Understanding the Effects of Hypervisor I/O Scheduling for Virtual Machine Performance Interference

Ziye Yang, Haifeng Fang, Yingjun Wu, Chunqi Li, Bin Zhao
EMC Labs China
{ziye.yang, fang.haifeng, yingjun.wu, chunqi.li, bin.zhao}@emc.com

H. Howie Huang
The George Washington University
{howie}@gwu.edu

Abstract

In virtualized environments, the customers who purchase virtual machines (VMs) from a third-party cloud would expect that their VMs run in an isolated manner. However, the performance of a VM can be negatively affected by co-resident VMs. In this paper, we propose vExplorer, a distributed VM I/O performance measurement and analysis framework, where one can use a set of representative I/O operations to identify the I/O scheduling characteristics within a hypervisor, and potentially leverage this knowledge to carry out I/O based performance attacks to slow down the execution of the target VMs. We evaluate our prototype on both Xen and VMware platforms with four server benchmarks and show that vExplorer is practical and effective. We also conduct similar tests on Amazon's EC2 platform and successfully slow down the performance of target VMs.

1. Introduction

Cloud providers employ virtualization techniques that allow physical machines to be shared by multiple virtual machines (VMs) owned by different tenants. While resource sharing improves hardware utilization and service reliability, this may also open doors to side channel or performance interference attacks by malicious tenants. For example, CPU cache based attack has been studied in cloud environment [1, 2, 3, 4], which might be mitigated to a lesser degree when each core in new multi-core CPUs is used exclusively by a single VM (at the cost of reduced CPU utilization). On the other hand, I/O resources are mostly shared in virtualized environments, and I/O based performance attacks remains a great threat, especially for data-intensive applications [5, 6, 7]. In this paper, we discuss the possibility of such attacks, and especially focus on the effects of disk I/O scheduling in a hypervisor for VM performance interference.

The premise of virtual I/O based attacks is to deploy malicious VMs that are co-located with target VMs and aim to slow down their performance by over-utilizing the shared I/O resources. Previous work shows the feasibility of co-locating VMs on same physical machines in a public cloud [1]. In this work, we will demonstrate that a well designed measurement framework can help study virtual I/O scheduling, and such knowledge can be potentially applied to exploit the usage of the underlying I/O resources.

Extracting the I/O scheduling knowledge in a hypervisor is challenging. Generally, hypervisors can be divided into two classes, i.e., open-source hypervisor (e.g., Xen) and closed-source hypervisor (e.g., VMware ESX server). For an open-source hypervisor, while the knowledge of the I/O schedulers is public, which one is in use is unknown. To address this problem, we use a gray-box method in our framework to classify the scheduling algorithm. For a closed-source hypervisor, we use a black-box analysis to obtain the scheduling properties such as I/O throughput and latency.

With the knowledge of I/O scheduling algorithm, a malicious user can intentionally slow down co-located (co-resident) VMs by launching various attacking workloads. The main feature of such I/O performance attack is to deploy non-trivial I/O workloads and manipulate the shared I/O queues to have an unfair advantage. Note that space and time locality are the two major considerations in I/O scheduling schedulers. For example, the scheduling algorithms (e.g., Deadline, and Completely Fair Queuing or CFQ) merge the I/O requests that are continuous in logical block address (LBA) for better space locality, while other algorithms (e.g., Anticipatory Scheduling or AS [8] and CFQ too) have a time window to anticipatorily execute the incoming I/O requests that are adjacent with previous I/O requests in LBA.

In this work, we design and develop a distributed performance measurement and analysis framework, vExplorer, that allows co-resident VMs to issue a group of I/O workloads to understand I/O scheduling algorithms in a hypervisor. In particular, two types of representative workloads are proposed in this framework: the *Prober* workload is responsible for identifying the I/O scheduling characteristics of a hypervisor that include the algorithm and related properties, and the *Attacker* workload can be utilized to form I/O performance attacks, where one can dynamically configure the I/O workloads with the parameters (e.g., percentage of read/write operations) based on the extracted scheduling knowledge. To summarize, we make the following contributions in this paper:

- We design and develop vExplorer , which can be used to identify the characteristics of I/O scheduling in a hypervisor. Also, the *Prober* workloads can be adopted as an I/O profiling benchmark in virtualized environments.
- We discuss the feasibility of VM based I/O performance attacks through a simple mathematical model, and also design a set of *Attacker* workloads that are shown effective on virtualized platforms such as Xen and VMware. Furthermore, we conduct the experiments on Amazon EC2 platform [9], where several VMs are deployed on a physical host and their virtual disks (local instance store) are mapped into one local disk. For four benchmarks we

observe significant performance reduction on target VMs.

The remainder of this paper is organized as follows. Section 2 presents the design and implementation of our prototype system, vExplorer. Section 3 presents the profiling work of I/O scheduling on both Xen and VMware. Section 4 demonstrates VM I/O scheduling based attacks with the predefined mathematical model. Section 5 shows a case study of our approach on Amazon EC2, and Section 6 discusses related work. Finally, we conclude in Section 7.

2. System Design and Implementation

The challenge of exploring I/O performance attacks is to control the access patterns of the I/O workloads in various VMs for extracting the scheduling characteristics of a hypervisor. Figure 1 shows the architecture of vExplorer system that consists of distributed I/O controller (**DC**), I/O measurement daemon (**IMD**) and analytical module (**AM**). When the measurement begins, the **Monitor** in the DC interacts with the IMDs within various VMs and directs each IMD to execute the I/O tasks generated by the **Workload** module; then the outputs produced by each IMD are stored into the **Output Container** (e.g., a database); finally the DC delivers the results to the AM for knowledge extraction. This process can be repeated iteratively for training and analysis.

