A generic mitigation framework against cross-VM covert channels

Wen Qi, Jin Wang, Hermine Hovhannisyan, Kejie Luyz, Jianping Wang, Junda Zhu

Post-print deposited in the CityU Institutional Repository, City University of Hong Kong.

Citation:
Qi, W., Wang, J., Hovhannisyan, H., Lu, K., Wang, J., & Zhu, J. (2016). A generic mitigation framework against cross-VM covert channels. *2016 25th International Conference on Computer Communication and Networks (ICCCN)*,1-10.
doi: 10.1109/ICCCN.2016.7568479

© 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

A Generic Mitigation Framework against Cross-VM Covert Channels

Wen Qi*, Jin Wang§, Hermine Hovhannisyan*, Kejie Lu†‡, Jianping Wang*, Junda Zhu◊
* Department of Computer Science, City University of Hong Kong
§ School of Computer Science and Technology, Soochow University
† School of Computer Engineering, Shanghai University of Electric Power
‡ Department of Electrical and Computer Engineering, University of Puerto Rico at Mayagüez
◊ Faculty of Science and Technology, University of Macau

***Abstract*—In recent years, many cross-VM covert channels have been discovered in cloud computing, causing serious security concerns. For such covert channels, some mitigation schemes have been proposed, but usually one mitigation scheme aims at a specific covert channel, which may be inefficient in defending against potential new attacks. In this paper, we propose a generic solution to mitigate the risk of a broad class of timing-based cross-VM covert channels. The design is motivated by our finding that the capacity of most timing-based cross-VM covert channels highly depends on the co-run probability among VMs, where the co-run probability depends not only on how VMs are assigned to servers, but also how VMs are scheduled on a single server, which is related to managing the vCPUs assigned to each VM. We find that the VM co-run probability can be reduced when the number of vCPUs increases, but it also causes extra system overhead in resource utilization. In this paper, we propose a generic VM provisioning and VM scheduling solution to jointly minimize the co-run probability among VMs, meanwhile, maintaining high resource utilization. We experimentally demonstrate that the proposed scheduling algorithm can mitigate the risk of timing-based cross-VM covert channel with lower system overhead. We also conduct simulation of VM provisioning which shows that the proposed solution can achieve the balance between high resource utilization and low risk of information leakage caused by cross-VM covert channels.**

***Index Terms*—Cross-VM covert channel; Mitigation**

I. Introduction

With the development of the cloud computing industry, more and more businesses are now being moved to the cloud by utilizing various cloud services. As reported by Forrester Research [1], the public cloud market was 58 billion USD in 2013 and is expected to reach 191 billion USD by 2020. Among various cloud services, one of the most important types is *Infrastructure as a Service* (IaaS). To facilitate IaaS efficiently, most cloud service providers use the technology of *virtualization*, which allows a single server to simultaneously host multiple *virtual machines* (VMs) of different tenants. Virtualization can significantly improve the resource utilization because co-resident VMs on a server can share the same hardware resources.

However, virtualization may be exploited by the malicious user to establish the communication between a victim VM and an attacking VM even if the cloud security policies prohibit the communication between the two VMs. In the literature, such a communication channel is known as *cross-VM covert channel*.

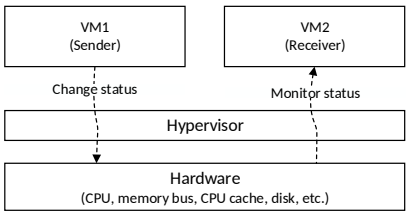


Fig. 1. A generic model for cross-VM covert channel.

Cross-VM covert channel has attracted significant attention recently in the cloud security community [2]–[7] due to the high security threat introduced. Confidential or private data can be leaked between two VMs even though the security policy does not allow them to communicate and once a cross-VM covert channel is established, it is very hard to be detected because the operations of VMs seem to be legitimate.

In general, cross-VM covert channels can be established by exploiting shared hardware components, such as CPU [2], memory [3], [4], cache [3], [5], [7], disk [3], *etc*. As shown in Fig. 1, a covert channel is created between a sender and a receiver. To send information, the sender will deliberately modify the status of a shared hardware component, while the receiver can obtain data by monitoring the status of the same component. For example, in a memory bus covert channel, the sender can lock the memory bus to create contention patterns if it wants to send a bit "1" while the receiver can infer "1" based on the monitored contention pattern at the same time. Similar to this example, most existing cross-VM covert channels require the sender and receiver to access the hardware component according to a predefined timing pattern. Therefore, we focus on *timing-based* covert channels.

To enhance the security of the cloud, it is critical but very challenging to mitigate cross-VM covert channels in the cloud environment. Recently, some mitigation schemes have been proposed [8]–[15], which can be generally classified into following types: (1) *Securing VMs' isolation from the hardware layer* [8], [9]. This method requires hardware modification, which is not easy to implement and cannot be applied to all existing hardware. Even if this approach is implemented with customized hardware, there will be huge cost introduced to the cloud provider. (2) *Offering dedicated servers* [10]. This approach can eliminate covert channel

threats completely. However, without the benefit of resource multiplexing, much higher cost is introduced to the tenants. (3) *Limiting VMs' access to high precision timing functions* [11]. Timing based channels require fine-grained timing. Therefore, putting restriction or adding noise to the access of high privileged timing functions can increase the error rate of these covert channels. But one downside of this approach is that some legitimate applications, which require fine-grained timing, may be affected or even stop working. (4) *Applying access policy to hardware or the hypervisor* [12], [13]. This technique requires modification of hypervisor and introduces performance overhead. (5) *Migrating VMs [15]*. This solution aims to reduce the probability of VMs co-residency in order to break the prerequisite for constructing cross-VM covert channels. In practice, the cloud provider cannot migrate VMs with a high frequency, otherwise the performance degradation will be intolerable, which means this solution cannot mitigate cross-VM covert channel completly.

To summarize, many existing mitigation methods require high implementation cost and lead to performance degradation. Moreover, some mitigation schemes have limited efficiency because they were designed to target a specific cross-VM covert channel. For example, [13] is only applied to CPU cache based covert channels. Considering the number of various covert channels reported and the number of unknown covert channels, it will not be efficient or effective to launch a mitigation solution for each specific channel.

Certainly, it is highly desirable to develop a generic mitigation solution against most (if not all) types of timing-based cross-VM covert channels. In this study, we will tackle this challenging issue. Specifically, we first conduct extensive experiments, where we consider practical scenarios that a VM may need multiple *virtual CPUs* (vCPUs). Our experimental results show that the capacity of various timing-based cross-VM channels depends on the probability that the sender and receiver are scheduled to run simultaneously, termed as the *co-run probability*. The results show that, with the increase of the number of vCPUs, both VM co-run probability and covert channel capacity decrease monotonically, but the system overhead increases.

