

4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
<http://www.univ-orleans.fr/lifo>

Rapport de Recherche

A Security-Aware Scheduler for Virtual Machines on IaaS Clouds

Zaina Afoulki, Zaina Afoulki, Jonathan
Rouzaud-Cornabas
LIFO, ENSI de Bourges

Rapport n° **RR-2011-08**

A Security-Aware Scheduler for Virtual Machines on IaaS Clouds

Zaina Afoulki
ENSI de Bourges
88, bld Lahitolle
Bourges, France
zaina.afoulki@ensi-
bourges.fr

Aline Bousquet
ENSI de Bourges
88, bld Lahitolle
Bourges, France
aline.bousquet@ensi-
bourges.fr

Jonathan
Rouzaud-Cornabas
LIFO – ENSI de Bourges
88, bld Lahitolle
Bourges, France
jonathan.rouzaud-
cornabas@ensi-
bourges.fr

ABSTRACT

With the number of services using virtualization and clouds growing faster and faster, it is common to mutualize thousands of virtual machines (VMs) within one distributed system. Consequently, the virtualized services, softwares, hardwares and infrastructures share the same physical resources. This has given rise to important challenges regarding the security of VMs and the importance of enforcing non-interference between those VMs. Indeed, cross-VM attacks are an important and real world threat. The problem is even worse in the case of adversary users hosted on the same hardware and therefore the isolation facility within clouds needs to be strong. Furthermore, each user has different adversaries and the placement and scheduling processes need to take these adversaries into account.

First, we show that the current isolation facility within clouds i.e. virtualization is weak and can be easily attacked. To avoid such vulnerabilities, we propose a security-aware scheduler that implements security policies expressing isolation requirements. The policies are expressed by the cloud users themselves by giving them the possibility to choose their own adversaries. Then, we show how we enforce these policies within our VM placement and migration algorithms. Finally, we present virtual networks with a better granularity for the clouds based on lists of trusted users. We conclude by presenting both physical and simulated experimental results showing the efficiency of our approach and the inability of other solutions to enforce such policies.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous

General Terms

Algorithms, Scheduling, Virtualization, Security

Keywords

Virtualization, Cloud Computing, Scheduling, Security, Isolation

1. INTRODUCTION

Nowadays, server farms are popular for running a large range of services from web hosting to enterprise systems or even HPC clusters. The common way to deal with those growing server farms is to mutualize services, softwares, hardwares and infrastructures using virtualization technologies. These mutualized and virtualized infrastructures are named clouds.

The concept of cloud computing goes back to 1961 with the introduction of Utility as a service by John McCarthy. The goal was to bill computing power as water or electricity. The computing power would have been provided by a large infrastructure such as a shared grid. In the past, building such infrastructure was impossible due to high costs. However, nowadays with affordable and efficient hardware, it has become possible.

Security is one of the top concerns in clouds [14]. An important part of cloud security is cloud trust that can be defined as “trust without touch” [11]. Indeed, the data is no longer within the company or the user’s computer but on an outsourced infrastructure. Accordingly, the data is on hardware that is outside the scope of the entity that owns it. There are two issues related to trust [10]: the lack of transparency of clouds’ infrastructure and unclear security assurances. Our goal is to increase security assurances by introducing an explicit isolation policy between users and transparency through traceability.

The security risks within clouds that we are interested in here are the threats that come from other users of the clouds. Indeed, the hardware can be shared between multiple users (multi-tenancy) and these users can be adversaries. Therefore, as stated in [16], “a customer’s VM can be assigned to the same physical server as their adversary”. The isolation of adversaries co-located on the same hardware can be broken to start an attack against the integrity, confidentiality and availability of other users’ VMs. This type of threats is described in section 2.