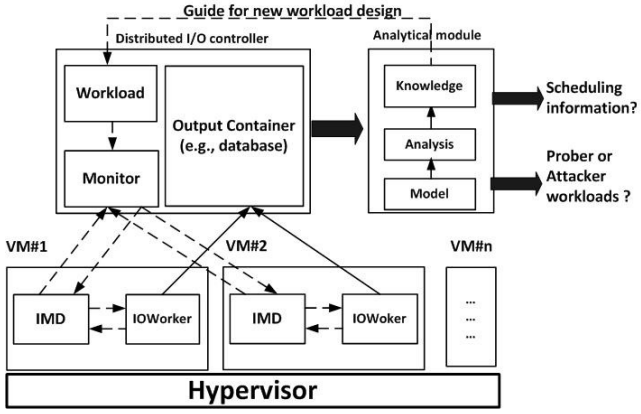

Fig. 1. vExplorer System Architecture

2.1. Distributed I/O Controller

The Monitor module is in charge of communicating with each IMD and dispatching the workloads. At the beginning, it waits for the registry requests from each IMD. Upon receiving a registry request, a service process is spawned for information exchange through the network. When the number of IMDs exceeds a threshold (e.g., 3), the Monitor starts to dispatch the I/O tasks to each IMD, where each IMD parses the configuration and executes the I/O workloads without further interaction with the monitor. Such approach is suitable for supporting more concurrent connections from IMDs.

The Workload module generates representative I/O workloads in our vExplorer system, where the regular patterns of I/O workloads are defined through a group of single I/O commands (**IOs**), in form of <sequence_id, daemon_id, launch_time, end_time, file_info, IO_mode, IO_offset, IO_size> shown in Table 1.

TABLE 1. IO Description

Sequence_id	Unique id of the IO in time sequence
Daemon_id	The execution owner (IMD) of the IO
Launch_time	The launch time of the IO, controlled by the DC
End_time	The ending time of the IO, collected by each IMD
File_info	Target file of the IO, e.g., /dev/sda1
IO_mode	Operation mode: read or write, sync or non-sync
IO_offset	The offset of the IO
IO_size	The I/O size, e.g., 4KB, 8KB and etc.

We also define several typical workload modes that will be used in our experiments.

- **Sequential mode**. Each program sequentially reads or writes a target file (i.e., a raw disk device) from the beginning to end. Furthermore, if each adjacent pair of IOs (sorted by issuing time) satisfies this formula, $IO_j(IO\_offset) = IO_i(IO\_offset) + IO_i(IO\_size)$, then such workload can be categorized as **seq-non-gap** mode, which is designed for verifying the scheduling optimization for space locality.
- **Burst mode**. Each receiver continually runs a given set of I/O tasks in a time interval. This mode can be applied to identify the maximum I/O throughput of the hypervisor.
- **Random mode**. Among a fixed number of I/O commands, the program randomly reads/writes a target file in a ratio (ranged from 0% to 100%), and the remaining IOs are sequential I/O commands. The usage of *random* mode is to measure VM I/O latency on different I/O sizes.

2.2. I/O Measurement Daemon

IMD, a daemon running within a VM, is responsible for interacting with the Monitor and executes dispatched I/O commands. Once an IMD is adopted as a working node by the Monitor, it spawns several IOworkers according to the requirements from the monitor. For executing IOs at a specified *launch_time*, two approaches are provided:

- **Time synchronization**. Each VM holding the IMD must synchronize the time with the DC host through NTP (Network Time Protocol) during the working node registration.
- **Timer event control**. We choose the *timer* policy proposed in Linux 2.6 due to its flexibility and accuracy, which will not be affected by the side effects of process scheduling.

2.3. Analytical Module

This module applies statistical analysis on experimental results in order to determine the representative I/O workloads and extract the I/O scheduling knowledge. Generally, I/O performance attacks can be carried out through the following three stages.

- Stage I: Identify the *Prober* workloads. Each workload selected by the *prober* must distinguish at least two types of I/O schedulers, i.e., such workload can differentiate either the space or time locality of two different schedulers.
- Stage II: Extract the scheduling characteristics of the target hypervisor by utilizing the *prober* workloads.
- Stage III: Identify the *Attacker* workloads. Here the selected workloads shall leverage the discovered scheduling knowledge to observe the I/O behaviors of target VMs and try to reduce their performance by over-utilizing the I/O resources.

In the following two sections, we will discuss these three stages in details.

3. Identifying Hypervisor I/O Scheduling

In this section, we focus on the first two stages, Stage I and II, that aim to determine the *Prober* workloads and identify the characteristics of hypervisor disk I/O scheduler. To evaluate our prototype vExplorer, several experiments are conducted on Xen and VMware platforms. As Xen is an open-source hypervisor, the major task is to classify its I/O scheduling algorithm. On the other hand, our main focus on closed-source VMware platform is to profile the scheduling properties. Compared with non-virtualized environment, I/O operations within a VM are influenced by I/O schedulers in two tiers, i.e., the guest VM kernel and hypervisor. To precisely extract knowledge of I/O scheduler in a hypervisor, the influence from the guest VM kernel must be reduced to a minimum. Thus we use basic FIFO-like I/O scheduling (i.e., Noop) is selected in guest VMs, and bypass the file buffer cache through direct I/O.