Based on the above observation, we propose a generic VM provisioning and scheduling framework to facilitate dynamic VM creation in real-time, where each VM creation request contains not only the resource requirement but also its security requirement for the risk of covert channel. Our framework aims to minimize the overall system cost for VM creation, and can guarantee the security requirement of both the newly arrived VM and existing VMs. Within the framework, we develop an efficient VM scheduling scheme which minimizes the maximum co-run probability between any two running VMs on a server and an efficient provisioning algorithm which selects a server to accommodate a newly arrived VM request with the minimum incurred cost to the cloud service provider.

The major contributions of this paper are summarized as follows:

- We conduct extensive experiments in XEN to demonstrate the relationship between covert channel capacity and the co-run probability between co-resident VMs under various covert channels.
- Based on our observation from the experiments, we propose a generic cross-VM covert channel mitigation framework under various covert channels, which aims to minimize the cost while ensuring user specified security level through joint VM scheduling and VM provisioning.
- Within the framework, we design an efficient *Equal Scheduling* scheme. Theoretical analysis shows that *Equal Scheduling* can minimize the maximum co-run probability between any two co-resident VMs in the worst case, without prior knowledge of how many or what kinds of VMs are co-resident. The experiments on XEN hypervisor have shown that the system overhead under our proposed *Equal Scheduling* scheme is much lower than that under another generic mitigation solution in the literature for the same mitigation performance.
- We also propose a VM provisioning algorithm to place VMs on servers so as to minimize resource consumption as well as to fulfill tenants' security level requirements. According to the simulation results based on Google Cluster traces, the proposed VM provisioning algorithm launches much less number of active servers than the conventional VM provisioning algorithms when provisioning VMs with security requirements.

The remainder of this paper is organized as follows. In Section II, we discuss our preliminary efforts on analyzing the problem. In Section III, we introduce the design of the proposed mitigation framework. In Section IV, we present a theoretical analysis of scheduling strategy inside a single server. In Section V, we discuss the VM provisioning among servers, while we introduce the problem formulation and present a VM provisioning algorithm. The framework evaluation is given and analyzed in Section VI. In Section VII, we discuss related work on cross-VM covert channels and VM provisioning. Finally, Section VIII concludes this paper.

II. Preliminary Experiments

In this section, we first introduce the background of scheduling in hypervisor. Then, we present our study on timing-based cross-VM covert channel and the co-run probability between two VMs. After that, we discuss the system overhead regarding the vCPUs scheduling on a server.

A. Hypervisor and Scheduling

In practice, a hypervisor usually splits CPU time to consecutive time slots and assigns them to VMs, thus each VM owns the run time of CPU cores in a fair manner. A hypervisor has different ways to schedule VMs across CPU cores, which determines the co-run probability among VMs. A hypervisor can schedule VMs co-resident on a server statically or dynamically where the former makes the scheduling decision before VMs' execution and the latter makes the scheduling decision at runtime. Dynamic scheduling algorithms are now widely applied in hypervisors, *i.e.*, *Credit Scheduler* in XEN.

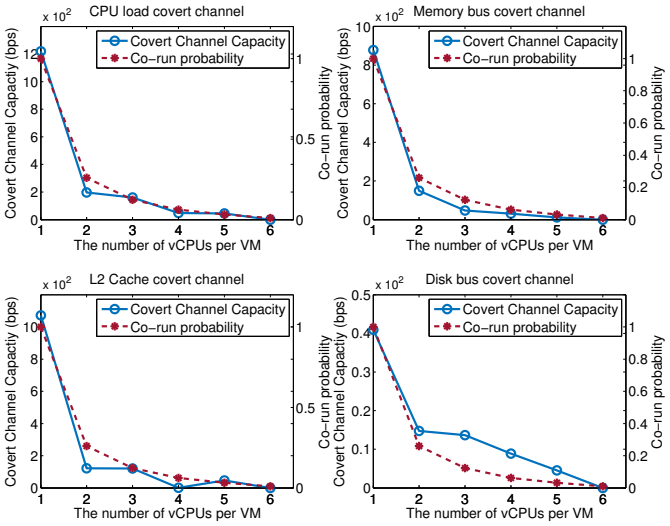


Fig. 2. Capacities of four types of cross-VM covert channels and co-run probabilities between two VMs with different VMs.

However, existing dynamic scheduling cannot ensure that the co-run probability between any two co-resident VMs will not be more than the security level specified by the tenant. In this paper, to control the co-run probability between two VMs, we introduce a static scheduling algorithm which can ensure the co-run probability between any two VMs launched on a server will not be more than the security level specified by the tenant.

B. Cross-VM Covert Channels and Co-run Probability

If a VM is configured with multiple vCPUs, in theory, the attacker may launch several sender and receiver processes to leak and receive data. However, it is very difficult to do so because it is hard to synchronize between multiple covert channels with multiple pairs of senders and receivers to ensure the correct sequences. Therefore, it is more practical to launch only one sender process and one receiver process to construct a covert channel between a pair of VMs co-resident on a server. Hence we will only consider the co-run probability between one vCPU pair co-resident on a server. With such an assumption, we give the following definitions:

Definition 1. *The co-run probability between two vCPUs is the proportion of time slots during which two vCPUs are scheduled to run simultaneously in a scheduling period.*

Definition 2. *The co-run probability between two VMs is the maximum co-run probability between any vCPU pairs, of which the two vCPUs belong to the two VMs respectively.*

In our study, we have conducted extensive experiments to show that, as the co-run probability between two VMs decreases, the covert channel risk is reduced. All the experiments in this paper are conducted on a Dell PowerEdge T620 server with Xen 4.4.0 hypervisor and Ubuntu 14.04.1 LTS. The server is configured with two CPU cores and two VMs. Six groups of experiments are conducted. In each group, the number of vCPUs of each VM varies from 1 to 6. In each VM, we launch CPU-intensive applications which mainly consume CPU resources. The cross-VM covert channel capacity is measured among any two VMs based on the framework proposed in [16], which utilizes the *Shannon entropy formulation* and presents a fully automatic capacity profiler. In this framework, one VM acts as the sender and the other one acts as the receiver. The sender transmits sampling data to the receiver as well as ground truth. With the sampling data and the ground truth, the receiver calculates the capacity. Meanwhile, the co-run probability between the sender and the receiver is measured by calculating the proportion of co-run time slots among all time slots. For each group, three rounds of measurements are conducted and the average values are reported. The results are shown in Fig. 2. We can see that both the cross-VM covert channel capacity and the co-run probability between two vCPUs decrease rapidly and are nearly identical as the number of vCPUs increases, which demonstrates the strong correlation between co-run probability and risk of covert channels.

C. Overhead Modelling

When a VM is created with multiple vCPUs, there are two sources of overheads, caused by the hypervisor and by the multithreaded scheduling. The hypervisor overhead is for the hypervisor to schedule vCPUs and it is at the operating system level. As the number of vCPUs increases, the system overhead raises accordingly. The overhead introduced by multithreaded design is at the application level. For a given multithreaded task, the introduced overhead is due to the software design or architecture. Therefore, when a given task is executed in a multithreaded environment, it typically takes more accumulated CPU time than the same task executed sequentially.

