. "https://aws.amazon.com/elasticloadbalancing/", 2017.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. In *Tech. Rep. UCB/EECS-2009-28, University of California, Berkeley*, 2009.

[7] A. Bakshi and Y. B. Dujodwala. Securing cloud from ddos attacks using intrusion detection system in virtual machine. In *Proceedings of Second International Conference on Communication Software and Networks (ICCSN'10)*, pages 260–264. IEEE, 2010.

[8] H. S. Bedi and S. Shiva. Securing cloud infrastructure against co-resident dos attacks using game theoretic defense mechanisms. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pages 463–469. ACM, 2012.

[9] M. Bertoli, G. Casale, and G. Serazzri. Java modelling tools: an open source suite for queueing network modelling andworkload analysis. In *QEST 2006*, pages 119–120, 2006.

[10] H. Chen, Q. Wang, B. Palanisamy, and P. Xiong. Dcm: Dynamic concurrency management for scaling n-tier applications in cloud. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2097–2104. IEEE, 2017.

[11] Clavister. *Clavister DoS and DDos Protection*. "Clavister, Inc.", 2014.

[12] K. Curtis, P. Bodík, M. Armbrust, A. Fox, M. Franklin, M. Jordan, and D. Patterson. *Determining SLO Violations at Compile Time*. 2010.

[13] M. Darwish, A. Ouda, and L. F. Capretz. Cloud-based ddos attacks and defenses. In *Proceedings of 2013 International Conference on Information Society (i-Society)*, pages 67–71. IEEE, 2013.

[14] C. Delimitrou and C. Kozyrakis. Bolt: I know what you did last summer... in the cloud. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 599–613. ACM, 2017.

[15] I. Engineering. Open-sourcing a 10x reduction in apache cassandra tail latency. https://instagram-engineering.com/open-sourcing-a-10x-reduction-in-apache-cassandra-tail-latency-d64f86b43589, March 2018.

[16] L. Engineering. Who moved my 99th percentile latency? https://engineering.linkedin.com/performance/who-moved-my-99th-percentile-latency, April 2018.

[17] 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*, page 22. ACM, 2011.

[18] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang. Reduction of quality (roq) attacks on internet end-systems. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 2, pages 1362–1372. IEEE, 2005.

[19] R. M. Hollander and P. V. Bolotoff. *RAMspeed, a cache and memory benchmarking tool*. " http://alasir.com/software/ramspeed", 2002.

[20] Q. Huang and P. P. Lee. An experimental study of cascading performance interference in a virtualized environment. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):43–52, 2013.

[21] IETF. *RFC 6298*. " https://tools.ietf.org/search/rfc6298/".

[22] M. S. Inci, T. Eisenbarth, and B. Sunar. Hit by the bus: Qos degradation attack on android. In *Asia CCS 2017*, pages 716–727.

[23] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar. Cache attacks enable bulk key recovery on the cloud. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 368–388. Springer, 2016.

[24] S. A. Javadi and A. Gandhi. Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing. In *Proceedings of 2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 135–144. IEEE, 2017.

[25] R. E. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[26] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 51. IEEE Computer Society Press, 2012.

[27] R. Kohavi and R. Longbotham. Online experiments: Lessons learned. *IEEE Computer Society*, 2007.

[28] KVM. *Kernel Virtual Machine*. " https://www.linux-kvm.org/page/Main_Page".

[29] L. Litty, H. A. Lagar-Cavilla, and D. Lie. Computer meteorology: Monitoring compute clouds. In *HotOS*, 2009.

[30] H. Liu. A new form of dos attack in a cloud and its avoidance mechanism. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, pages 65–76. ACM, 2010.

[31] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: improving resource efficiency at scale. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 450–462. ACM, 2015.

[32] A. K. Maji, S. Mitra, and S. Bagchi. Ice: An integrated configuration engine for interference mitigation in cloud services. In *Proceedings of 2015 IEEE International Conference on Autonomic Computing (ICAC)*, pages 91–100. IEEE, 2015.

[33] 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*, pages 277–288. ACM, 2014.

