# VIRTUAL MACHINE MONITORS IMPROVE YOUR CLOUD PERFORMANCE

**C.Titus**[*]

**S.Aparna**[**]

***Abstract-***

Cloud computing is quickly becoming the platform of choice for many web services. Virtualization is the key underlying technology enabling cloud providers to host services for a large number of customers.Effective resource management for shared storage systems is challenging, even in research systems with complete end-to-end control over all system components.We evaluate the effectiveness of our implementation using quantitative experiments, demonstrating that this approach is practical.In this paper, we presented a novel I/O workload based performance attack which uses a carefully designed workload to incur significant delay on a targeted application running in a separate VM but on the same physical system. Such a performance attack poses an especially serious threat to data-intensive applications which require a large number of I/O requests.Hence, no hypervisor is needed to allocate resources dynamically, emulate I/O devices, support system discovery after boot up, or map interrupts and other identifiers.

***Keywords*—Cloud computing,virtualization,scheduling**

[*] Assistant Professor, Department of Computer Science and Engineering, Mahendra College of Engineering, Minnampalli, Salem – 636106

[**] PG Scholar, Department of Computer Science and Engineering, Mahendra College of Engineering, Minnampalli, Salem – 636106

## I. INTRODUCTION

Usually the Cloud is used to transfer the information from one to another by the help of the proxy server Cloud. In that way in this we transfer the information from on to another easily. Now-a-days Transfer Information has become very unsecure . The cloud information has been easily accessed by the hacker. The Hacker can view the personal information of a person without any authorization. To overcome come this we Implement secure method in this .  We use to send the file in encrypted format but it is not secure to maintain in the cloud to over come this we had split the file into multiple and save them in the cloud location. In this work, we design and implement Swiper, a framework that exploits the virtual I/O vulnerability in three phases: 1) co-location ("sneaking-up"): place the adversary VM on the same physical machine as the victim VM; 2) synchronization ("getting-ready"): identify whether the targeted application is running on the victim VM and, if so, the state of execution for the targeted application

(which we shall elaborate below); and 3) exploiting ("swiping"): design an adversarial workload according to the state of the victim application, and launch the workload to delay the victim.

The main contribution of this paper are listed as follows :

• An I/O-based co-location detection technique and verified its effectiveness on public clouds.

• A discrete Fourier transformation (DFT) based algorithm which recovers the victim's original I/O pattern from the observed (distorted) time-series of I/O throughout, and then determines if the victim application has reached a pre-determined point when it is most vulnerable to an exploitation.

• Discover patterns which cause maximum interference.

• A theoretical framework to observe and synchronize with predefined I/O patterns.

• A comprehensive set of experiments on Amazon EC2 - with the results clearly showing that Swiper is capable of degrading various server applications by 22.54% on average (and up to 31%) for different instance types and benchmarks, while keeping the resource consumption to a minimum.

According to this paper it mainly introduces the system and threat models in sec. 2.It presents I/O-based co-location detection method in sec. 3.In sec.4 it describes our approach,and explains the synchronization and exploiting stages.Sec.5 discusses issues in practicing Swiper.Sec.6 presents the experiment and results.we conclude in sec.7

## II. THREAT MODEL

In general, a cloud computing system provides its endusers with a pool of virtualized computing and I/O resources supported by a large amount of distributed, heterogeneous,commodity computers. For example, Amazon EC2 is using Xen virtualization, whose architecture and terminology are described in Appendix A.For I/Os, VMs utilize the device drivers (the frontend drivers) in the guest OS to communicate with the backend drivers in DOM0, which access the physical devices, e.g., hard drives and networks, on behalf of each VM.In other words, application I/Os within a VM – which basically consist of block reads and writes to the virtual disks - are translated by the virtualization layer to system calls in the host OS, such as requests to the physical disks. In Xen, the hypervisor and DOM0 work together to ensure security isolation and performance fairness among all VMs. While fairness in CPU and memory virtualization is relatively easy to achieve, in this paper we show that maintaining performance isolation for virtual I/O can be extremely challenging - which opens doors for security threats.In this work, we also evaluate our framework on KVM (Kernel-based Virtual Machine) that utilizes hardware assisted full virtualization instead of Xen's paravirtualization.Although Xen and KVM are used to demonstrate this threat in our work, our test and previous work indicate that other virtualization framework like VMware also exhibits similar interference problem [21].