To verify the above viewpoints and measure the overhead of our design, we conduct a few experiments with PARSEC 3.0, a popular benchmark suite composed of multithreaded programs. We utilize "vips", which is an image processing system and consumes much CPU time during the running time. The benchmark program runs multiple rounds in a series of VMs. Except the vCPU, all VMs have the same configuration. In Fig. 3, we report the overhead ratio under different number of vCPUs which varies from 1 to 18. For a VM with n vCPUs, the overhead ratio is as follows.

$$f(n) = \frac{t(n) - t_0}{t_0} \tag{1}$$

Here, $t(n)$ is the run time and t_0 is the baseline run time. The baseline run time is measured when the VM is assigned with only one vCPU. We can see that, when the number of vCPUs is less than 12, the overhead is smaller than 5%. However, as the number of vCPUs reaches 16, the overhead quickly reaches around 10%. To obtain $f(n)$, we fit a curve with data points in Fig. 3 and get $f(n) = 0.005629(n-1)$ on our platform.

III. Mitigation Framework Design

In this section, we present the framework of mitigation solution. We consider a cloud with of thousands of servers. There are one or more VMs running on each server. We

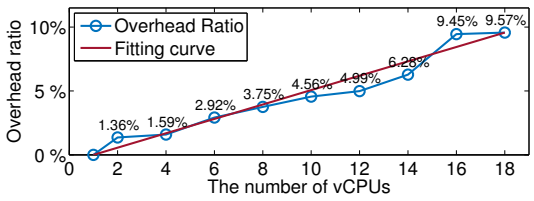


Fig. 3. Overhead analysis with different number of vCPUs on a server.

assume that each VM runs CPU-intensive tasks. For simplicity, we only consider the CPU resources and assume all servers have the same hardware configuration, *i.e.*, the number of CPU cores and the configuration of all CPU cores are the same among all servers. Each server can be virtualized into different number of vCPUs, according to which, servers can be classified into different types. Let $S = \{1, 2, \cdots, I\}$ be the set of server types. We denote the lth server with type k as $s_{k,l}$ and $s_{k,l}$ is virtualized to n_k vCPUs. The number of currently utilized vCPUs, n'_k, is no more than n_k. To provide each VM with a consistent and predictable amount of CPU capacity, we define the *Physical Computing Unit* (PCU) as the computing capacity of one CPU core, which is similar to *EC2 Compute Unit* (ECU) defined in Amazon EC2.

Let m be the number of CPU cores on a server. When $n_k < m$, each vCPU is executed in one CPU core dedicatedly and can get up to one CPU core's computing power. Otherwise, n_k vCPUs share m CPU cores. For type k servers, $k \in S$, c_k denotes the *computing power* of each vCPU in terms of PCU. Therefore, we have:

$$c_k = \begin{cases} 1, & \text{if } n_k \leq m; \\ \frac{m}{n_k}, & \text{otherwise.} \end{cases} \tag{2}$$

For a given type k, n_k is fixed, and thus c_k is fixed, no matter how many vCPUs are running on this server. So, the computing power of each vCPU varies when n_k changes. Since the increase of n_k leads to a higher system overhead, the minimum ratio $\frac{m}{n_k}$ should be limited. For example, Amazon EC2 guarantees the minimum CPU utilization for each $t2.micro$ server, *i.e.*, the minimum computing power of each vCPU, as 10%. In this paper, we set the smallest $\frac{m}{n_k}$ as 10%.

As we have discussed in Section II, covert channel capacity decreases as the increase of co-run probability between two VMs. The lower the co-run probability, the higher the security level. Since co-run probability may be hard for the tenants to understand or specify, the cloud provider can offer security level options $O = \{o_1, o_2, \cdots, o_I\}$, where each security level option corresponds to certain co-run probability. With the above analysis, we give two definitions as follows:

Definition 3 (Feasible Scheduling)**.** *A scheduling is feasible if it satisfies that (1) when $m < n_k$, in each time slot, m vCPUs are scheduled to run, which means all CPU cores are utilized; and (2) in a scheduling period with T time slots which can be sufficiently large, the number of time slots assigned to each vCPU are equal.*

Definition 4 (The Min-cost VM Provisioning and Scheduling (MVPS) Problem)**.** *For a newly arrived VM with computing power and security level requirements, the objective is to find a server and a feasible scheduling, while achieving the minimum cost caused by the provisioning.*

TABLE I

Notations	Meaning
S	the set of server types
$s_{k,l}$	lth server with type k
m	the number of CPU cores on a server
d	the number of VMs on a server
n	the number of vCPUs on a server
n_k	the number of vCPUs on a server with type k
$\binom{n}{m}$	the number of combinations of selecting m from n
v_i	VM i
$\bar{v}_a$	vCPU a
V	the set of vCPUs on a server
V_i	the set of vCPUs of VM v_i
I	the number of server types
c_k	computing power of each vCPU on a server with type k
q_r	a request submitted by the tenant to launch a VM
u_r	amount of computing power required by request q_r
o_r	security level required by request q_r
P_k	the maximum co-run probability between any two VMs on a server with type k
$P_{i,j}$	the co-run probability between VM v_i and VM v_j
$p_{a,b}$	the maximum co-run probability between vCPU $\bar{v}_a$ and vCPU $\bar{v}_b$
T	the number of time slots in a scheduling period

To solve the MVPS problem, we decompose this problem to two subproblems: (1) a VM scheduling problem on a single server, which determines whether a given server fulfills the security requirement of a request; and (2) a VM provisioning problem among servers, which selects an appropriate server among all servers satisfying the security requirement to accommodate the newly arrived VM. We present the solutions of such two subproblems in Sections IV and V, respectively.

To facilitate the discussions, in Table I, we list all the notations used in the paper.

IV. VM Scheduling on a Single Server

In this section, we first formulate the VM scheduling problem as a binary integer linear program (BILP), then present an efficient *Equal Scheduling* algorithm and derive theoretical analysis of *Equal Scheduling*.

A. Problem Formulation

Suppose that a server $s_{k,l}$ has m CPU cores and n_k vCPUs. There are d' VMs currently running on $s_{k,l}$. Without loss of generality, we denote the running VMs on $s_{k,l}$ by $\{v_1, v_2, \cdots, v_{d'}\}$ where v_i consumes a set of vCPUs V_i. The total number of currently utilized vCPUs is $n_{k'} = \sum_{i=1}^{d'} |V_i|$. For each VM pair, v_i and v_j, the co-run probability between them is denoted as $P_{i,j}$.

The optimal scheduling problem on a single server $s_{k,l}$ is defined as follows. Upon receiving a VM request $q_r = \{u_r, o_r\}$, $s_{k,l}$ will not be considered if the remaining vCPUs in server $s_{k,l}$ cannot satisfy the computing power requirement u_r. If $(n_k - n'_k)c_k \geq u_r$, the objective of the problem is to find a feasible scheduling to minimize $P_{i,j}$ between any

VM pairs v_i and v_j such that the scheduling satisfies the security requirement for q_r and does not violate the security requirements of existing VMs. Depending on the requirements of existing VMs and q_r, the feasible scheduling may not exist. In this case, $s_{k,l}$ will not be considered for VM request q_r. This problem can be formulated as a BILP.