[34] L. man page. *xentrace(8)*. "https://linux.die.net/man/8/xentrace", 2017.

[35] G. Mantas, N. Stakhanova, H. Gonzalez, H. H. Jazi, and A. A. Ghorbani. Application-layer denial of service attacks: taxonomy and survey. *International Journal of Information and Computer Security*, 7(2-4):216–239, 2015.

[36] 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*, pages 248–259. ACM, 2011.

[37] Microsoft. Microsoft azure. https://azure.microsoft.com/en-us/?v=17.14, 2017.

[38] Microsoft. *Microsoft Azure Virtual Machine Sizes for Cloud Services*. " https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general", 2018.

[39] J. Mirkovic and P. Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[40] T. Moscibroda and O. Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security 2007*.

[41] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Annual Technical Conference*, number EPFL-CONF-185984, 2013.

[42] OW2. Rubbos. http://jmob.ow2.org/rubbos.html, 2017.

[43] I. Palo Alto Networks. *Application DDoS Mitigation*. " https://live.paloaltonetworks.com/t5/Tech-Note-Articles/Application-DDoS-Mitigation/ta-p/54531", 2014.

[44] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212. ACM, 2009.

[45] SECURELIST. *DDoS attacks in Q2 2018*. "https://securelist.com/ddos-report-in-q2-2018/86537/", 2018.

[46] H. Shan, Y. Chen, H. Liu, Y. Zhang, x. Xiao, X. He, M. Li, and W. Ding. ϵ-diagnosis: Unsupervised and real-time diagnosis of smallwindow long-tail latency in large-scale microservice platforms. In *Proceeding of the Web Couference-30 years of then web (WWW'19)*, 5 2019.

[47] H. Shan, Q. Wang, and C. Pu. Tail attacks on web applications. In *Proceedings of the 24nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017.

[48] H. Shan, Q. Wang, and Q. Yan. Very short intermittent ddos attacks in an unsaturated system. In *Proceedings of the 13th International Conference on Security and Privacy in Communication Systems*. Springer, 2017.

[49] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris. *Fundamentals of queueing theory*, volume 399. John Wiley & Sons, 2018.

[50] D. Sites. *Data Center Computers: Modern challenges in CPU design*. "https://www.cs.wisc.edu/events/1887", 2017.

[51] O. team. Oprofile. "http://oprofile.sourceforge.net/", 2017.

[52] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *CCS 2012*, pages 281–292.

[53] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift. A placement vulnerability study in multi-tenant public clouds. In *Proceedings of the 24th USENIX Security Symposium*, pages 913–928, 2015.

[54] T. Vissers, T. Van Goethem, W. Joosen, and N. Nikiforakis. Maneuvering around clouds: Bypassing cloud-based security providers. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1530–1541. ACM, 2015.

[55] Q. Wang, C.-A. Lai, Y. Kanemasa, S. Zhang, and C. Pu. A study of long-tail latency in n-tier systems: Rpc vs. asynchronous invocations. In *ICDCS 2017*.

[56] Q. Wang, S. Malkowski, Y. Kanemasa, D. Jayasinghe, P. Xiong, C. Pu, M. Kawaba, and L. Harada. The impact of soft resource allocation on n-tier application scalability. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1034–1045. IEEE, 2011.

[57] Y. Wang, C. Lin, Q.-L. Li, and Y. Fang. A queueing analysis for the denial of service (dos) attacks in computer networks. *Computer Networks*, 51(12):3564–3573, 2007.

[58] Z. Xu, H. Wang, and Z. Wu. A measurement study on co-residence threat inside the cloud. In *USENIX Security*, pages 929–944, 2015.

[59] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys and tutorials*, 15(4):2046–2069, 2013.

[60] T. Zhang, Y. Zhang, and R. B. Lee. Dos attacks on your memory in cloud. In *ASIA CCS 2017*, 2017.

[61] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. Cpi 2: Cpu performance isolation for shared compute clusters. In *EuroSys 2013*, pages 379–391.

[62] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler vulnerabilities and coordinated attacks in cloud computing. *Journal of Computer Security*, 21(4):533–559, 2013.