provided, and in [4], a scalable distributed media transcoding system that can reduce the transcoding time is presented. In queue waiting time of transcoding servers is used to make an admission control for video streams and job dispatching for video transcoding to prevent jitters. In virtual machine provision for video transcoding, is considered as the cost-efficient. In [5], mechanisms for allocation and reallocation of virtual machines and video transcoding servers are provided.

## III. PROBLEM DEFINITION

A straightforward way to delay a victim process is to launch an attacking process which constantly requests a large amount of resources shared with the victim (e.g., I/O bandwidth). Nonetheless, such an attack can be easily detected and countered (e.g., a dynamic resource allocation algorithm can restrict the amount of resourcesobtained by each process). Thus, our focus in this paper is to incur the maximum delay to the victim while maintaining the resource request from the attacker to a pre-determined (low) threshold.

**Prior Knowledge of the Adversary**:

Since the adversary now has to target the attack specifically to the victim process (instead of blindly delaying all processes sharing the resource), it has to possess certain characteristics of the victim process which distinguishes it from others. For the purpose of this paper, we consider the case where the adversary holds the trace of resource requests from the victim process as the “fingerprint” of the victim.

Research on cross-VM side channels can be used to sustain this assumption [22]–[25] - malicious VMs are able to retrieve a variety of information, such as data and instruction caches, I/O usage profile, and even private keys, from co-located VMs and hosts via side channels.The techniques for co-location detection in Sec. 3 can also be adapted to profile I/O access patterns as well. We plan to extend the profiling technique as future work. In the experiment section, we shall demonstrate that the various workloads we tested all exhibit unique resourcerequest time-series that can be easily distinguished from others.

**Limits on the Adversary**:

Many cloud computing systems charge by the amount of resource requests. For example, Amazon Elastic Block Store (EBS) charges $0.1 - $0.11 per 1 million I/O requests and Amazon EC2, on the other hand, charges by total network consumption - i.e., the amount of data transferred in and out of the system [26]. Thus, the adversary must minimize the amount of resource request initiated by itself. In this paper, we consider a pre-determined upper bound on the total resource consumption by the adversary.

**Problem Statement:**

Given a workload fingerprint of a victim process, determine an adversarial workload of I/O request which incurs the maximum delay on the victim process without exceeding the pre-determined threshold on the adversary's own resource consumption.

## I/O BASED CO-LOCATION DETECTION

In this work, we use Amazon EC2 as one testing platform to carry out experiments. As we focus on vulnerability with competition for I/O resources, we choose two types of Amazon EC2 instances, micro and small, to be the experiment instance types. Please refer to Appendix B for the introduction of Amazon EC2. The co-location detection mainly consists of two stages: Probing and Locking-on.

**Probing:**

An adversary can locate the geographical zone of a victim process by the victim's IP information [6]. To conveniently manage separate networks for all availability zones, Amazon EC2 partitions internal IP address space between availability zones. Administration tasks will be more difficult if the internal IP address mapping changes frequently. Because different ranges of internal IP address represent various availability zones and public IP addresses can be mapped to private IP addresses by DNS, an adversary can easily locate the availability zone of a victim, thus greatly reduce the number of instances needed before achieving a collocation placement.

Once an adversary knows the availability zone of avictim, it uses network probing to check for the coresidence.In general, if an adversary and a victim are co-located, they are likely to have 1) identical DOM0 IP address, and 2) small packet round-trip times (RTT).

Therefore, an adversary can create several probing instances to perform a TCP SYN traceroute operation to a victim's open service port. If one probing instance and the victim were co-located, they would share the same DOM0 and there would be only a single hop to the victim with a small RTT. In our experience, if the RTT is smaller than half of the average RTT of all onehop instances in the same zone, the probing instance is very likely on the same physical machine as the victim.