In the BILP, binary variable $x_{a,t}$ denotes whether vCPU $\bar{v}_a$ is scheduled on time slot t. $\bar{x}_{a,b,t}$ stands for whether VM v_a and VM v_b co-run at time slot t. θ_i is defined as the co-run probability requirement of VM v_i, corresponding to o_i. To ensure that each vCPU has the same computing power, we assume the scheduling period T is a sufficiently large number such that the number of time slots each vCPU occupied, $\frac{Tm}{n_k}$, is an integer. To simplify the formulation, when the newly arrived VM is assigned to this server, it is denoted as $v_{d'+1}$. If all unutilized vCPUs are assigned to $v_{d'+1}$, which means $\lceil \frac{u_r n_k}{m} \rceil = n_k - n'_k$, then the total number of VMs $d = d' + 1$. Otherwise, when $\lceil \frac{u_r n_k}{m} \rceil < n_k - n'_k$, we assume the remaining unutilized vCPUs, $n_k - n'_k - \lceil \frac{u_r n_k}{m} \rceil$, are assigned to a virtual VM $v_{d'+2}$ and then the total number of VMs $d = d' + 2$. The formulations are as follows.

Minimize: P

Subject to:

$$\sum_{a=1}^{n_k} x_{a,t} = m, \forall t \in [1, T] \tag{3}$$

$$\sum_{t=1}^{T} x_{a,t} = \frac{Tm}{n_k}, \forall a \in [1, n_k] \tag{4}$$

$$\bar{x}_{a,b,t} \geq x_{a,t} + x_{b,t} - 1, \quad \forall a \in [1, n_k - 1], t \in [1, T], b \in [a+1, n_k] \tag{5}$$

$$0 \leq \bar{x}_{a,b,t} \leq x_{b,t}, \forall a \in [1, n_k - 1], t \in [1, T], b \in [a+1, n_k] \tag{6}$$

$$0 \leq \bar{x}_{a,b,t} \leq x_{a,t}, \forall a \in [1, n_k - 1], t \in [1, T], b \in [a+1, n_k] \tag{7}$$

$$\bar{x}_{a,b,t} \in \{0, 1\}, \forall a \in [1, n_k - 1], t \in [1, T], b \in [a+1, n_k] \tag{8}$$

$$x_{a,t} \in \{0, 1\}, \forall a \in [1, n_k - 1], t \in [1, T] \tag{9}$$

$$P_{i,j} \geq \frac{\sum_{t=1}^{T} \bar{x}_{a,b,t}}{T}, \quad \forall \bar{v}_a \in V_i, \bar{v}_b \in V_j, a < b, i \in [1, d-1], j \in [i+1, d] \tag{10}$$

$$\begin{cases} P_{i,j} \leq \theta_i, \forall i \in [1, d-1], j \in [i+1, d], \text{if } \lceil \frac{u_r n_k}{m} \rceil = n_k - n'_k \\ P_{i,j} \leq \theta_i, \forall i \in [1, d-2], j \in [i+1, d-1], \text{if } \lceil \frac{u_r n_k}{m} \rceil < n_k - n'_k \end{cases} \tag{11}$$

$$\begin{cases} P_{i,j} \leq \theta_i, \forall i \in [1, j-1], j \in [2, d], \text{if } \lceil \frac{u_r n_k}{m} \rceil = n_k - n'_k \\ P_{i,j} \leq \theta_i, \forall i \in [1, j-1], j \in [2, d-1], \text{if } \lceil \frac{u_r n_k}{m} \rceil < n_k - n'_k \end{cases} \tag{12}$$

$$\begin{cases} P \geq P_{i,j}, \forall i \in [1, d-1], j \in [i+1, d], \text{if } \lceil \frac{u_r n_k}{m} \rceil = n_k - n'_k \\ P \geq P_{i,j}, \forall i \in [1, d-2], j \in [i+1, d-1], \text{if } \lceil \frac{u_r n_k}{m} \rceil < n_k - n'_k \end{cases} \tag{13}$$

Constraints 3 and 4 present the two constraints of a feasible scheduling. Constraints 5, 6, and 7 make sure that binary variable $\bar{x}_{a,b,t}$ equals to 1 if and only if vCPU $\bar{v}_a$ and vCPU $\bar{v}_b$ are scheduled to run simultaneously at time slot t. Constraints 8 and 9 show that variables $x_{a,t}$ and $\bar{x}_{a,b,t}$ are binary variables. Constraint 10 restricts the co-run probability $P_{i,j}$ between VM v_i and VM v_j to the maximum co-run probability between any vCPU pairs, of which the two vCPUs belong to the two VMs respectively. Constraints 11 and 12 make sure that the co-run probability $P_{i,j}$ is no more than the co-run probability θ_i of VMs required by VM v_i, corresponding to the security level required by VM v_i. These two sets of constraints also show $P_{i,j}$ is restricted only if any one of VM v_i and VM v_j is not a virtual VM. Constraint 13 gives the maximum co-run probability between any two VMs.

If the BILP has *no feasible solution*, current server $s_{k,l}$ will not be considered for q_r. If the BILP has a feasible solution, it gives an optimal scheduling for the newly arrived VM and n'_k currently utilized vCPUs on this server. Once a server is selected for q_r on the provisioning solution, the server will apply the scheduling solution of the BILP to all vCPUs except the vCPUs of the virtual VM, where the scheduler reserves the time slots for later provisioning.

Although BILP can determine whether a VM can be accommodated on a server and provides an optimal scheduling to minimize the risk of covert channels among co-resident VMs if the newly arrived VM can be accommodated on it, it has the following two disadvantages. (1) When a new VM arrives, whether it can be accommodated by a server depends on not only the server type k, but also the security requirements of existing VMs on the sever. Thus, the BILP should always be applied to each server to verify the existence of the feasible scheduling. Considering the number of servers for checking, the VM provisioning time will be quite long. (2) Even if an optimal solution is found, the scheduling of all the existing VMs should be changed according to the optimal scheduling.

With the above consideration, we try to solve the problem by proposing a more efficient and effective scheduling algorithm, with which the maximum co-run probability between any two VMs is only determined by the type of the server, *i.e.*, the number of vCPUs. It is independent of what VMs have been provisioned to the server. In this way, for a given server type k, we can directly determine whether the newly arrived VM can be accommodated by a server without violating the security level requirement. Therefore, the cloud only needs to check the computing power requirement of the newly arrived VM for each server, which can significantly reduce the provisioning time. The design of *Equal Scheduling* is shown as follows.

B. Equal Scheduling

In this subsection, we first introduce *Equal Scheduling* and show the co-run probability between vCPU pairs under *Equal Scheduling*. We then give rigorous theoretical analysis to show that *Equal Scheduling* can minimize the maximum co-run probability between any two VMs in the worst case.