3.1. Classifying Scheduling Algorithm in Xen

For Xen, we design a set of workloads with *seq-non-gap* mode (defined in Section 2), named as ***Prober-1***, which is comparatively suitable for classifying the scheduling algorithms. Currently, our *Prober-1* is designed to read a raw hard disk from low LBA (logical block address) to high LBA. Table 2 lists the key terms for analyzing the effects after executing the *Prober-1* on a virtualized platform.

TABLE 2. Terminologies

IMDN	Number of IMDs (VMs) in an experiment
TN	Total number of IOs in an experiment
Switch	I/O request serving from one VM to another
Service period	I/O service for one VM between two switches
TSP	Total number of service periods
Cyclic Switch (CS)	Regular switches patterns which involve all IMDs
CSN	Total number of CS appeared in an experiment
CSF	CSF=CSN*IMDN/TN
OPN	Number of IOs in a service period
AOPN	Average OPN in an experiment, AOP=TN/TSP
SDOP	Standard deviation of all OPNs in an experiment
RT(IO_SIZE)	response time of an IO on IO_SIZE
ART(IO_SIZE)	Average response time of all IOs on an IOSIZE
incART	incART = ART(2*IO_SIZE)/ART(IO_SIZE)
SNR	Signal-to-Noise Ratio, SNR=AOP/SDOP

The concept of *switch* is introduced to measure the frequency when the hypervisor stops serving I/O requests issued by one VM and starts to serve I/O requests from another VM. In the analytical phase, all executed IOs are sorted by the *end_time* in an ascending order. If the neighboring IOs are issued by different VMs, it is considered as a switch. A *service period* for a VM can be defined as the I/O service time between two switches. The *Cyclic Switch* (CS) describes some regular switch patterns which involves all IMDs. For example, if there are three IMDs (with id of 0, 1, 2, respectively) and each issues three commands, the final sequence of the IOs can be represented by the IMD ids, e.g., 1,0,1,2,1,2,0,2,0. Then a tuple of (0,1,2) is a CS, a tuple of (1,2,0) is another CS, and in this case CSN that standards for the total number of cyclic switches is 2. The CSF is 2/3, which reflects the fairness of hypervisor I/O scheduler. Further, *RT(IO_SIZE)* describes the I/O response time of a single IO on IO_SIZE, and ART(IO_SIZE) represents the average I/O response time of many IOs on IO_SIZE. Last, incART defines the variation of I/O size on the impacts of ART.

In the following experiments, we deploy the DC on a DC-Host machine and three IMDs in different domUs (guest VMs) on a Xen-Host machine, which has Intel Pentium 4 3.2GHz, 4GB RAM, and 250GB 7200RPM SATA disk. We use Xen version 3.0.1 in the experiments. All three domUs have the same configuration and each owns a raw disk with 10G size with "file-typed" mount. In each experiment, every domU is required to concurrently execute the *Prober-1* workload on this disk. Moreover, we repeat the same experiments under four different I/O schedulers (i.e., Noop, Deadline, AS, CFQ) configured in Dom0. To reduce the errors, the *Prober-1* with different configurations of *IO_size* (from 4KB to 1MB) is executed for at least ten iterations.

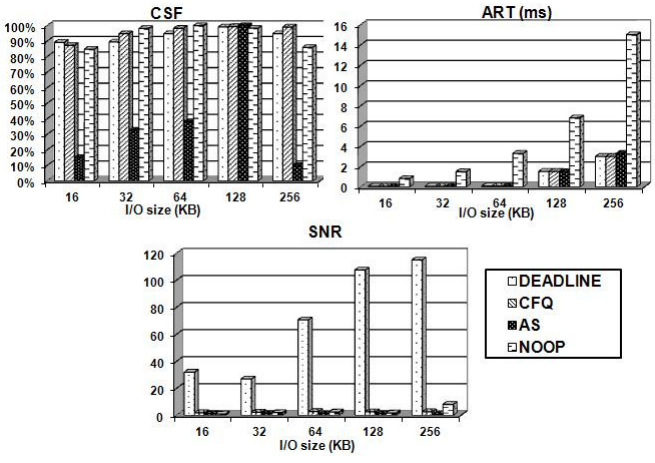

Fig. 2. Values of CSF, SNR, ART on four schedulers

Figure 2 shows the statistical value of several items (i.e. CSF, SNR, ART) from the experiments. The value of CSF on Deadline, CFQ and Noop lies from 80% to 100% with different I/O sizes, while AS shows a relatively low value except for the I/O size of 128KB. This phenomenon shows that Deadline, CFQ and Noop can provide equal probability to different VMs