**Locking-on:**

Co-location on the same physical machine does not necessarily mean the sharing of the I/O resources - co-located VMs may end up using different storage types. In our tests, if two co-located VMs do not share one hard drive, launching a workload to compete for I/O resources shows limited effect on I/O throughput. On the other hand, if two instances share the same storage device and both try to max out the bandwidth,they can only get part of the total bandwidth. Prior
works also have shown similar interference effect in virtualized environments [27]–[29].Because the adversary knows its performance under a given I/O workload, for it to confirm the I/O sharing,it needs a VM instance that would potentially co-locate with the victim and try to compete for I/O resources. The adversary then can simply measure the I/O performance and an obvious performance degradation would be a strong indicator of VM co-location.

## IV. SWIPER FOR A TWO-PARTY SYSTEM

There are two critical challenges for incurring the maximum delay to a victim-synchronization and adaptive attack.

**Synchronization:**

In order for the adversary to incur the maximum delay under a resource constraint, it has to be able to (1) determine whether the victim process is running, and (2) predict the resource request from the victim process at a given time.

**Adaptive Attack:**

Based on the result of synchronization, the adversary should align its resource request (i.e., attack) with the victim. In general, the higher demand the victim has at a given time, the larger request the adversary should submit to the shared resource.

**Ideas for synchronization:**

In this paper we consider a simple adversarial strategy of conducting an observation process with a sequential read operation. We chose read over write because the timeseries of throughput allocated to write operations tend to have sharp bursts, which would make the synchronization significantly more difficult. Both sequential and random reads in our tests yield similar results in terms of the accuracy of synchronization. We chose sequential read over random read because the latter is rarely the behavior of a normal user and therefore may be detected by the cloud computing system.Before describing the details for synchronization, we first introduce a few basic notions: Recall that the adversary holds as prior knowledge of the I/O request time series from the victim (when no other process is
running). Let v(t) be the bandwidth requested by the victim at t seconds after the victim starts running. At run-time, let tob (seconds) be the length of the observation process (where ob stands for observation length) and a(t) (t 2 [1, ob]) be the (observed) throughput allocated to the adversary for the t-th second since the observation process starts. Let aU be the (upper bound on) throughput for the sequential read operation when no other process is running.The objective of synchronization is for the adversary to align the pre-known v(t) with the observed timeseries aU − a(t). In the ideal case, aU − a(t) would be a concatenation of two sub series: one with zero readings (i.e., when the victim has not yet started running or has finished running), and a sub-sequence of v(t). In practice,however, additive noise and rescaling on both time and throughput may be applied, leading to a requirement on aligning v(t) with aU−a(t) with offset, stretching, and scaling factors. Appendix C provides the complete definition of these three factors, and Appendix D explains our main theory for addressing the challenge of synchronization.

Performance Attack

Based on the result of synchronization, we consider a performance attack which launches multiple segments of sequential read operations to delay the victim process.Each segment persists for a fixed, pre-determined,amount of time. In the following, we discuss three critical issues related to the design of such a performance attack: (1) when should each segment be launched, (2) how long should each segment persist, and (3) why should each segment use a sequential read operation.

**Positioning of Attack Segments:** To incur the maximum delay to the victim process, the attack segments should be positioned to cover the moments of peak requests from the victim process. Thus, to position h attack segments each persisting for ` seconds, we use a greedy algorithm which first locates the `-second interval in v(t) which has not yet been executed and has the maximum total request, i.e., finds the start of interval tS 2 [ob−to↲,N−1]

such that

$$t_S = \arg\max_t \sum_{i=t}^{t+\ell} v(i), \qquad (1)$$

and then repeat this process after removing interval [tS, tS + `] from consideration, until all h intervals are found. Note that there must be tS # ob − to↲ because by the end of the observation process, the first ob − to↲ seconds of the victim process have already passed and thus cannot be attacked.

