Security Implications of Data Mining in Cloud Scheduling

Christina Delimitrou and Christos Kozyrakis

Abstract—Cloud providers host an increasing number of popular applications, on the premise of *resource flexibility* and *cost efficiency*. Most of these systems expose virtualized resources of different types and sizes. As instances share the same physical host to increase utilization, they contend on hardware resources, e.g., last-level cache, making them vulnerable to side-channel attacks from co-scheduled applications. In this work we show that using data mining techniques can help an adversarial user of the cloud determine the nature and characteristics of co-scheduled applications and negatively impact their performance through targeted contention injections. We design Bolt, a simple runtime that extracts the sensitivity of co-scheduled applications to various types of interference and uses this signal to determine the type of these applications by applying a set of data mining techniques. We validate the accuracy of Bolt on a 39-server cluster. Bolt correctly identifies the type and characteristics of 81 percent out of 108 victim applications, and constructs specialized contention signals that degrade their performance. We also use Bolt to find the most commonly-run applications on EC2. We hope that underlining such security vulnerabilities in modern cloud facilities will encourage cloud providers to introduce stronger resource isolation primitives in their systems.

Index Terms—Super (very large) computers, security and privacy protection, scheduling and task partitioning, application studies resulting in better multiple-processor systems

1 Introduction

Cloud computing hosts an increasing number of applications in private and public clouds. It offers two main premises to end users and datacenter operators: *flexibility* and *cost-efficiency*. Public clouds expose resources as virtual machines and more recently containers of different types and sizes.

Public cloud schedulers typically colocate several VMs on the same physical machine to increase system utilization. As VMs get co-scheduled they share certain hardware resources, such as the last level cache, or the network switch. This hides security and privacy pitfalls, as resource isolation is not strictly enforced. For example, while memory capacity is partitioned, contention in the last level cache can still leak information about a co-scheduled program. There has been significant related work on side-channel attacks [1], [2], VM detection [3], [4], distributed denial-of-service attacks (DDoS) [2], [5], [6] and data leakage vulnerabilities [1], [7] in cloud providers. Most of these schemes leverage the lack of strictly enforced isolation, and the naming conventions cloud providers use for machines to extract information about a victim VM.

Recent work on datacenter scheduling proposed using data mining techniques to accurately determine the resource preferences of new applications without significant overheads [8], [9], [10]. These techniques improve the decision quality of the scheduler, and as a result per-application performance and cluster utilization increase. This approach relies on the fact that the system has knowledge about applications it has previously-seen which can be mined to understand the preferences of new, unknown applications. Unfortunately in a shared, public cloud infrastructure applying such techniques can have security implications.

In this paper we present Bolt, a practical system that leverages data mining techniques to quickly determine the type and characteristics of any applications scheduled on the same machine in a public cloud. Bolt can determine the type of a co-scheduled victim application, by projecting the contention an adversarial VM experiences, against the large dataset of previously-seen workloads. Second, Bolt uses this information to issue internal denial of service (DoS) attacks to victim applications through specialized interference injection.

We evaluate Bolt on a local cluster and demonstrate that it correctly detects the type of co-scheduled applications for 81 percent out of 108 diverse workloads. Additionally, it is able to degrade the performance of those workloads, by 32 percent on average and more than 60 percent in some cases, by injecting interference in the resources a victim application is sensitive to. It does so without saturating compute and/or memory utilization, which could trigger migration and autoscaling in cloud providers that offer such solutions to remedy performance unpredictability.

2 Related Work

We discuss related work with respect to VM placement detection, DDoS attacks, side-channel attacks and data extraction.

VM placement detection. Ristenpart et al. [3] show how leveraging the IP naming conventions of machines in cloud providers can help an adversarial user pinpoint where a victim VM is residing in a large-scale cluster. Subsequently, the adversarial user can launch VMs until one is co-scheduled on the same physical machine as the target VM. HomeAlone [4] tracks the utilization of the L2 cache during periods of low traffic from "friendly" VMs to detect whether the physical machine is shared across VMs.