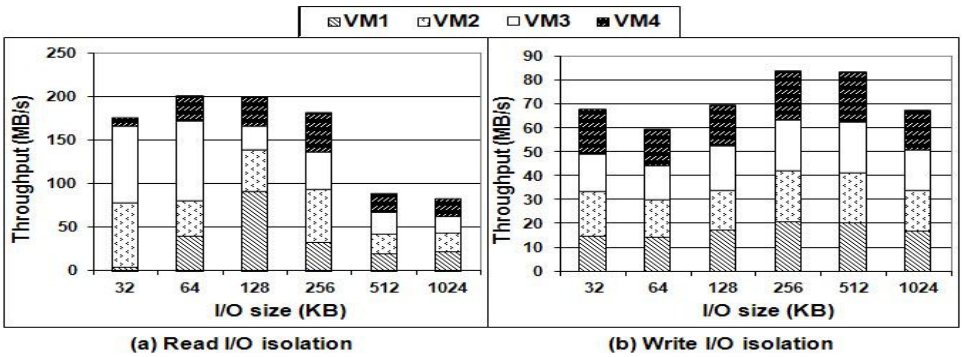

Fig. 3. I/O throughput isolation among equal VMs

for the *Prober-1* workload. Meanwhile, Deadline has a large SNR value which indicates that it provides a stable I/O service to *Prober-1* workload since it has a fixed size of batched FIFO queue to serve continuous read I/O requests. The ART diagram describes how the IO_SIZE variation changes the value of ART, and only the value of incART on Noop is close to 2 while other scheduling algorithms have no such effects. The reason is that when there are equal I/O requests issued from different VMs in a period, Noop alternatively serves the requests in FIFO manner, thus the value of ART nearly doubles when the IO_size doubles, which can help distinguish the Noop scheduler.

I/O scheduling algorithm classifier: The features extracted in previous experiments can be used to classify the open-source I/O scheduling algorithms which can be summarized as three decision rules listed in Table 3. *Rule1* examines the CSF value to verify whether the scheduling method provides VMs with equal service, then As-like scheduler can be predicted if CSF is no larger than α. As *Rule2* suggests, if SNR values are consistently larger than a certain threshold β, then we can predict that the hypervisor deploys a Deadline-like scheduler. *Rule3* can check whether Noop-like scheduler is selected. If none of the three rules is satisfied, then CFQ can be suggested as the possible scheduling algorithm if there are only four schedulers. In the case when more than one rule is triggered, the scheduling algorithm is undetermined. It might indicate that there exists new scheduling algorithms, and the classification rules should be retrained and updated.

TABLE 3. Classification Rules

Rule1	IF $CSF < \alpha$, As-like scheduler.
Rule2	IF $SNR > \beta$, Deadline-like scheduler.
Rule3	IF $incART \in [2-\epsilon, 2+\epsilon]$, Noop-like scheduler.

Currently, the summarized three rules are quite useful to classify the I/O scheduling on Xen platform with four common I/O schedulers. In practice, We set β to 5.0 according to Rose criterion [10], α to 80.0% and ϵ to 0.20 empirically. When we make use of our vExplorer to measure other Xen platforms with different versions or configurations, it successfully determines the I/O scheduler.

3.2. Profile I/O Scheduling in VMware

The proposed I/O scheduling algorithm classifier is not suitable for the hypervisors with closed-source I/O schedulers. Here we aim to understand a number of scheduling properties (e.g., VM I/O throughput, I/O execution latency).The test machine has Intel Xeon X5355 2.66GHz, 16GB RAM, and 300GB SCSI disk, and we use VMware ESX 4.0 in the tests.

Profiling on I/O throughput: VM I/O throughput characteristics can be profiled through two aspects: (1) throughput variation of single VM with co-resident VMs; (2) throughput isolation among VMs. Here we utilize a new workload, named ***Prober-2***, which combines both the seq-non-gap and Burst modes (defined in section 2) to continually read or write the target files in a fixed interval (e.g., 10 seconds), with each IO size ranging from 4KB to 1024KB. Figure 3 presents the performance isolation results among four equal VMs. Clearly, the performance isolation on ESX for equal VMs is relatively poor for reads, but nearly perfect for writes.

Profiling on I/O execution latency: We design another workload (***Prober-3***), which utilizes the Random mode (defined in section 2) to continually read or write operations with random file offsets. Generally, I/O response time (RT) of each IO is calculated by this formula, i.e, RT(IO)=End_time(IO)-Launch_time(IO). For a hypervisor scheduler, RT time of each IO is composed of the wait time in I/O queues (wait_time) and the real I/O serving time (serve_time), i.e., RT(IO)= wait_time(IO) + serve_time(IO). Since all IOs are continually executed in the experiments, the serve_time of each IO_i can be expressed by serve_time(IO_i)= End_time(IO_i) - End_time(IO_j), IO_j is the latest completed IO before IO_i.

Figure 4 and 5 present the I/O RT and serve_time variation of IOs on 128 KB with Random mode issued by a single VM (target VM) in either single or multi VM environments. In each diagram, X-axis indicates the proportion of sequential I/O operations, and different lines represent the number of peer co-resident VMs in the platform. In Figure 4, we can see that the sequential proportion greatly affects the RT value of read I/O operations, i.e., the higher sequential proportion, the lower value of RT; However, the sequential proportion has no influence on the RT value of write operations. Moreover, the