**Length of Attack Segments:**

Somewhat surprisingly, our experiments (as we shall present in Sec. 6) show that as long as each attack segment covers a peak of the victim’s request, the length of the attack segment does not have a significant impact on the delay incurred to the victim process. Intuitively, this is because the length of the attack which does not overlap with the peaks of victim’s request incurs little delay to the victim. Nonetheless, this does not mean that the adversary should set each attack segment to be as short as possible - Instead, it has to take into account the estimation error of synchronization,and make the attack segment long enough to ensure the coverage of the peaks.

**Operations of Attack Segments:**

Each attack segment may perform four types of operations: equential read,random read, sequential write, and random write. We choose the sequential read operation due to the following reasons.First,we excluded the write operations from consideration for the same reason as that discussed for the design of the observation process: write perations tend to introduce sharp bursts on throughput, which makes it difficult to be synchronized with the victim's peak requests. We chose sequential read over random read because a random read operation is unlikely to sustain a high throughput to "compete" with the victim process and delay it.

One note of caution is that, while each attack segment should erform a sequential read operation, the adversary must ensure that consecutive (but different) attack segments do not read sequentially on adjacent blocks.This is because the hard drive or operating system may pre-fetch the latter blocks while performing the previous attack segment. As a result, the latter attack segment does not actually incur any I/O to the hard drive,incurring no delay on the victim process. To address this issue, a simple attack strategy is for each segment to first randomly choose one from a set of files, and then read the file sequentially.

Figure 1 shows an example trace when Swiper issues overlapping sequential read operations to slowdown an co-located FileServer. When comparing Figure 1 with the unaffected trace, we found Swiper issues most I/O operations when victim issues as well. In addition, victim's trace has been obviously distorted to certain degree. We will further analyze the performance decreases in Sec. 6.

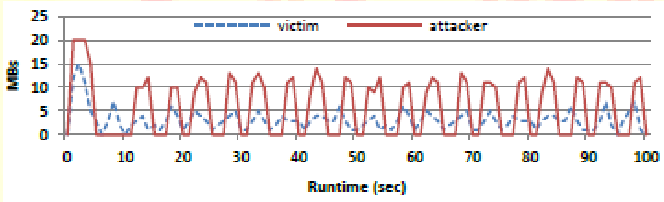

Fig. 1: Overlapping I/O of an attacker and a victim

V. PRACTICAL ISSUES IN RUNNIG SWIPER

We have established a framework to locate and interfere with target VMs, including a theory for syn chronizing I/O patterns. There are critical issues that need to be addressed when deploying

Swiper in realworld.We explicitly discuss two important factors in this section. First, some applications' activities depend on user inputs. Thus, we talk about how to deal with such non-determinism in Sec. 5.1. Second, migration is an important feature for virtualized systems to manage resources. Co-locating the target and attacker is critical in the proposed method. Since the target VM could be migrated thereafter, we discuss migration in Sec. 5.2.

**Non-Deterministic User Behavior**

Some applications' activities, e.g., Twitter andWikipedia,are generated by users. Although one person may not repeat the same behavior hour after hour and day after day, a recent study on a Twitter trace revealed that aggregate workload demonstrates much more predicable I/O activities than single user's, i.e., similar aggregate I/O activities in one hour may occur at the same hour tomorrow and next week [30]. A previous analysis on long-term traces from Amazon Web Services and Google App Engine also found yearly and daily patterns [31].Although historical traces could help in predicting I/O behaviors, self-learning and adaptivity to new I/O patterns are still good to have in a fast-changing world. Swiper can be easily extended to deal with nondeterministic workloads by integrating with a pattern repository and learning module. Figure 2 demonstrates a high level sample architecture of an extended Swiper.

With this extended design, Swiper can adjust various

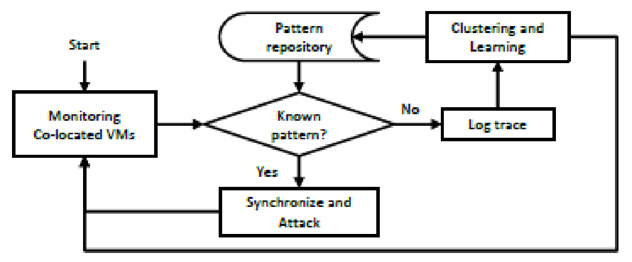

Fig. 2: An extended Swiper architecture for dealing with non-deterministic workloads

parameters, e.g. the stretching factor, and capture more patterns to improve its success rate. We examine Swiper with non-deterministic workloads in Sec. 6.2. Note that designing clustering and learning methods for Swiper may by itself a new research topic. Thus we leave them as future works.