DDoS attacks. Distributed Denial of Service attacks [11], [12], [13] in the cloud have increased significantly in number and impact over the past few years. This has generated a lot of interest in detection and prevention techniques [14]. Gupta and Kumar [5] outline the characteristics of cloud facilities that make DDoS attacks more likely, discuss the challenges that current DDoS prevention schemes face, and propose a scheme based on VM profiling to detect network DDoS attacks. Bakshi and Yogesh [6] develop a system that detects abnormally high network traffic in cloud machines that would signal an upcoming DDoS attack. Finally, Darwish et al. [15] explore different DDoS attack types in cloud resources, and propose practical defense mechanisms.

Side-channel attacks in public clouds. Such systems attempt to extract information about co-scheduled applications, including confidential data, such as private keys [16], [17], [18], [19]. Zhang et al. [7] describe a system that launches side-channel attacks in a virtualized environment. The system overcomes three main challenges: the frequent re-scheduling of VMs by a hypervisor or cluster scheduler, the noise in shared resource usage and the implications introduced by core migrations. They demonstrate that the system can extract an ElGamal decryption key from a victim VM. Wang et al. [2] specifically target intrusion detection in cloud settings, while Liu et al. [1] design a scheduling system that protects against covert channels in resources such as the memory bus in a cloud environment. The system controls the overlapping execution of different VMs and injects noise on the memory bus to prevent the extraction of confidential information by an adversarial user.

3 Bolt

3.1 Threat Model

We consider an IaaS provider that operates a public cloud. Multiple VMs can be co-scheduled on the same physical server. Each VM has no control over where it is placed in the cluster, and has no a priori information on any other VMs scheduled on the same host. We assume that the cloud provider is neutral with respect to VM detection by an adversarial VM, i.e., it does not employ any

• *The authors are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305. E-mail: {cdel, kozyraki}@stanford.edu.*

Manuscript received 20 May 2015; revised 20 July 2015; accepted 22 July 2015. Date of publication 26 July 2015; date of current version 5 Jan. 2017.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2015.2461215

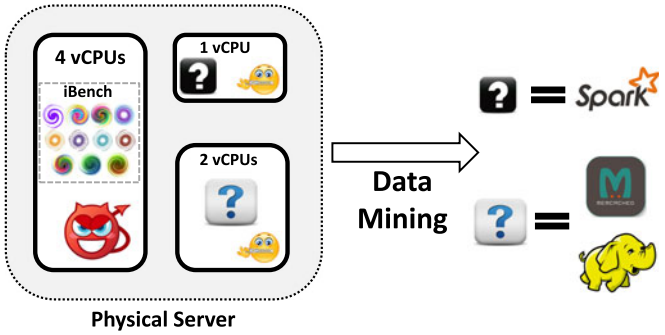

Fig. 1. Overview of the system's operation. The adversarial VM uses iBench to measure the pressure the two victim VMs place on shared resources. Bolt then uses data mining to determine the type and characteristics of applications running in the victim VMs.

additional resource isolation techniques than what is available by default to hinder such attacks.

Adversarial VM. An adversarial VM has the goal of determining the nature and characteristics of jobs co-scheduled on the same physical host, and negatively impacting their performance.

Friendly VM. This is a VM scheduled on a physical host that runs one or more applications. Friendly VMs do not attempt to determine the existence and characteristics of other co-scheduled VMs. They also do not employ any schemes to prevent detection.

3.2 Application Detection

The operation of Bolt at a high level is shown in Fig. 1. The adversarial VM runs a set of microbenchmarks that have tunable intensity and put progressively more pressure in a specific shared resource each [20]. Examined shared resources include the different levels of the cache hierarchy, the memory system, the core, and the network and storage subsystems. Once a microbenchmark starts running it will increase its intensity until it finds pressure from co-scheduled workloads, i.e., until its performance is lower than its expected value, when it is running in isolation. The intensity of the microbenchmark at that point is the pressure the co-scheduled applications put in shared resource i and is denoted by c_i, where $i \in [1, N]$, $N = 10$. Large c_i values mean that the co-scheduled applications put a lot of pressure in resource i. The same operation is performed for two-three microbenchmarks, and requires 5-10 seconds.