After presenting the vulnerabilities that can be used to spy on a VM from another one running on the same hardware, we present the cloud architecture we used for our implementation and experiments. In section 3, we start by briefly describing the OpenNebula virtual infrastructure manager and its advanced scheduler Haizea and the way they interact with each other. Then in section 4, we discuss the way we extended them to support a better isolation scheme between adversaries on an open public cloud. We introduce our solution for VMs placement and VMs migration and also the enforcement of a better isolation of virtual networks. In section 5, we present preliminary results of our solution and show that it is efficient to enhance the isolation between adversaries within a cloud.

2. CROSS-VM ATTACKS

The virtualization layer, often a hypervisor, seems to increase the isolation between multiple components (users, applications, operating systems, etc...) that are hosted on the same hardware. The way the hypervisor achieves this is by putting each component in a container, a VM, which helps preventing interference between each container. But virtualization must not be seen as a security component as it increases the attack surface of the system by bringing additional complex code. In fact, the hypervisors and other virtualization related facilities add new attack vectors. As a consequence, virtualization must not be the only isolation technology within the cloud.

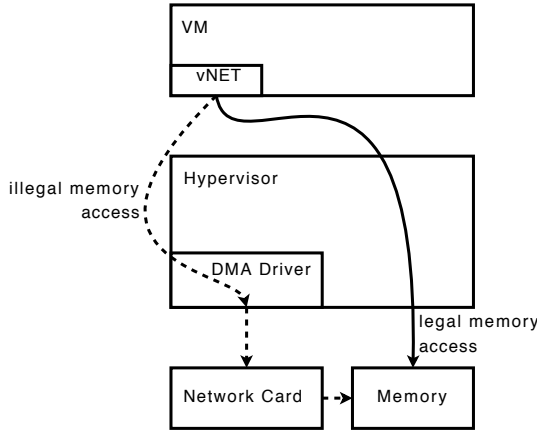


Figure 1: Example of a DMA attack

In our case, we are primarily interested in VM to VM attacks where a virtual machine tries to corrupt, disrupt or spy on another one. The common method used by attackers to achieve this is to attack the hypervisor from the VM and then use privilege escalation to attack another VM. This can be done through multiple vectors such as using classic exploits within the hypervisor. For instance a buffer overflow that would allow to inject code into the hypervisor's space.

Another vector has been well studied in [13, 21]. It exploits the ability of some drivers to directly access the underlying hardware from a VM. This can be done using Direct Memory Access (DMA). Indeed, these drivers can access all the physical memory without passing through the kernel or the hypervisor. Hence, a malicious DMA access can read / write all

the physical memory by bypassing the access control mechanisms of the OS and the hypervisor. As a consequence, a malicious VM that can exploit a fault in a DMA driver, has full access to the hypervisor and all its hosted VMs. Such an attack is described in figure 1 where an exploitable virtual network (vNet) driver is used to directly access the physical memory and bypass the hypervisor checks.

Finally, it is always possible for a VM to exploit the accounting of resources provided by the hypervisor [16]. In this case, the VM consumes most of the resources to block or slow down the other VMs. It can also spy on the resources consumption of other VMs to infer what they do.

These different attacks highlight the need of better isolation between adversaries on an open public cloud. Indeed, by placing VMs from adversary users on the same hardware, a cross-VM attack can occur corrupting the integrity, confidentiality and availability of the attacked VM. Therefore, as explained in [15] and as other components in the cloud, the VM scheduler needs to take into account security policies that describe security objectives when placing and migrating VMs. In this case, the security policy must enhance the isolation between adversary users.

3. VM MANAGEMENT MODEL AND EXPERIMENTATION FRAMEWORK

A virtual infrastructure manager (VIM) is required in order to deploy, control and monitor VMs over multiple physical computers (PCs) in a cloud. Several managers are already available: commercial products such as vSphere¹ from VMWare and open source alternatives such as OpenNebula [17]² or Eucalyptus [4]³. We have chosen OpenNebula to implement our scheduling and security models because of the open-source and research nature of the OpenNebula project.