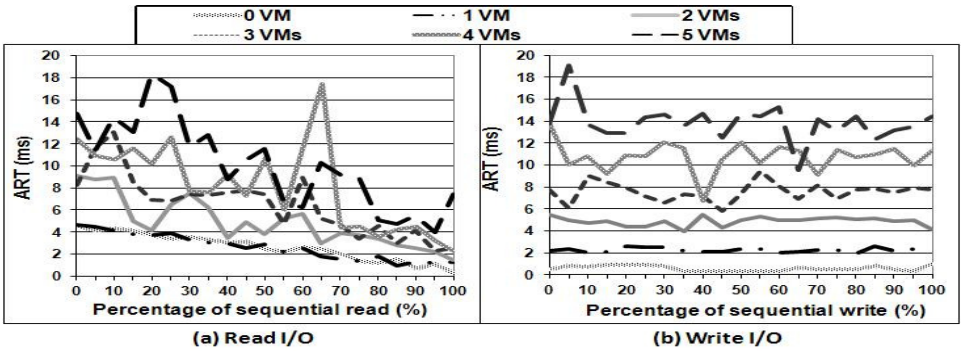

Fig. 4. I/O RT variation(IOSIZE=128KB) of a single VM

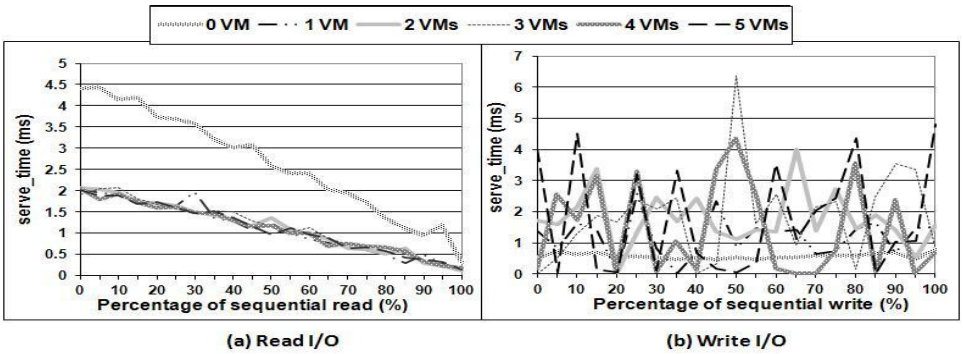

Fig. 5. I/O serve_time variation(IOSIZE=128KB) of a single VM

RT value for both read and write operations increase when the number of VMs increase. Except the case when there are two equal read VMs, the variation of RT has no change.

Figure 5(a) presents the serve_time variation of IOs on read operations. When there is only one target VM, the value of serve_time is really high that indicates that ESX-4.0 only grants parts of I/O capability to the target VM. When there are more co-resident VMs, ESX-4.0 has to utilize more capabilities to serve concurrent IOs from different VMs, so the serve_time is reduced instead. Moreover, the serve_time is nearly stable if we continue increasing the number of VMs, which indicates the actual capabilities of the underlying hard drivers. Figure 5(b) presents the serve_time variation of IOs on write operations. When there is only one target VM, the serve_time is nearly stable. But when the number of co-resident VMs increases, the serve_time is unstable and oscillates, which indicates there are frequent I/O service switch events among the VMs.

In all, we summarize the scheduling properties on VMware ESX as follows:

Reads. ESX aims to ensure the serve_time of each read operation, however it does not seem to guarantee the throughput isolation across the VMs. In most cases, the applications in the VM are expected to immediately consume the results of read operations, so ensuring the serve_time with low VM I/O service switch is reasonable. Also the RT value of sequential read is lower than random read which suggests that ESX maintains caching and prefetching to optimize the read operations.

Writes. ESX aims to guarantee the throughput isolation among the VMs instead of the serve_time of each single IO. As most write operations are asynchronous I/Os, it seems that ESX chooses to delay the immediate execution and periodically flush IOCs to the disk. As a result, the serve_time of each IO varies irregularly. Moreover, the RT value of write operations seems stable even if sequential write proportion changes, which incidates that ESX might have no optimization for sequential write I/O patterns.

4. I/O Scheduling Based Performance Attacks

The vExplorer system can be utilized as an attacker that can potentially hurt the performance of co-resident VMs. To formalize the attack approach, a mathematical model is presented, and a number of experiments are conducted on both VMware and Xen.

4.1. Mathematical Model

Suppose that n VMs are deployed on a virtualized platform, each VM's I/O behavior can be expressed by four features shown in Table 4, denoted as a vector $\vec{X} = \{X_{iosize}, X_{pread}, X_{pseq}, X_{pburst}\}$. Generally, the throughput variation of a VM$_i$ (denoted as THR$_i$) in a host is influenced by its own behavior $\vec{X_i}$ and the effects from hypervisor I/O scheduler, which can be denoted as a function S. Thus we have $[THR_1, THR_2, ..., THR_n] = S(\vec{X_1}, \vec{X_2}, ..., \vec{X_n})$. For a VM$_i$,

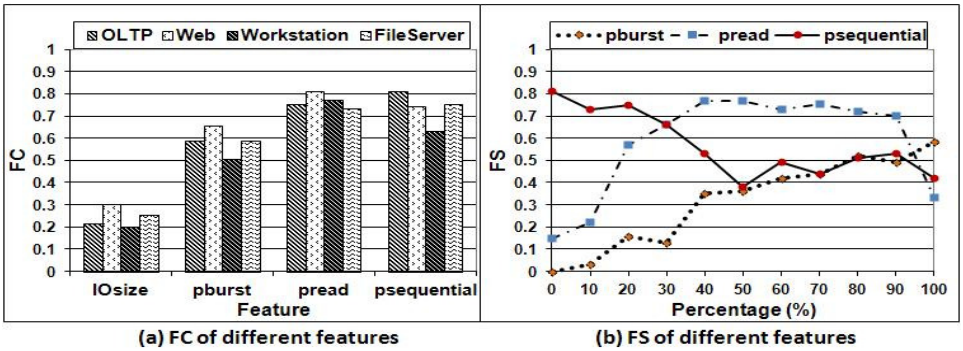

Fig. 6. FC and FS on ESX-4.0

if there is no other VMs on the host, $THR_i = S(\vec{X_i})$. In such situation, THR_i can be stable if $\vec{X_i}$ is fixed in a time period, which can be denoted as $\text{ST}(\text{THR}_i)$. If the VM performance isolation is not guaranteed by S, malicious VMs can adjust their I/O behaviors ($\sum \vec{X}$) to attack the target VMs.