This profiling signal is used by an online recommender system that derives the resource pressure of the victim VMs in the remaining shared resources. Missing pressure scores are determined using singular value decomposition (SVD) and PQ-reconstruction with Stochastic Gradient Descent (SGD) [9], [10]. The recommender system then finds similarities between the new victim VM and previously-seen VMs by extracting similarity concepts, such as the memory pressure applications create, the type of compute operations they perform, or the network/storage access patterns they launch. SVD produces three matrices, U, Σ and V. Matrix U captures the correlation between each application and each similarity concept. Using Pearson correlation coefficients on the U_i vectors of different applications we can determine which application types a new, unknown application is similar to. The output is a distribution of confidence scores of how closely the victim VM resembles different types of previous applications. For example a victim VM may be 65 percent similar to memcached applications, 18 percent similar to Spark jobs, 10 percent similar to Hadoop jobs running SVM classifiers, and 3 percent to Hadoop jobs running k-means.

There are cases where a victim VM may not have a clear application type it resembles, or may change behavior during its execution. To address this issue Bolt repeats the classification periodically (every 5 minutes in our experiments), until the confidence scores of similarity converge. For the majority of examined applications, convergence occurs after two iterations.

Multiple co-scheduled jobs. A challenge with the previous approach is the case where more than one victim VMs are co-scheduled on the same physical server. The adversarial VM has no access to the hypervisor, and hence only sees the aggregate interference from all co-scheduled applications. To decouple different jobs sharing a physical server, Bolt examines the combinations of interference profiles of different application types, and compares them against the aggregate interference observed by the adversarial VM. For example, if we have only seen three types of applications, A_1, A_2 and A_3, with U vectors: $[u_{11}, u_{12}, \ldots, u_{1N}]$, $[u_{21}, u_{22}, \ldots, u_{2N}]$ and, $[u_{31}, u_{32}, \ldots, u_{3N}]$ and the victim VM has interference profile: $[x_1, x_2, \ldots, x_N]$, Bolt will examine all combinations of the three application types, and return the one that most closely resembles the victim VM. We plan to investigate solutions that scale better and detect non-linear interference relations in future work.

3.3 Internal Denial of Service Attack

Once an adversarial VM knows the type of applications that coexist on the same physical host, it tries to degrade their performance through a targeted interference injection. Constructing an interference signal that will affect the victim VM relies on the knowledge of the resources the victim VM is sensitive to. This corresponds to the level of interference the application can tolerate in the different shared resources. The injected contention is then simply an interference signal that slightly exceeds that level. For example, in the case of memcached, the detection scheme determines that the victim VM puts a lot of pressure in the LLC, followed by lower pressure in the CPU, memory and network subsystems. This means that the application tolerates a lot of contention in storage, followed by lower tolerance in the memory, network and CPU. Note that tolerated interference is not simply the complement of the generated contention of a workload.

Injecting this contention signal degrades the performance of the victim VM with high probability. Furthermore, because the injection does not simply saturate shared resources, e.g., by scanning the entire cache/memory or saturating the CPU, it will not trigger the migration/autoscaling schemes that cloud providers have in place to avoid machine oversubscription when application load increases. As a result, performance degrades without the user being able to ameliorate the problem, except by terminating and restarting the VM.

In the following section we evaluate the accuracy of Bolt in detecting the type and characteristics of victim VMs, and the performance degradation it can induce to them.

4 Evaluation

4.1 Controlled Experiment

We first perform a controlled experiment, in which all victim applications are known in advance. This allows us to validate the accuracy of Bolt in detecting application types and degrading their performance through targeted contention injection.