By default, OpenNebula comes with a basic VM scheduler. This scheduler uses a match-making algorithm. The algorithm is divided into two steps. First, it creates a list of PCs that can host the VM i.e. PCs with enough resources available. Second, it evaluates an expression on each PC on this list. This expression implements a placement policy and computes a score for each PC. The PC with the highest score is chosen to start the VM. Three main placement policies exist:

- **Packing:** its purpose is to minimize the number of PCs on the cloud. Thus, it packs the VMs into a set of PCs to reduce VM fragmentation.
- **Striping:** its purpose is to maximize the quantity of resources available for each VM. Thus, it spreads the VMs on all PCs available.
- **Load-aware:** As for the striping policy, its purpose is to maximize the quantity of resources available for each VM. But contrary to the striping policy, it does not spread the VMs across all PCs but chooses the PC where to place the VMs based on the load of the PCs.

¹<http://www.vmware.com/products/vsphere>

²<http://www.opennebula.org>

³<http://www.eucalyptus.com>

Haizea [9] [17]⁴ is an advanced VM scheduler and can be used as a replacement for OpenNebula’s default scheduler. Haizea is a resource scheduler that can receive VMs requests and make scheduling decisions. OpenNebula and Haizea complement each other, since OpenNebula interacts with the hypervisors⁵ and VMs on each PC while Haizea provides the VM placement decisions. Using both allow resource providers to lease their resources, using potentially complex lease terms, instead of only allowing users to request VMs that must start immediately. As described in the figure 2, the user submits a lease request to the cloud infrastructure i.e. OpenNebula. This lease is forwarded to Haizea that uses it and its configuration file to compute a set of instructions. These instructions, start/stop/suspend/migrate VMs, are forwarded to OpenNebula which allows the placement of VMs contained within the lease.

Haizea “uses leases as a fundamental resource provisioning abstraction, and implements those leases as VMs.” Three types of leases can be used within Haizea:

- **Immediate leases** where requested VMs are scheduled to run at the time of request;
- **Advance Reservation leases (ARs)** where the requested VMs need to be run at a specific time in the future;
- **Best-Effort leases (BE)** where resources are provisioned as soon as possible and requests are placed on a queue if necessary.

For instance, best-effort leases can be preempted by AR leases by suspending the VMs of the best-effort leases before the start of the AR, and resuming them afterwards.

The scheduler’s decisions are translated into enactment commands and their corresponding VM states. These commands can be tested in a simulated setting within Haizea or they can be sent to the VIM, in our case OpenNebula. One of the major benefits of this second setting is that all the scheduling algorithms we have implemented are automatically compatible with the VIM. Thus, the results of our simulation studies can be verified in a real-world setting without much additional cost. In addition to testing our scheduling model in the simulation mode, we have experimented with a private IaaS cloud based on OpenNebula.

OpenNebula and Haizea provide a flexible infrastructure to build IaaS clouds. However, this infrastructure can also introduce non-obvious threats from other users as described in section 2. This is due primarily to the way physical resources can be shared between multiple VMs. The default configuration and setup of OpenNebula allows the creation of a multi-tenant cloud thus it allows the co-residence of VMs belonging to different users. In particular, to improve servers consolidation and reduce management and power costs, multiple VMs may be assigned to run on the same physical host. In the following section, we propose a new scheduling policy

⁴<http://haizea.cs.uchicago.edu>

⁵At the time of writing this paper, OpenNebula supports Xen, KVM and VMWare ESX

that takes into account adversaries to reduce the threat of cross-VM attacks.

4. A SECURITY-AWARE SCHEDULER

This section presents our implementation of a security-aware scheduler based on Haizea that enforces security policies dedicated toward a better isolation between adversary users. In practice, to avoid attacks between VMs running on the same hardware, the policy specifies a set of users that are adversaries. Then the policy is used by the scheduler to never place VMs that are owned by an adversary user on the same PC.