TABLE 4. VM I/O behavior description

X_{iosize}	IO size of each read/write operation
X_{pread}	Percentage of read operations (%)
X_{pseq}	Percentage of sequential read/write operations (%)
X_{pburst}	Utilization rate of VM's maximal IOPS (%)
Workloads	$ATT(\vec{X})$

We propose to observe a single feature while the other features are fixed at each time. We study the following two concepts **Feature Contribution (FC)** and **Feature Sensitivity (FS)**:

FC depicts the influence of a single feature (named as X_f) in $\vec{X_{att}}$ on THR_{tar}. Suppose $\vec{X_{tar}}$ is fixed in a time period, then with different assignments of X_f, we can obtain the maximal and minimal value of THR_{tar}, named as $\text{Max}(\text{THR}_{tar})$ and $\text{Min}(\text{THR}_{tar})$. Thus the contribution of X_f on FC can be defined as: $\text{FC}(\text{X}_f) = (\text{Max}(\text{THR}_{tar}) - \text{Min}(\text{THR}_{tar}))/\text{ST}(\text{THR}_{tar})$. The higher value of FC, the larger impacts of X_f.

FS describes the influence of X_f in $\vec{X_{att}}$ on detecting the variation of THR_{tar}. Generally, attacking VMs detect the variation of THR_{tar} by observing their own throughput variation, THR_{att}. Suppose the variation patterns of THR_{tar} is fixed in a time period, we could obtain the $\text{Max}(\text{THR}_{att})$ and $\text{Min}(\text{THR}_{att})$ when there is an assignment on X_f (i.e., X_f=a). Thus the contribution of X_f can de defined as: $\text{FS}((\text{X}_f\text{=a})) = (\text{Max}(\text{THR}_{att}) - \text{Min}(\text{THR}_{att}))/\text{ST}(\text{THR}_{att})$. Obviously, a higher value of FS indicates more meaningful observation of THR_{tar}, and the assignment that leads to the highest FS can be considered as an optimal assignment.

We conduct the experiments on VMware ESX with two VMs deployed on a host, i.e., VM_{att} and VM_{tar}. We use four different benchmarks (FileServer, OLTP, WebServer, Workstation) described in [11]. As shown in Figure 6(a), the I/O behavior of VM_{tar} is fixed and VM_{att} adjusts the features on four

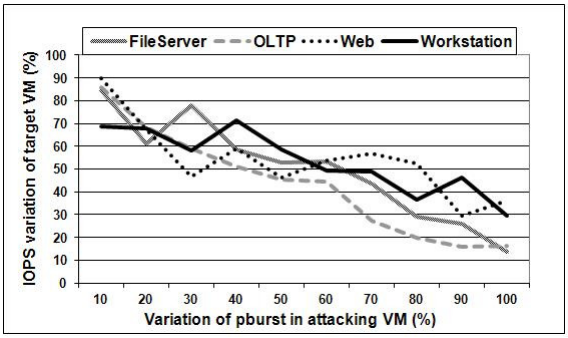

Fig. 7. Impact of pburst on VMware platform

benchmarks. This indicates that X_{iosize} has little contribution on FC, while the remaining features are the key FC if the threshold is 0.5. Figure 6(b) shows that the value change of VM_{att}'s X_{pread}, X_{pseq} and X_{pburst} on detecting the behavior of target VM. When X_{pread} ranges from 40 to 50, X_{pseq}=0, and X_{pburst}=100, the observation seems quite effective. Although some features with little contribution to either FC or FS (e.g., X_{iosize}) can be set with random values, it is suggested to be assigned with some fixed value for lower cost.

4.2. Attacking Target VMs

While the I/O behavior of target VM_{tar} is identified, attacking workloads, named as *ATT*, can be designed for I/O resource occupation based on FC and the scheduling characteristics of hypervisor (function S). In the following, some attacking experiments are done in both VMware and Xen platform to demonstrate the feasibility of our mathematical model.

VMware ESX: We would like to verify the influence of X_{pburst}, as it is one of the non-negligible FCs for ESX-4.0. Thus the malicious VM can use ATT(4KB, 0, 100, X_{pburst}) as the attacking workload *ATT* and aim to reduce the throughput of VM_{tar}. Here, X_{iosize} is assigned with 4KB for lower attacking cost, and X_{read} and X_{pseq} are assigned with 0 and 100.

Figure 7 demonstrates the experiments on ESX. There were two VMs, the attacking VM (named as VM_{att}) performing the *ATT* and the VM_{tar} running four different benchmarks

(FileServer, OLTP, WebServer, Workstation). In the diagram, X-axis describes the X_{pburst} variation of VM_{att} and Y-axis represents the IOPS variation of VM_{tar} in percentage. When X_{pburst} increases, VM_{tar}'s I/O throughput demonstrates a decreasing trend, which indicates that value change of X_{pburst} can influence the I/O behaviors of VM_{tar} in a fine-grained way.