## VM Migration

While live migration is a possible method of mitigating the interference from co-located workloads without service interruption, it does not come without a price.Indeed, previous work have shown that the performance may be substantially affected during migration [32]–[34]. For I/O-intensive applications in particular, since data can be stored or cached on high performance local storage to reduce the access latency, VM migration can be even more costly - lasting several minutes to hours depending on the size of VM virtual storage that is stored locally.

Alternatively, a practical method for reducing the migration time is to only migrate the computing instance (CPU and memory states) and keep VMs virtual disks on networked storage. For such setting, our experiments in Sec. 6.3 show that it is possible for Swiper to locate and impede the target VM again. In this case, a critical problem for the adversary is the cost because now the attacker needs to launch a number of probing VMs to search for the target after the VM migration. Note that this cost can be minimal in the cases that the adversary already held many hacked user accounts, which had
happened before - e.g., in [35].

## VI. EXPERIMENT RESULTS

Because a substantial portion of Amazon EC2’s address space hosts publicly accessible web servers [6], we test Swiper with the following popular cloud applications or benchmarks: **YCSB** (Yahoo! Cloud Serving Benchmark) is a performance measurement framework for cloud data serving [36]. YCSB’s core workload C is used to emulate read-intensive applications; **Wiki-1** and **Wiki- 2** are running Wikibench [37] with real Wikipedia request traces on the first day of September and October 2007 respectively; **Darwin** is an open source version of Apple’s QuickTime media streaming server; **FileServer** mimics a typical workload on a file system, which consists of a variety of operations (e.g., create, read, write, delete) on a directory tree; **VideoServer** emulates a video server, which actively serves videos to a number of client threads and uses one thread to write new videos to replace obsolete videos; **WebServer** mostly performs read operations on a number of web pages, and appends to a log file. The FileServer, VideoServer and WebServer belong to the FileBench suite [38]. Micro and small Amazon EC2 instances and a local machine are used as the test platforms in this work. We use technique described in Sec. 3 to locate Amazon EC2 instances,which dwell in the same storage device. The tests are repeated for 50 times and the means are reported. To evaluate the effectiveness of an attack, we define three metrics: 1) the slowdown/decrease in percentage of the victim, S, which assesses the overall effect of an attack. This can be measured as the runtime in seconds or the throughput in KB. 2) the victim slowdown divided by the total runtime (in seconds) of the attacker, SAT ,which determines the impact of the length of an attack. A bigger SAT indicates that an attacker can infiltrate large damages within a shorter time window. 3) the victim slowdown divided by the total throughput (in MB) of the attacker, SAC, which evaluates the effect of the bandwidth consumption of an attacker. A bigger SAC means that an attack is effective while consuming a smaller amount of bytes.

**Dealing with Non-determinism**

This section demonstrates how Swiper works as a pattern detection method with a repository of collected patterns to cope with user randomness (see Figure 2). The results and analyses indicate that Swiper could help in accomplishing an automatic attacking framework. As a prototype implementation, the pattern store consists

of pre-stored 120 one-minute Wikipedia traces which are from 9 to 11 am on the 1st of October, Monday,2007. Then, we replay a 24-hour trace on the same day to evaluate how Swiper reacts to the trace. Note the pattern here is the time and amount of bandwidth usage by the target. Since we do not use any advanced pattern learning module (which by itself may become a separate research topic), we relax the scaling and stretching factors by 10% to allow Swiper to accept similar patterns in the 24-hour testing set. If there are more than one matched patterns due to the relaxation, the one with the least distortion will be selected. When Swiper identifies a known pattern in a one-minute interval, it will synchronize with and attack the victim during the remaining time of the matched minute. We limit the data usage of Swiper at 1 GB per matched minute. The machine setting of this experiment is the same as the two-VM one. In Figure 6, we first show the matched and attacked minutes at every testing hour during the experiment.This evaluation essentially shows how many one-minute traces are similar to the I/O patterns in the repository.The polynomial fit of the matched minutes shows a trend