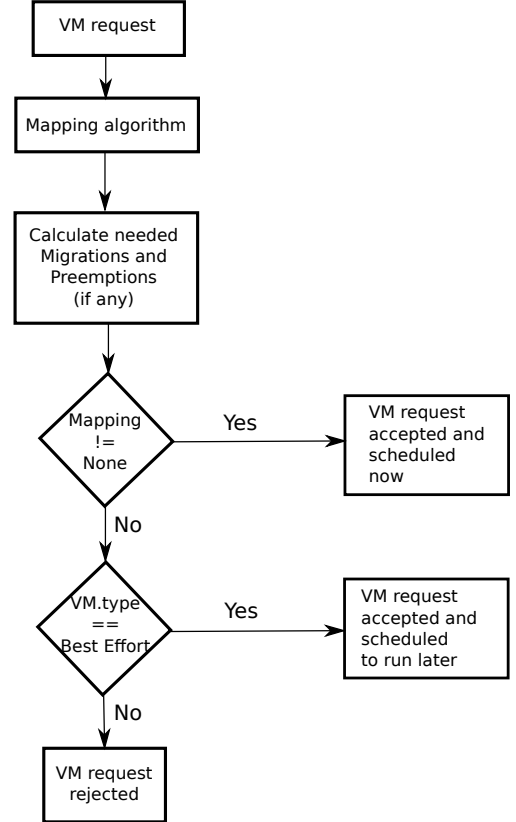


Figure 3: The VM placement algorithm

Our scheduling process works in four major steps as described in figure 3. First of all, the algorithm presented in the section 4.1 tries to find a valid placement for the requested VM without making any changes (preemptions or migrations) to the already running or scheduled VMs. Then, if the scheduler is unable to find any valid placement (due to a lack of resources or a restrictive placement policy) and if the requested VM is of high priority (AR), the scheduler tries to preempt one or more VMs of lesser priority (BE).

If the scheduler determines that even after this step, the requested VM cannot be scheduled (which may happen in case of a low priority VM), the scheduler tries to do some migrations as explained in the section 4.2. Once again if the scheduler fails to provide a placement for the requested

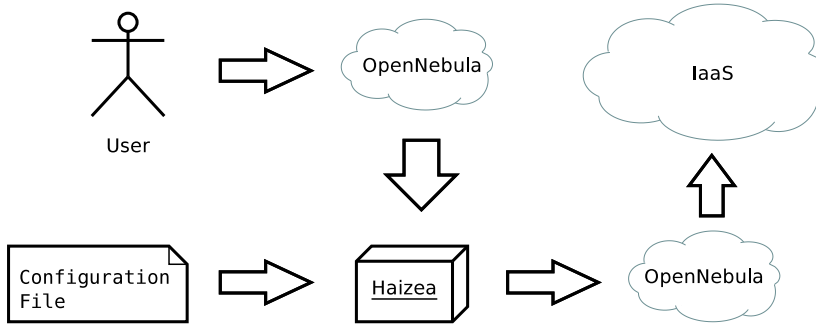


Figure 2: OpenNebula and Haizea

VM, two situations can happen: if the requested VM can be postponed (BE) then it is scheduled to run at a later date; if not, the VM request is canceled.

4.1 VM placement algorithm

As recommended in several publications [16, 15] and in order to build a secure cloud environment, we have implemented a solution based on incompatibilities between users. Each cloud user can submit a list of adversary users with whom he does not want to share a PC. All the lists of adversary users are then merged to create incompatible groups that are taken into account when scheduling a VM placement. A user can also choose to be placed on PCs running only his own VMs even if these PCs are under-utilized.

Each user group described above, contains two users: first, the one who declared the group, and then the user declared as an adversary. Listing 1 shows an extract of an XML groups definition file.

The same set of users can be described in two different groups. For instance, if user "company1" declares user "company2" as an adversary, and user "company2" does the same, there will be two groups (company1 -> company2) and (company2 -> company1). This results in numerous superfluous groups, however, we considered this to be a minor issue since it allows a user to update his adversaries list without impacting other users' lists. A user can also use the wildcard "*" to request dedicated PCs for himself.