The design of *Equal Scheduling* is shown as follows. In a server configured with m CPU cores and n vCPUs, each vCPU belongs to one VM. For each time slot, the scheduler selects m vCPUs from n vCPUs to run on m CPU cores. We denote the set of combinations of selecting m vCPUs from n vCPUs as a set $K = \{k_1, k_2, \cdots, k_{|K|}\}$, where, $|K| = \binom{n}{m}$. For each combination k_i in K, the scheduler selects corresponding vCPUs in k_i to run on m CPU cores. As the example shown in Fig. 4, each column represents a time slot and the shadow rectangular in row i indicates vCPU i is scheduled to run

Algorithm 1 Equal Scheduling

Input: m, the number of CPU cores; n, the number of vCPUs.
1: K = The set of all combinations of selecting m vCPUs from n vCPUs, $|K| = \binom{n}{m}$.
2: **for** ith time slot **do**
3: $i' = i \bmod |K|$
4: Schedule each vCPU in $k_{i'}$ to one CPU core.
5: **end for**

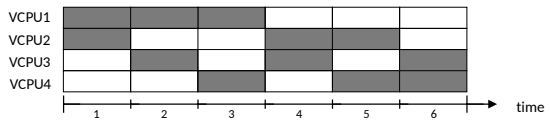


Fig. 4. An example of *Equal Scheduling* with $m = 2, n = 4$.

in current time slot. In this example, the co-run probability between any vCPU pair is $\frac{1}{6}$. The details of *Equal Scheduling* is presented in Algorithm 1.

Then, we analyze the complexity of *Equal Scheduling*. The first part of the algorithm is to list all the combinations of selecting m vCPUs from n vCPUs, which requires $O(\binom{n}{m})$ time. For each time slot, scheduling vCPUs requires $O(n)$ time. Thus, the complexity of *Equal Scheduling* is $O(n\binom{n}{m})$.

Next, we give theoretical analysis in following theorems.

Lemma 1. *When Equal Scheduling is applied, the co-run probability between each vCPU pair $\bar{v}_a$ and $\bar{v}_b$, $a \neq b$, is*

$$p_{a,b} = \frac{m(m-1)}{n(n-1)}. \tag{14}$$

Proof. For given m and n the scheduling period $T = \binom{n}{m}$. Since there are m vCPUs running in each time slot, for a given vCPU pair $\bar{v}_p$ and $\bar{v}_b$, the number of combination of selecting $m-2$ vCPUs from $n-2$ vCPUs is $\binom{n-2}{m-2}$. Therefore, the number of time slots that the vCPU pair $\bar{v}_a$ and $\bar{v}_b$ are running simultaneously in one scheduling period is $\binom{n-2}{m-2}$. We get $p_{a,b} = \frac{\binom{2}{2}\binom{n-2}{m-2}}{\binom{n}{m}} = \frac{m(m-1)}{n(n-1)}$. □

According to Lemma 1, we can see that the co-run probabilities between any vCPU pair are the same. Thus, we have the following Theorem.

Theorem 1. *With Equal Scheduling, the co-run probability between any two VMs v_i and v_j, $i \neq j$, is*

$$P_{i,j} = \frac{m(m-1)}{n(n-1)}. \tag{15}$$

Proof. According to Definition 2,

$$P_{i,j} = \max_{\bar{v}_a \in V_i, \bar{v}_b \in V_j, i \neq j} p_{a,b} = \frac{m(m-1)}{n(n-1)}.$$

□

We will show that for any feasible scheduling, the sum of co-run probability of all vCPU pairs is a constant.

Lemma 2. *For each feasible scheduling Ω, $\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} p_{a,b}^{\Omega} = \binom{m}{2}$, in which $p_{a,b}^{\Omega}$ denotes the co-run probability of vCPU $\bar{v}_a$ and vCPU $\bar{v}_b$ in scheduling Ω.*

Proof. During the scheduling period T, we denote the number of time slots that two vCPUs $\bar{v}_a$ and $\bar{v}_b$ are running simultaneously as $Y_{a,b}$. Let $y_{a,b,t} \in \{0,1\}$ indicates whether two vCPUs $\bar{v}_a$ and $\bar{v}_b$ are running simultaneously in time slot t or not. Therefore, $Y_{a,b} = \sum_{t=1}^{T} y_{a,b,t}$. We have:

$$\begin{aligned}\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} p_{a,b}^{\Omega} &= \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} \frac{Y_{a,b}}{T} \\ &= \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} \frac{\sum_{t=1}^{T} y_{a,b,t}}{T} \\ &= \sum_{t=1}^{T} \frac{\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} y_{a,b,t}}{T}.\end{aligned}$$

For each time slot, there are m running vCPUs. Therefore, for time slot t, $\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} y_{a,b,t} = \binom{m}{2}$. Thus we have

$$\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} p_{a,b}^{\Omega} = \sum_{t=1}^{T} \frac{\binom{m}{2}}{T} = \binom{m}{2}.$$

□

The next Lemma will show that for any feasible scheduling, the maximum value of co-run probability of all vCPU pairs is no less than $\frac{m(m-1)}{n(n-1)}$.

Lemma 3. *For any feasible scheduling Ω, the maximum co-run probability between any vCPU pairs, $\max_{a \neq b} p_{a,b}^{\Omega} \geq \frac{m(m-1)}{n(n-1)}$.*

Proof. We prove the Lemma by contradiction. Suppose that $\max_{a \neq b} p_{a,b}^{\Omega} < \frac{m(m-1)}{n(n-1)}$, we have:

$$\begin{aligned}\sum_{a=1}^{n-1}\sum_{b=a+1}^{n} p_{a,b}^{\Omega} &< \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} \max_{a \neq b} p_{a,b}^{\Omega} < \sum_{a=1}^{n-1}\sum_{b=a+1}^{n} \frac{m(m-1)}{n(n-1)} \\ &= \binom{n}{2}\frac{m(m-1)}{n(n-1)} = \binom{m}{2}.\end{aligned}$$

The above equation contradicts with Lemma 2. Therefore, we have $\max_{a \neq b} p_{a,b}^{\Omega} \geq \frac{m(m-1)}{n(n-1)}$. □

The *VM distribution* is defined as $\Phi = \{\phi_1, \cdots, \phi_d\}$, in which $\phi_i > 0$ and $\sum_{i=1}^{d} \phi_i = n$. The VM distribution Φ means there are d VMs to be lunched on the server and VM v_i is configured by ϕ_i vCPUs. We also denote the co-run probability of VM v_i and VM v_j under the VM distribution Φ and scheduling Ω as $P_{i,j}^{\Omega,\Phi}$.

Theorem 2. *For any feasible scheduling Ω, the maximum value of the maximum co-run probability between any VM pairs in all VM distributions, $\max_{\forall \Phi} \max_{i \neq j} P_{i,j}^{\Omega,\Phi} \geq \frac{m(m-1)}{n(n-1)}$.*

Proof. For each vCPU pair, $\bar{v}_a$ and $\bar{v}_b$, there exists a VM distribution Φ^*, in which $\bar{v}_a$ and $\bar{v}_b$ are configured on different VMs. Consequently, for the VM distribution Φ^*, $\max_{i \neq j} P_{i,j}^{\Omega,\Phi^*} \geq p_{a,b}^{\Omega}$.