We use a cluster of 39 dedicated machines, with eight two-way hyperthreaded physical cores and 64 GB of RAM each. In each server we launch a two physical core (of two vCPU each) VM running Ubuntu 14.04 as the adversarial VM. The remainder of the machine is allocated to one or more victim VMs, running over Ubuntu 14.04 or Debian 8.0. Victim applications are scheduled using the Quasar cluster manager [10]. Quasar minimizes interference between co-scheduled workloads by only colocating applications that do not contend in the same resources. This helps quantify the impact of the adversarial VM on performance, and decouple it from any existing interference between co-scheduled applications. We will explore how different schedulers affect the detection accuracy as part of future work. The adversarial

TABLE 1
Estimation Accuracy for the Controlled Experiment

Applications	Correct app type	1st incorrect estimation	2nd incorrect estimation
Aggregate	81%	-	-
memcached	76%	Spark	speccpu2006
Hadoop	84%	Cassandra	Spark
Spark	85%	speccpu2006	memcached
Cassandra	87%	Hadoop	Spark
speccpu2006	81%	Spark	memcached

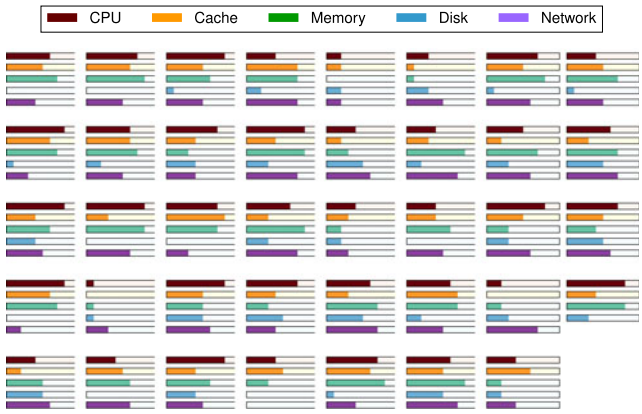

Fig. 2. Per-server resource pressure in the controlled experiment.

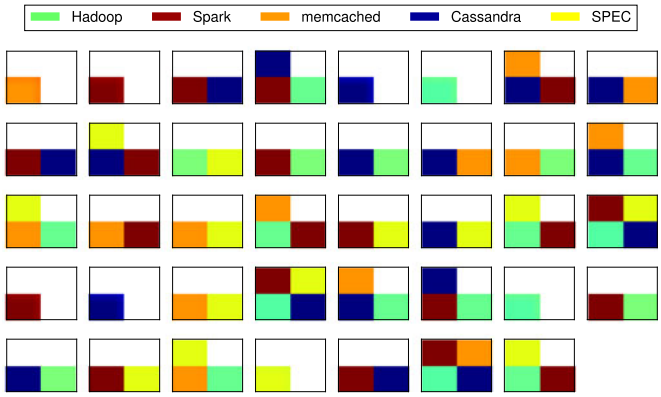

Fig. 3. Per-server application mapping in the controlled experiment.

applications have no a priori information on the number and type of co-scheduled applications. We schedule a total of 108 applications, from five classes (Table 1). Within each class there are several individual workloads, including the Mahout library for Hadoop and machine learning applications for Spark. Each machine has at least one victim VM, and at most four victim VMs. Each VM can use one or more physical cores. When exceeding four applications the interference between the victim applications alone does not permit them to meet their QoS constraints. Fig. 3 shows the type of applications running in each physical machine, and Fig. 2 shows the level of interference they induce in each of the main shared resources (from 0 to 99 percent), as detected by Bolt. Note that different mixes of applications produce different interference profiles in Fig. 2. Based on this signal Bolt tries to identify the type of co-scheduled applications. Table 1 shows Bolt's detection accuracy, per application type, and aggregate. Bolt correctly identifies the majority of jobs, 81 percent, and for certain application types like databases and analytics, the accuracy exceeds 85 percent. 86 percent of correctly identified applications required one profiling run for identification. For an extra 9 percent, a second profiling

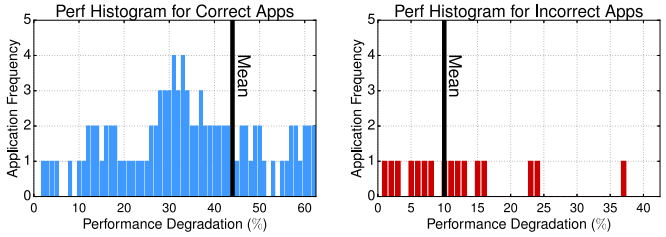

Fig. 4. Performance degradation in the controlled experiment.