Listing 1: Example of the XML groups definition file

```

<group user1="company1" user2="company2"/>
<group user1="company2" user2="company1"/>
<group user1="company1" user2="company3"/>
<group user1="company4" user2="*/>
  
```

When a user requests a VM, the scheduler computes the score of each PC to find the one that should be favored to receive the requested VM. When using our user's isolation policies, the scheduling algorithm will first of all determine if a PC is acceptable or not i.e. if there are adversary VMs running on it. If the PC is not running any adversary VMs, the scheduler applies the default greedy policy; should the opposite occurs, the PC is not considered for a mapping that does not involve any preemptions or migrations.

The default greedy policy scores PCs such that those with fewer VMs already scheduled on them, with the highest capacity and with fewest VMs scheduled in the future are scored highest. The greedy policy tries to minimize the number of preemptions by putting VMs on the least used PCs first so they can benefit the most from the available resources.

If the scheduler cannot find a placement without making any preemptions or migrations, it goes back to the process described in figure 3 and looks for some preemptions or migrations that would allow to satisfy the requested VM. This is done while keeping under consideration the incompatibilities between users. For example, if the chosen PC has some adversary VMs running, all of them will have to be preempted in order to start the new VM. The migration algorithm is explained in section 4.2.

Obviously, this user-based policy and especially the possibility of having no co-residence for a user reduces resources-utilization. This issue must be taken into account when billing the price of a VM. For example, the price of running a VM can be increased according to the number of adversaries the VM has.

We have chosen to implement a policy based on adversary users and not trusted users, which would have consisted of users that should be favored when two VMs have to run on the same physical computer. This choice is justified by constraints related to resources consolidation: in large-scale clouds, it will be more difficult to find a valid placement for a VM request that has a limited number of trusted users. In particular, the scheduler is less likely to find a placement for a VM that can only share a PC with another VM if it is trusted than if the VM had no adversaries at all. However, we can assume that most users are neither trusted nor adversary and therefore are not declared in any group. A solution could be to implement both types and allow co-residence of the same physical computer for two VMs that are neither trusted nor adversary, while favoring this co-residence with trusted users whenever possible.

4.2 Migration Algorithm

In a cloud environment, VM migration is used to solve various issues including, but not limited to, load balancing, placement policies and adjustments in the number of available PCs.

Since our scheduling algorithm uses a greedy policy to spread requested VMs across all available PCs and reject new VM requests when the maximum capacity of the PC is reached, it is unlikely that a physical computer will ever suffer from high load averages. Therefore there is no need to implement any migration schemes to leverage this case. Also, when a new PC is added to the cloud, the scheduler automatically maps all new VM requests to it until it obtains a smaller score than the other PCs. Again, there is not much need to implement migration for this use-case.

However, with our user's isolation policies we noticed that it is possible for a user to prevent another legitimate user from running any VMs at all. For instance, if a user requests a certain number of VMs as ARs, and then one of his adversaries wants to request a VM as well (either as an AR or a best-effort) the placement fails if all PCs are occupied by other adversary tenants. The straightforward solution to this situation is to have a migration algorithm that would attempt to migrate adversary VMs to other PCs in order to satisfy the requested VM. The placement of these adversary VMs elsewhere needs to comply with our security policies as well. This section details the migration algorithm we implemented to solve this kind of situations.

Data: Set of Nodes, requested_vm
foreach *Node i* **do**
 if *max_resources(i) >= requested_vm.resources* **then**
 node_score(i) = (*incompatible_vms* +
 incompatible_users + 1) * (*used_cpu*/*max_cpu* +
 used_mem/*max_mem*)/2
 else
 node_score(i) = -1
 end
end

Algorithm 1: Scoring of physical computers

We start by generating a sorted list of PCs using a specific score for which the pseudo-code is described in Algorithm 1. This score tells whether it is desirable to migrate VMs from a PC or not. The elements that are taken into consideration when computing this score for a PC are:

- The number of adversary VMs running in the PC i.e. the number of VMs belonging to an untrusted user
- The number of adversary users who own the running VMs
- The available resources in the physical computer (free Memory and CPU).