Xen: As we can classify the I/O scheduler on Xen platform, customized *ATT* can be designed for different I/O scheduling algorithms (Noop, Deadline, CFQ, AS) based on their unique characteristics. To understand the effects of four different parameters, i.e., X_{pburst}, X_{iosize}, X_{read}, and X_{pseq}, we conduct several experiments on Xen and conclude the following rules:

- X_{iosize} has nearly no effects on resource utilization on four I/O schedulers.
- X_{pread} only has significant effects on AS. The appropriate selection of X_{pread} is important for throughput influence on target VM, e.g., X_{pseq} ranging from 20% to 40% is recommended.
- X_{pseq} plays little contribution on throughput influence for CFQ and AS. However for Noop and Deadline, X_{pseq} can be set to 100 - ξ ($\xi > 0$) , which may have large impacts on resource consumption.
- X_{pburst} is effective on all schedulers. As the value of X_{pburst} increases, the performance interference can be enhanced.For the AS scheduler, the maximal value X_{pburst} on read I/O attacks can be set with 100 - ϵ ($\epsilon > 0$), since AS has an anticipatory execution time window (e.g., 7ms) for the next read operations issued by the same process.

5. Case Study: Experiments on a Public Cloud

To verify the practicality of our approach, we deploy vExplorer on Amazon EC2 platform in Singapore. As perviously described, the successful VM I/O based performance attacks are relied on two conditions: (1) VM co-residence and (2) I/O scheduling knowledge of the underlying hypervisor.

5.1. Hypervisor I/O Scheduler Identification

With the technique described in [1], we deploy four VMs using Amazon m1.small instances with 1ECPU and 1.7GB RAM. The goal is to differentiate the I/O scheduling algorithms on Amazon's Xen platform and three of them are co-resident in the same host. In each IMD VM, Prober-1 workload is operated on its virtual disk "/dev/sdb". However, such experiment on Amazon is slightly different with the one on local Xen platform in two aspects: (1) There may exist co-resident VMs owned by other tenants which may influence the profiling work for hypervisor I/O scheduler; and (2) The virtual disks operated by the three co-resident VMs in the same host may not be mapped into the same physical disk.

The first issue is not difficult to solve. We can detect the existence of other storage co-resident VMs before each experiment through the workloads guided by FS (defined in 4.1) and select a time period that when other VMs are relatively idle. However, the second issue is very challenging. If the virtual disk operated by the three VMs are located into different disks which means non-sharing of I/O resource, then Prober-1 workloads will have limited impacts. In the tests, we find that the virtual disk (/dev/sdb) of two VMs are most likely mapped into a physical disk.Thus only two VMs (VM_A and VM_B) are available to perform the probing work of hypervisor disk I/O scheduler.

Table 5 shows the testing results of prober-1 on VM_A and VM_B. Obviously, the value of CSF is consistently equal to 100% when there are two VMs, so Rule1 (described in Section 3.1) can never be used. All values of SNR is less than 5, so Deadline-like scheduler can be excluded. All values of incART are ranged from 1.5 to 2, so it is not Noop-like scheduler according to Rule3 if ϵ is set to 0.2. But when the ϵ is configured to 0.5, it is Noop-like scheduler. If the I/O scheduler of Amazon Xen-like hypervisor (i.e., the I/O scheduler in domain0) only supports four open source algorithms (e.g., Noop, Deadline, AS, CFQ), the possible schedulers can only be AS, CFQ and Noop.

TABLE 5. Statistical results on Amazon EC2

I/O size (KB)	CSF	SNR	ART (ms)	incART
16	100%	0.35	0.22	N/A
32	100%	0.59	0.33	1.5
64	100%	0.27	0.5	1.52
128	100%	0.63	1.0	2
256	100%	1.04	2.0	2

Derived from Section 4.2, when the pause time ranged from 0 to X ms (e.g., X can be set with 7ms), the attacking effects on AS scheduler can be the same. Thus we conduct the following attacking experiment through two VMs, i.e., the VM_{tar} is configured with $\vec{X}$={4KB, 100, 0, 100} and the VM_{att} runs ATT(4KB, 100, 100, X_{pburst}) on Amazon. Figure 8 shows the results of X_{pause} experiment on Amazon, the X-axis describes the variation of pause time of VM_{att}, and the Y-axis shows the IOPS variation of VM_{tar}. With the pause time changed from 0 to 25 ms, there is no phenomenon revealing that the attacking effects are same in a time window. With such experiment, we can confirm that Amazon's Xen Hypervisor is not configured with AS scheduling algorithm. With the previous two experiments, it can be deduced that Amazon adopts either Noop or CFQ scheduler, and it is likely that Amazon uses CFQ instead of Noop.