Therefore, $\max_{\forall\Phi}\max_{i\neq j} P_{i,j}^{\Omega,\Phi} \geq \max_{a\neq b} p_{a,b}^{\Omega}$. Since Lemma3 shows that $\max_{a\neq b} p_{a,b}^{\Omega} \geq \frac{m(m-1)}{n(n-1)}$, $\max_{\forall\Phi}\max_{i\neq j} P_{i,j}^{\Omega,\Phi} \geq \frac{m(m-1)}{n(n-1)}$. □

Theorem 3. *Equal Scheduling can minimize the maximum co-run probability between any two VMs in the worst case.*

Proof. Theorem 2 shows for any feasible scheduling Ω, there exists a VM distribution so that the maximum co-run probability that Ω can achieve under that VM distribution is no less than $\frac{m(m-1)}{n(n-1)}$. According to Theorem 1, since for any VM distribution, the co-run probability between any two VMs is $\frac{m(m-1)}{n(n-1)}$. Therefore, *Equal Scheduling* minimizes the maximum co-run probability of VMs in the worst case. □

Corollary 1. *For a given VM distribution $\Phi^* = \{\phi_1, \cdots, \phi_d\}$, in which $\phi_i = 1, \forall i \in \{1, \cdots, d\}$, Equal Scheduling minimizes the maximum co-run probability between any two VMs among all feasible scheduling.*

Proof. We prove the Corollary by contradiction. Suppose that there exists a feasible scheduling Ω that the maximum co-run probability between any two VMs is less than $\frac{m(m-1)}{n(n-1)}$. Since the $\phi_i = 1, \forall i \in \{1, \cdots, d\}$, the co-run probability between any two VMs is equal to the co-run probability of the corresponding vCPU pairs, which means that the maximum co-run probability of vCPU pairs is less than $\frac{m(m-1)}{n(n-1)}$. Since Lemma 2 shows that the sum of the co-run probability of different vCPU pairs is $\binom{m}{2}$, the maximum co-run probability of vCPU pairs is no less than $\frac{\binom{m}{2}}{\binom{n}{2}} = \frac{m(m-1)}{n(n-1)}$, which is a contradiction. Therefore, for the VM distribution Φ^*, the minimum co-run probability between any two VMs is $\frac{m(m-1)}{n(n-1)}$, which is achieved by *Equal Scheduling*. □

Theorem 3 shows that *Equal Scheduling* achieves the optimal co-run probability of VMs in the worst case. Corollary 1 shows that *Equal Scheduling* achieves the optimal co-run probability of VMs when each VM has only one vCPU.

V. VM Provisioning among Servers

In this section, we aim to design an efficient VM provisioning algorithm among multiple servers to guarantee the security and resource requirements of requests while maximizing the server utilization.

Since all VM pairs on a server have the same co-run probability under *Equal Scheduling* principle, we abuse the notation a little bit. Let P_k be the co-run probability between any two VMs on a type k server under *Equal Scheduling*. As we know from Eq. (15), for fixed m, with the increase of n, the co-run probability between two VMs declines proportionally. Thus, we have the following theorem.

Theorem 4. *If the server with type k' matches the requirement, any type k'' server with more vCPUs than $n_{k'}$ also fulfills the security requirement.*

Proof. From Eq. (15), we have $P_{k'} - P_{k''} = \frac{m(m-1)}{n_{k'}(n_{k'}-1)} - \frac{m(m-1)}{n_{k''}(n_{k''}-1)}$. Since $n_{k'} < n_{k''}$, we have $P_{k'} - P_{k''} > 0$, which means that the risk level of a server with type k'' is smaller than the server with type k'. Thus servers with type k'' fulfill the security requirement. □

We denote the server types, which fulfill the requirements as S'. To determine which type of servers from S' to accommodate a request q_r, we take the following cost factors into consideration.

- Over provisioning cost, denoted as w_1. Over provisioning cost is incurred when the total computing power allocated to a request is more than what it requests.
- The cost for launching a new server, denoted as w_2.
- System overhead, denote as w_3.

For a given server type k, $k \in S'$, if none of existing $s_{k,l}$ fulfills the computing power requirement of q_r, the scheduler has two options: (a) assigning the VM to an active server with another type k', (b) assigning the VM to a newly launched server with type k. We discuss these two options separately.

Applying option (a) introduces over provisioning cost w_1 but without w_2. Since the cost of running a VM is caused due to the resource usage during the runtime and the resource will be released once the VM is destroyed, we define the cost to be proportional to the VM's runtime. Thus, the over provisioning cost incurs until the termination of the VM. We denote the runtime of a VM by $\bar{T}_{run}$. Thus, we have

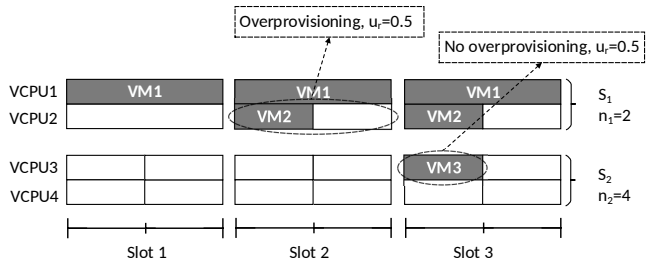
$$w_1 = (\lceil \frac{u_r}{c_k} \rceil n_k - u_r)\bar{T}_{run}. \tag{16}$$

If option (b) is applied, more servers are launched than that in option (a). However, launching a new server now may avoid launching a new server in the near future and the newly launched server could be used for the coming VM requests. We only consider the cost of running a newly launched server before the next VM request arrives. Let VM inter-arrival time be $\bar{T}_{int}$. We have

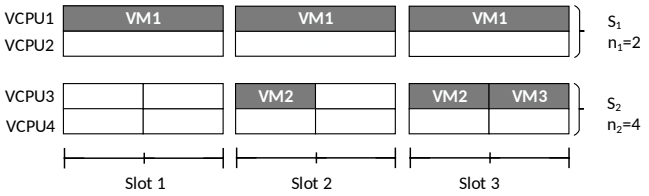
$$w_2 = (m - u_r)\bar{T}_{int}. \tag{17}$$

In Fig. 5, we use an example to show that the decision of choosing option (a) or option (b) depends on future request arrivals. In time slot 1, only *VM 1* is provisioned. The requests for *VM 2* and *VM 3* come in time slots 2 and 3, respectively. In scenario 1, option (a) is applied. *VM 2* is placed on the first server to avoid launching a new server in time slot 2. However, to provision *VM 3*, a new server is still needed. In scenario 2, we apply option (b) and *VM 2* is placed on a new server in time slot 2. Comparing these two scenarios, we can see that the cost for launching a new server in scenario 2 lasts for one more time slot than that in scenario 1. Meanwhile, an over provisioning cost is introduced in scenario 1.