If a PC does not have enough resources to satisfy the requested VM, even when no VM is running on it, we do not take it as candidate PC. This is particularly useful when handling VMs in a cloud environment where PCs are heterogeneous. An example might be when a VM is requesting 1.5Gb of RAM and the PC is empty but has only a total of 1Gb RAM.

Basically, the fewer adversary VMs on a PC and the more available resources it has, the more interesting it is to migrate VMs from it. The PCs are sorted in order to have those

with the smallest score on the top of the list. We select the first PC in the list to simulate migration decisions and make sure that a valid mapping can be found for every VM that needs to be migrated, and that after these migrations, the requested VM can be safely placed on this PC.

We first of all look into the running VMs in the PC and then assign a score to each one. This score indicates the order in which we will try to migrate VMs in. Obviously we need to migrate all the adversary VMs first, so those are given a penalizing score. The scoring of the other VMs is simply based on the requested resources by each VM. So as to have a sorted list of VMs with those with the smallest amount of requested resources in the top of the list. Indeed, as stated in [7, 1], the live migration's time of a VM is linked with the amount of memory it uses.

As described in the pseudo-code of Algorithm 2: after simulating the migration of all adversary VMs and if it is not sufficient to map the requested VM, we simulate the migration of other VMs in the same PC. We start by calculating the remaining resources (CPU and Memory) needed to map the requested VM. Then we loop through all the remaining VMs to find the VM that will free the closest amount of needed resources if migrated, while freeing enough resources to allow the requested VM to run.

Once again if the mapping of the requested VM is still unsuccessful we try with the remaining VMs and start by migrating VMs that requested the smallest amount of resources (since those are easier to place elsewhere). We do this until a mapping is found or there are no more leases left on the candidate PC.

Data: Set of VMs, requested_vm
Result: Mapping for the requested VM
mapping \leftarrow None
for *vm* **in** *vms_in_node* **do**
 if *vm.score* == -1 **then**
 simulate_migration(vm)
 vms_in_node.pop()
 end
end
mapping = *map(requested_lease)*
if *mapping* == None **and** *vms_in_node* remaining **then**
 optimal_vm = *compute_optimal_vm_to_migrate()*
 simulate_migration(optimal_vm)
 vms_in_node.pop()
end
mapping = *map(requested_lease)*
while *mapping* == None **and** *vms_in_node* remaining **do**
 for *vm* **in** *vms_in_node* **do**
 simulate_migration(vm)
 vms_in_node.pop()
 mapping = *map(requested_lease)*
 if *mapping* != None **then**
 break
 end
 end
end

Algorithm 2: Simplified Migration Algorithm

At this point, if no mapping was found for the requested

interval of time, the VM request will be either rejected (if it was an AR) or scheduled to run later (best-effort).

However if a valid mapping is found, we make a final check by ensuring that a valid mapping can be found for each VM that needs migration. If all the above steps are accomplished without issues, we can confirm that *i)* If a number of VMs is migrated, there will be enough resources for the requested VM (in this PC), *ii)* a valid mapping (on other PCs) was found for each VM that will be migrated; and *iii)* that therefore the VM request can be accepted.

When a scheduler has migration capability one of the issue that can arise is over-migration. Over-migration arrives when a VM is migrated multiple time. It disreases the performance of the VM and increases the infrastructure usage. Futhermore, it can lead to cyclic migration where a VM is migrated to a PC and then migrated back to the original PC. To avoid this issue, we have extended Haizea to deny a migration if the VM has already been migrated between now and one year ago. It is not a bulletproof solution but our experimentations have shown it is enough in our case.