that similar patterns demonstrate time locality, which supports the findings in [30]. The requests during the night time (hour 12 to 20) are less frequent and intense and thus less similar to the stored patterns, which are from day time. Note that Swiper is looking for the similarity in I/O patterns. The request traces could be accessing different files but the disk could show similar reading patterns.

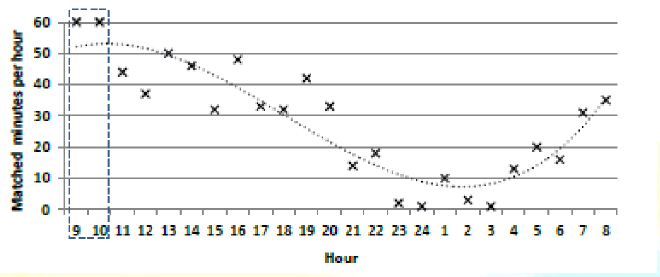

Fig. 6: Matched minutes at each testing hour in the one-day test when holding a two-hour traces in the repository. The dotted line shows a polynomial fit of the observed data points. The dotted rectangle shows the period for the training set Because the extended Swiper relaxes the matching criterion and does not hold a full trace, attacking one matched minute does not necessary mean a correct match and guarantee a significant degradation as before. Therefore, Figure 7 examines the average throughput decrease per attack at each testing hour. Although the

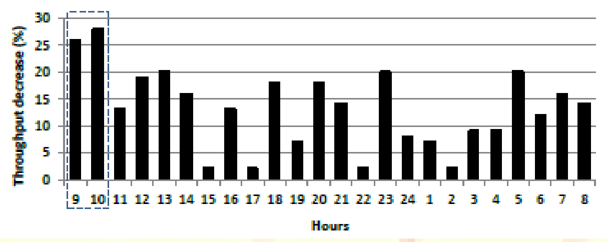

Fig. 7: The average throughput decrease per attack at each testing hour. The dotted rectangle shows the period for the training set last 22 hours are not as good as the first two, the results confirm that a historical trace could still be useful in the future. The throughput degradation ranges from 2 to 20% and has an overall average of 13.12%. As future work, using clustering methods to identify and generate patterns may greatly improve the effectiveness of Swiper.

## VII CONCLUSION

In this paper, we presented a novel I/O workload based performance attack which uses a carefully designed workload to incur significant delay on a targeted application running in a separate VM but on the same physical system. Such a performance attack poses an especially serious threat to data-intensive applications which require a large number of I/O requests. Performance degradation directly increases the cost of per workload completed in cloud-computing systems. Our

experiment results demonstrated the effectiveness of our attack on different types of victim workloads in realworld systems with various number of VMs. Interested readers may refer to Appendix I for the literature review and more discussions, where we have proposed a number of possible solutions to these types of attacks as future work. Also, it would interested to study the effects of system parameters, e.g., I/O schedulers and buffer sizes, on defending such attack.

## REFERENCES

[1] J. Szefer et al., “Eliminating the hypervisor attack surface for a more secure cloud,” in Proceedings of the 18th ACM conference on Computer and communications security. ACM, 2011, pp. 401–412.

[2] A. Gulati et al., “Parda: proportional allocation of resources for distributed storage access,” in FAST, 2009.

[3] E. Keller et al., “Nohype: virtualized cloud infrastructure without the virtualization,” in ISCA, 2010.

[4] D. Ongaro et al., “Scheduling i/o in virtual machine monitors,” in VEE, 2008.

[5] R. Shea et al., “Understanding the impact of denial of service attacks on virtual machines,” in IWQoS. IEEE Press, 2012, pp. 27:1–27:9.

[6] J. Liu et al., “High performance vmm-bypass i/o in virtual machines,” in ATEC, 2006