5.2. VM-based I/O Performance Attacks on EC2

As we estimate that the I/O scheduler of Amazon's Xen is either CFQ or Noop, we conduct the experiments on Amazon with the variation of X_{pburst}. In our experiments, there are still two VMs (VM_{att} and VM_{tar}), i.e., VM_{att} is equipped with ATT(4KB, 10, 95, X_{pburst}) (according to Section 4.2) and VM_{tar} runs four different benchmarks (FileServer, OLTP, WebServer, Workstation). In Figure 9, we could see that VM_{tar}'s I/O throughput demonstrates a decreasing trend with the increasing of X_{pburst} in VM_{att}. It demonstrates that our

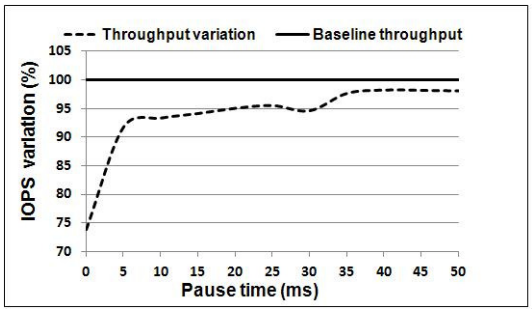

Fig. 8. Impact of pause on Amazon EC2

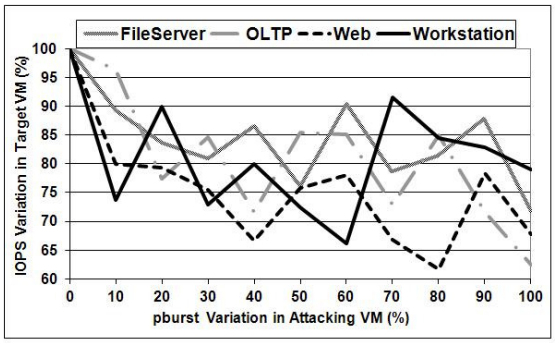

Fig. 9. I/O Performance attacks on Amazon EC2

attacking approach remains effective on public cloud. Also if the target VM is provided with relatively high IOPS, we may need to launch more attacking VMs instead of only one VM for performance influence.

6. Related Work

VM I/O performance. Most prior works [12, 13, 14, 15, 16, 17] have focused on I/O scheduling optimization of the hypervisor to provide better virtual I/O service. [18] aims to ensure performance fairness among different VMs. The goal of vExplorer is to extract the characteristics and measure the quality of I/O scheduling subsystem by designing meaningful I/O patterns.

VM Security. [1] used network probing in EC2 and was able to extract the sensitive information from the co-resident VMs by issuing CPU cache based side channel attacks. To mitigate such issues, Zhang et al. [2] proposed to reversely use the CPU cache as a guard, where the tenants can observe CPU cache usage and detect such attacks. Here we propose a new system, vExplorer, to detect and influence the behaviors of co-resident VMs by analyzing disk I/O patterns.

7. Conclusion

This paper presents vExplorer, a distributed I/O performance measurement system, which can help identify the characteristics of disk I/O scheduler in a hypervisor and conduct I/O based performance attacks. We conduct a number of experiments on both Xen and VMware platforms. In addition, we deploy vExplorer on Amazon EC2 and successfully slow down the performance of co-resident VMs. We plan to further study VM vulnerability for I/O based attacks, and study the preventive methods in future work.

Acknowledgment

The authors thank anonymous reviewers for their helpful suggestions. This work is in part supported by National Science Foundation grant OCI-0937875.

References

[1] 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*, 2009, pp. 199–212.

[2] Y. Zhang, A. Jules, A. Oprea, and M. K. Reiter, "Homealone: Co-residency detection in the cloud via side-channel analysis," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy (Oakland)*, 2011, pp. 313–328.

[3] A. Aviram, S. Hu, B. Ford, and R. Gummadi, "Determinating timing channels in compute clouds," in *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, 2010.

[4] T. Kim, M. Peinado, and G. Mainar-Ruiz, "Stealthmem: system-level protection against cache-based side channel attacks in the cloud," in *Proceedings of the 21st USENIX conference on Security symposium*, 2012.

[5] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor," in *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, 2001, pp. 1–14.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003.

[7] R. C. Chiang and H. H. Huang, "TRACON: interference-aware scheduling for data-intensive applications in virtualized environments," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011.

[8] S. Iyer and P. Druschel, "Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous i/o," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[9] Amazon, "Amazon elastic compute cloud," http://aws.amazon.com/ec2/.

[10] T. B. Jerrold, E. M. L. J. J. Anthony, Seibert, and M. B. John, *The Essential Physics of Medical Imaging (2nd Edition)*. Philadelphia: Lippincott Williams & Wilkins, 2006.

[11] R. Bryant, D. Raddatz, and R. Sunshine, "Penguinometer: a new file-i/o benchmark for linux," in *Proceedings of the 5th annual Linux Showcase & Conference - Volume 5*, 2001, pp. 10–10.

[12] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling i/o in virtual machine monitors," in *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, 2008.

[13] A. Gulati, A. Merchant, and P. J. Varman, "mclock: handling throughput variability for hypervisor io scheduling," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010.

[14] A. Gulati, C. Kumar, and I. Ahmad, "Modeling workloads and devices for io load balancing in virtualized environments," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, January 2010.

[15] M. Kesavan, A. Gavrilovska, and K. Schwan, "On disk i/o scheduling in virtual machines," in *Proceedings of the 2nd conference on I/O virtualization*, 2010.

[16] D. Boutcher and A. Chandra, "Does virtualization make disk scheduling passé?" *SIGOPS Oper. Syst. Rev.*, vol. 44, March 2010.

[17] S. R. Seelam and P. J. Teller, "Virtual i/o scheduler: a scheduler of schedulers for performance virtualization," in *Proceedings of the 3rd international conference on Virtual execution environments*, 2007.

[18] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing performance isolation across virtual machines in xen," in *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, 2006.