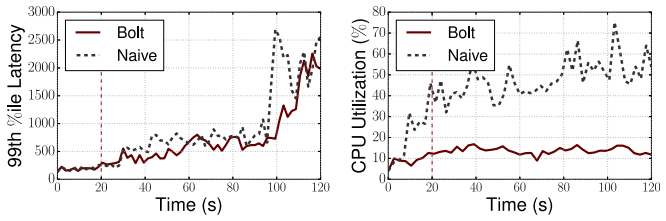

Fig. 5. Comparison of latency and utilization against a naïve scheme.

TABLE 2
Breakdown of the Number of Co-scheduled VMs Found by Bolt in the EC2 Experiment

	no VM	1 VM	2 VMs	$>$ 2 VMs
% of machines	41%	32%	16%	11%

run was necessary. We also show the most frequent misconceptions for each job type. Most incorrectly-identified applications occur in servers hosting more than three workloads.

We also examine the performance degradation Bolt can introduce through contention injection. Fig. 4 shows the PDF of degradation in execution time across correctly—(left) and incorrectly—identified applications (right). Correctly-identified jobs experience substantial performance degradation, since the system is able to determine the resources they are most sensitive to. On average there is a 32 percent degradation in execution time, and 62 percent in the worst case, while tail latency increases by 10× for applications such as memcached (Fig. 5a). On the other hand, the small number of incorrectly-identified applications only experience modest performance degradation, since the system introduces contention in the wrong resources.

If interference translates to CPU or memory saturation, there is a high probability that at least one of the co-scheduled VMs will be migrated to a new physical machine, for cloud providers that support live migration, e.g., Google Compute Engine, or scaled out to additional machines. Therefore it is critical to ensure that interference and system utilization are almost orthogonal. Fig. 5 compares the tail latency and CPU utilization Bolt causes to that of a naïve system that simply saturates the CPU through a compute-intensive kernel. We focus on a single victim VM running memcached. Performance degradation is similar for both systems as time progresses. On the other hand utilization is quite different, with Bolt keeping CPU utilization fairly low, hence not triggering migration or autoscaling, while the naïve scheme quickly saturates the core.

4.2 Bolt in the Wild

We now use Bolt in a real (non-controlled) setting on a large EC2 cluster to detect the type of applications submitted by external users. This experiment is limited to detecting, but not negatively impacting the performance of the co-scheduled applications, to prevent service interruptions for other users. We request 400 four vCPU on-demand instances, and verify that they are not on the same physical machine [21]. We inject iBench to determine the interference profile of the co-scheduled application(s) and use the

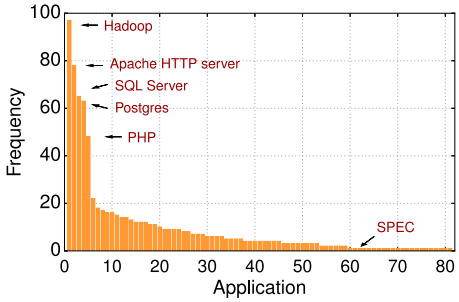

Fig. 6. Probability distribution function of application types on EC2.

classification engine to map interference profiles to specific application types. Table 2 shows how many co-scheduled applications were found across the 400 physical machines. For a large fraction of physical machines the adversarial VM is the only application occupying the server, despite the fact that it only requires FOUR vCPUs. Most of the remaining machines have one-two co-scheduled applications, while a small number of servers host more than three victim applications. We keep requesting four vCPU instances until we have 400 instances with at least one co-scheduled VM. Fig. 6 shows the probability distribution function (PDF) of detected application types. Interestingly there is a very small number of application types that dominates the utilization of the examined cluster. As expected, these applications primarily include analytics, webservers and databases. Apart from the main five applications, there is a long tail of less common applications, which were encountered less than 10 times across all VMs.[1]