To make a decision, one critical issue is how to predict the runtime of a VM $\bar{T}_{run}$ and the time interval between two requests $\bar{T}_{int}$. Using historical information, techniques like Linear Regression [17], Machine Learning [18], and Sliding Window method [19] can be utilized to predict $\bar{T}_{run}$ and $\bar{T}_{int}$. Here, considering the online requirement of the prediction, we adopt a Sliding Window based method. First, we keep a vector



(a) Scenario 1



(b) Scenario 2

Fig. 5. An example denotes the cost to launch a new server for a request.

Algorithm 2 Minimum Cost Provisioning (*MCP*)

Input: a VM request $q_r = \{u_r, o_r\}$
Output: the placement for the request
1: update $\bar{T}_{int}$ and $\bar{T}_{run}$
2: **for all** $k \in S$ **do**
3: **if** $P_k \leq \frac{1}{o_r}$ **then**
4: push k into S'
5: **end if**
6: **end for**
7: **for all** server type $k \in S'$ **do**
8: $w_1 = (\lceil \frac{u_r}{c_k} \rceil n_k - u_r)\bar{T}_{run}$
9: $w_2 = 0$
10: **if** none of active server $s_{k,l}$ can accommodate q_r **then**
11: $w_2 = (m - u_r)\bar{T}_{int}$
12: **end if**
13: $w_3 = f(n_k)\bar{T}_{run}$
14: $cost(k) = w_1 + w_2 + w_3$
15: **end for**
16: $k' = \arg\min_{k \in S'} cost(k)$
17: **if** exists active servers $s_{k',l}$ can accommodate q_r **then**
18: push eligible servers with type k' into queue Q
19: find the server $s_{k',l}$, the remaining computing power of which is the minimum one in Q
20: launch a VM for q_r on $s_{k',l}$
21: **else**
22: launch a VM for q_r on a new server with type k'
23: **end if**

of size K, which consists K historical data collected in the recent period of time. Then, we get the average value in the vector as the predicted output.

Similar to the over provisioning cost, the system overhead cost exists when the VM is running. Thus, we have

$$w_3 = f(n_k)\bar{T}_{run}. \tag{18}$$

Based on the above cost definitions, we propose a *Minimum Cost Provisioning* algorithm which is shown in Algorithm 2. We also analyze the computational complexity of *MCP*. To determine the server type, k', the algorithm takes time $O(I)$ while choosing the server $s_{k',l}$ takes time $O(l)$. Thus, the complexity of *MCP* is $O(I + l)$.

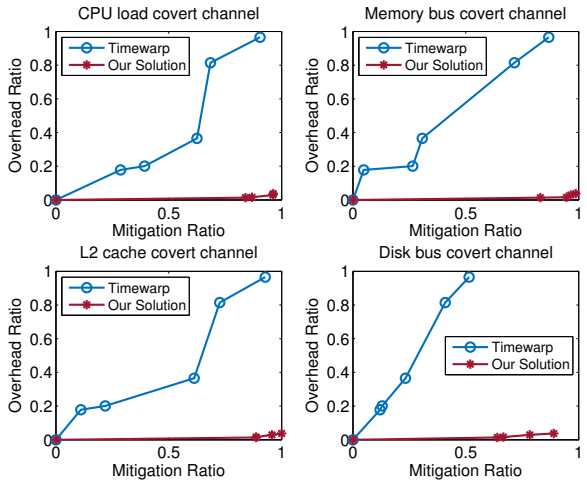


Fig. 6. The covert channel mitigation performance against the system overhead for *TimeWarp* and *Equal Scheduling*, respectively.

VI. Evaluation

In this section, we first compare *Equal Scheduling* with another generic mitigation method, *TimeWarp* [11], in terms of mitigation performance and introduced system overhead. Then we evaluate the VM provisioning algorithm regarding the number of servers used and the cost of the provider.

A. Mitigation Performance Evaluation

To evaluate the mitigation performance of our solution, we compare the results with another generic mitigation method, *TimeWarp*. The idea of *TimeWarp* is to introduce delays to *RDTSC* instructions in the hypervisor. With this solution, VMs cannot access high privileged timing functions and noise is introduced to cross-VM covert channels. Therefore, the risk of covert channels is mitigated.

In the experiments, we report the performance of the algorithms from two perspectives, mitigation ratio and corresponding system overhead introduced by the mitigation method. The mitigation ratio is measured as the decrease ratio of the covert channel capacity when a mitigation solution is deployed, compared with the measurement with no mitigation solution. The system overhead is measured with the method introduced in Section II-C. The experiment settings are the same as that in Section II-B. In our observations, different mitigation levels lead to different system overhead. For each mitigation solution, we measure the cross-VM covert channel capacity and the system overhead with different parameters. For *TimeWarp*, we vary the fuzziness of the *RDTSC* and tested six different fuzziness granularities, which range from 0 to 100,000 cycles. Five different VM configurations are evaluated for *Equal Scheduling*, where the number of vCPUs varies in the range of $\{2, 4, 6, 8, 10\}$ and the number of CPU cores is 2.

The results are presented in Fig. 6, which shows the same trend for different cross-VM covert channels. When *TimeWarp* is applied, the overhead increases rapidly with the increase of mitigation performance, while the results of *Equal Scheduling*

indicate that the overhead is insignificant compared with that introduced by *TimeWarp*.

B. Provisioning Algorithm Evaluation

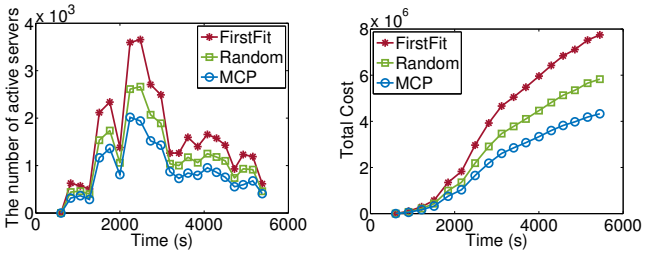
We conduct a simulation based on Google cluster traces to evaluate the VM provisioning algorithm in terms of two metrics, namely, the number of servers used and the cost of the provider. For given VM requests, the first metric simply counts on the instant active servers and the second metric is the accumulative cost of the provider.

1) Dataset Description: The Google cluster-usage traces [20] are used as the data set, which is collected from a Google server cluster of about 12,000 machines. The traces contain a large number of job records, where each job consists of several tasks and each task is assigned to one machine. In order to reduce the computational complexity, we choose a continuous subset of the traces and get around 49K task records. We treat each task as a request and extract three attributes that are relevant to our simulation, namely, task start time, task finish time, and task resource requirement for CPU cores. In the traces, task start time and finish time are provided precisely. The CPU core requirement is provided with scaled values in the range [0, 1]. To simulate real requests, we rescale the CPU core requirements to the range [1, 16]. There are 6 different server types, which is due to the fact that a public cloud generally offers less than 6 CPU options, like Amazon EC2. Corresponding to the 6 server types, 6 different security levels are provided. Since the security level requirement for each request cannot be obtained from the trace, we generate the security level requirements following the normal distribution.