5 Future Work

We have presented Bolt, a system that highlights the performance implications of using data mining in cloud settings. Bolt projects the contention an adversarial VM experiences in shared resources from co-scheduled VMs, against a dataset of application profiles to determine the type and characteristics of co-scheduled VMs. Furthermore, it negatively impacts victim applications, through targeted contention injections. Such attacks are facilitated by the lack of strict resource isolation guarantees between workloads sharing a physical machine. Introducing isolation techniques, primarily in the memory hierarchy which is a strong indicator of the type of co-scheduled applications could alleviate some of these security concerns.

References

[1] F. Liu, L. Ren, and H. Bai, "Mitigating cross-VM side channel attack on multiple tenants cloud platform," *J. Comput.*, vol. 9, pp. 1005–1013, 2014.
[2] H. Wang, H. Zhou, and C. Wang, "Virtual machine-based intrusion detection system framework in cloud computing environment," *J. Comput.*, vol. 7, pp. 2397–2403, Oct. 2012.
[3] 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 *Proc. 16th ACM Conf. Comput. Commun. Security*, Chicago, IL, USA, 2009, pp. 199–212.
[4] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proc. IEEE Symp. Security Privacy*, 2011, pp. 313–328.
[5] S. Gupta and P. Kumar, "VM profile based optimized network attack pattern detection scheme for DDoS attacks in cloud," in *Proc. Int. Symp. Security Comput. Commun.*, 2013, pp. 255–261.
[6] A. Bakshi and B. Yogesh, "Securing cloud from DDOS attacks using intrusion detection system in virtual machine," in *Proc. 2nd Int. Conf. Commun. Softw. Netw.*, 2010, pp. 260–264.
[7] Y. Zhang, A. Juels, et al., "Cross-VM side channels and their use to extract private keys," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 305–316.
[8] C. Delimitrou and C. Kozyrakis, "The Netflix challenge: Datacenter edition," *IEEE Comput. Archit. Lett.*, vol. 12, no. 1, pp. 29–32, Jan.-Jun. 2013.
[9] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," in *Proc. 18th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2013, pp. 77–88.
[10] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *Proc. 19th Int. Conf. Archit. Support Program. Languages Operating Syst.*, 2014, pp. 127–144.
[11] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, pp. 39–53, Apr. 2004.
[12] B. Farley, V. Varadarajan, et al., "More for your money: Exploiting performance heterogeneity in public clouds," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 20:1–20:14.
[13] V. Varadarajan, T. Kooburat, et al., "Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense)," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 281–292.
[14] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Comput. Surv.*, vol. 39, no. 1, p. 1:42, Apr. 2007.
[15] M. Darwish, A. Ouda, and L. Capretz, "Cloud-based DDoS attacks and defenses," in *Proc. Int. Conf. Inf. Soc.*, Toronto, ON, Canada, 2013, pp. 67–71.
[16] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 827–838.
[17] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 605–622.
[18] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, l3 cache side-channel attack," in *Proc. USENIX Con. Security Symp.*, 2014, pp. 719–732.
[19] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, "Ooh aah... just a little bit. : A small amount of side channel can go a long way," in *Proc. 16th Int. Workshop Cryptographic Hardware Embedded Syst.*, 2014, pp. 75–92.
[20] C. Delimitrou and C. Kozyrakis, "iBench: Quantifying interference for datacenter workloads," in *Proc. IEEE Int. Symp. Workload Characterization*, 2013, pp. 23–33.
[21] V. Varadarajan, Y. Zhai, et al., "Scheduler-based defenses against cross-VM side-channels," presented at the 23rd Usenix Security Symp., San Diego, CA, USA, 2014.

1. Anecdote: By verifying the timestamps, instance configuration and zone, and benchmark order, we were able to identify one victim VM running several SPECCPU2006 benchmarks which belonged to a student group at Stanford.