2) Simulation Results: We compare the performance of our algorithm (*MCP*) with two conventional VM provisioning algorithms, the random scheduler (*Random*) and the first-fit scheduler (*FirstFit*), which is widely used [21]. All these provisioning algorithms only consider the servers that fulfill the security requirements of VMs under *Equal Scheduling* for provisioning. The Random scheduler selects a server randomly that fulfills the computing power requirements. The *FirstFit* scheduler chooses the first one among servers that fulfill the computing requirements. The evaluation results are shown in Fig. 7(a) and Fig. 7(b). Fig. 7(a) shows the performance comparison regarding the number of active servers at each time point with three schedulers respectively. We can see that *FirstFit* always launches the maximum number of active servers, followed by *Random*. *MCP* requires less active servers to fulfill the requests. In average, compared with *FirstFit* and *Random*, *MCP* uses 43.91% and 26.15% less servers, respectively. Fig. 7(b) shows the total cost up to the current time point. We can see that our algorithm is more effective with the increase of running time. At the last time point, our algorithm saves cost up to 44.12% and 25.74% compared to *FirstFit* and *Random*, respectively.

VII. Related Work

Cross-VM covert channel attacks were first demonstrated by Ristenpart *et al.* [3] in 2009. Later, several cross-VM covert channels were presented with different shared resources such as CPU [2], memory [3], [4], CPU cache [5], [7], and disk [3]. A lot of mitigation methods have been discussed along with the presentation of cross-VM covert channel attacks. Existing mitigation methods include securing VMs' isolation from the hardware layer [8], [9], offering dedicated servers [10], limiting VMs' access to high precision timing functions [11], applying access policy to hardware or the hypervisor [13], migrating VMs dynamically [15], and adjusting hypervisor core scheduling [22]. However, none of these methods are proposed from a cloud provider's perspective to jointly consider resource utilization and security guarantee in the cloud.



(a) (b)

Fig. 7. (a) The number of active servers. (b) The total cost.

Alternatively, numerous VM placement strategies have been proposed since the popularity of data centers. [23] proposes a power management technique, VirtualPower, which includes VM consolidation, hardware scaling, and soft resource scaling. EnaCloud [24] leverage VM migration to minimize the number of active servers and the power consumption. [25] utilizes the resource scaling feature to provide a fine-grained resource allocation solution. However, these VM provisioning solutions do not consider security factors, which is the main focus of our solution.

VIII. Conclusion and Future Work

In this paper, we have empirically demonstrated that the co-run probability among VMs contributes to timing-based cross-VM covert channel capacity. To effectively mitigate timing-based cross-VM covert channel, it is important to reduce the co-run probability between any two VMs, which can be achieved through VM provisioning and VM scheduling. To this end, we have designed an *Equal Scheduling* algorithm with the aim to minimize the maximum co-run probability between any two VMs and presented an effective algorithm to provision VMs. The proposed work offers a generic mitigation solution against known and unknown timing-based cross-VM covert channels.

As a future work, we plan to implement and evaluate our solution in a large cloud platform with plenty of servers. Although similar results should be observed with other hypervisors, such as KVM, Hyper-V, and VMWare, we would like to evaluate them in the future. We also intend to explore more generic cross-VM covert channel mitigation solutions.

ACKNOWLEDGEMENT

The work is supported in part by Hong Kong Research Grants Council under GRF project 122913, the National Natural Science Foundation of China under Grant 61202378, the China Postdoctoral Science Foundation under Grant 2013M531402 and Grant 2014T70544, the Application Foundation Research of Suzhou under Grant SYG201401, and the Shanghai Oriental Scholar Program.

REFERENCES

[1] B. Andrew, R. R. John, S. James, K. Khalid, C. Joanna, and W. Dominique, "The public cloud market is now in hypergrowth," Forrester Research, Tech. Rep., April 2014.

[2] K. Okamura and Y. Oyama, "Load-based covert channels between xen virtual machines," in *SAC '10*, pp. 173–180.

[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 *CCS '09*, pp. 199–212.

[4] Z. Wu, Z. Xu, and H. Wang, "Whispers in the hyper-space: High-speed covert channel attacks in the cloud," in *Security'12*, pp. 159–173.

[5] Y. Xu, M. Bailey, F. Jahanian, K. Joshi, M. Hiltunen, and R. Schlichting, "An exploration of l2 cache covert channels in virtualized environments," in *CCSW '11*, pp. 29–40.

[6] Z. Yang and P. Chen, "Exploring virtual machine covert channel via i/o performance interference," in *CloudCom-Asia '13*, pp. 232–239.

[7] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: Cross-cores cache covert channel," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, vol. 9148, 2015, pp. 46–64.

[8] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, 2007, pp. 494–505.

[9] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 4, p. 35, 2012.

[10] I. Amazon Web Services, "Amazon EC2 Dedicated Instances," https://aws.amazon.com/ec2/purchasing-options/dedicated-instances/, 2016.

[11] R. Martin, J. Demme, and S. Sethumadhavan, "TimeWarp: rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *ISCA '12*, pp. 118–129.

[12] B. Saltaformaggio, D. Xu, and X. Zhang, "Busmonitor: A hypervisor-based solution for memory bus covert channels," in *EuroSec '13*.

[13] T. Kim, M. Peinado, and G. Mainar-Ruiz, "Stealthmem: System-level protection against cache-based side channel attacks in the cloud," in *Security'12*, pp. 189–204.

[14] J. Wu, L. Ding, Y. Lin, N. Min-Allah, and Y. Wang, "XenPump: a new method to mitigate timing channel in cloud computing," in *CLOUD '12*, pp. 678–685.

[15] S.-J. Moon, V. Sekar, and M. K. Reiter, "Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration," in *CCS'15*. ACM, pp. 1595–1606.

[16] R. Zhang, W. Qi, and J. Wang, "Cross-vm covert channel risk assessment for cloud computing: An automated capacity profiler," in *ICNP '14*, pp. 25–36.

[17] S. Weisberg, *Applied linear regression*. John W. & S., 2005, vol. 528.

[18] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[19] T. G. Dietterich, "Machine learning for sequential data: A review," in *Structural, syntactic, and statistical pattern recognition*. Springer, 2002, pp. 15–30.

[20] J. Wilkes, "More Google cluster data," 2011, posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[21] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *CCGRID'09*, pp. 124–131.

[22] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-vm side-channels," in *Security'14*, pp. 687–702.

[23] R. Nathuji and K. Schwan, "Virtualpower: coordinated power management in virtualized enterprise systems," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 6, 2007, pp. 265–278.

[24] B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong, "Enacloud: An energy-saving application live placement approach for cloud computing environments," in *CLOUD '09*, pp. 17–24.

[25] M. Cardosa, M. R. Korupolu, and A. Singh, "Shares and utilities based power consolidation in virtualized server environments," in *IM '09*, pp. 327–334.