4.3 Network Isolation

In the previous sections, the solution we have presented allows the isolation of VMs in a cloud, so that no co-residence is tolerated between adversary VMs. This section details our implementation of a network isolation solution based on the concept of Trusted Virtual Domains (TVDs) [20]. TVDs allow the grouping of VMs belonging to a specific user dispersed across multiple PCs into a virtual network (vNet) in which isolation requirements are specified by user policies and automatically enforced.

A PC can be connected to one or more networks that are available to the VMs and since sharing the same network makes the vNets potentially vulnerable to a malicious VM, it is critical for our security model to provide a method of isolation for the networks so the traffic generated in different vNets cannot be seen in others.

Each vNet is defined as a set of MAC addresses and all VMs belonging to this vNet share the same pattern for their MAC address. For instance, two VMs on a given vNet (MAC addresses `02:00:C0:A8:01:*`) will obtain addresses such as `02:00:C0:A8:01:01` and `02:00:C0:A8:01:02`.

The vNets provided by OpenNebula are independent from physical networks or vLANs, which makes them flexible. OpenNebula provides a set of hooks (scripts which are executed each time a new VM is created, migrated and/or shut-down) with the aim of isolating the vNets. The hook called when deploying a VM adds a set of ebtables⁶ rules to the PC hosting the VM and the hook called on VM shutdown removes these rules. One of the rules added by the hooks drops packet from a MAC address that does not match the vNet's MAC address, whereas another rule prevents MAC spoofing by dropping all packets coming from an unknown MAC address.

Using these vNets and ebtables rules, we provide a mini-

⁶<http://ebtables.sourceforge.net/>

mum level of network isolation to all users at the Ethernet level. However the access control granularity of these vNets is weak as they can only be private and public. Therefore, we extended the vNets within OpenNebula to allow creation of multi-users vNets, which can be useful when several users need to share the same vNet. To achieve this, we have added a new option for private vNets that allows the user to specify a list of trusted users he is willing to share his vNet with. This is then implemented in the scheduler, that will reject any VM requested on a private vNet if the VM user is not the vNet's creator or on the vNet's trusted users list. Thus, we bring a better granularity in the access control of vNets.

5. EXPERIMENTATION

The following experiments (section 5.1, 5.2, 5.3, 5.4) explore small and self-contained cases on real hardware to validate the accuracy of our solution for efficiently schedule VMs while respecting the security policies that guarantee isolation of adversary users and isolation of virtual networks. We also present simulation experiments in the section 5.5. The purpose of these simulations is to show the linear scalability of our scheduler.

To evaluate our solution, we introduce a security policy with three users. Two of these users (Orange and SFR) are concurrent Internet Service Providers thus they are adversaries. Another one (Renault) is an automobile constructor and is not an adversary of the other groups. To modelize this policy, we expressed two rules described in Listing 2. All the VMs are requested by one of these users. In the experiments presented here, all the VMs use the same resources template i.e. they ask for the same amount of resources in terms of memory, CPU, etc.

Listing 2: Adversary rules

```
<group user1="orange" user2="sfr"/>
<group user1="sfr" user2="orange"/>
```

It should be noted that the same results are obtained if only one of the users (Orange or SFR) declared the other one as an adversary. For instance, if Orange declares SFR as an adversary and SFR do not declare any adversaries, then our scheduling algorithm still prevents the placement of SFR VMs in PCs populated with VMs belonging to Orange.

Our testbed consists of three PCs (nodes) with the same hardware configuration. One of the PCs is used as a head node (frontend) that hosts a shared NFS filesystem for all the nodes, while the remaining two PCs are used to run virtual machines. The frontend also runs OpenNebula 2.2 and Haizea 1.1 (the latest versions at the time of writing this paper), which manages all the VMs during the experiments.

5.1 Basic Placement

In our first experiment, SFR requests a VM. This VM is placed on the first PC (node #1). Indeed, both PCs are homogeneous and no VMs are running on them, so the scheduler chooses the first PC in the list. Then Orange asks for a VM, the scheduler places it on the second PC (node #2). As shown in listing 3, it detects that node #1 is already hosting a VM that is owned by an adversary user: SFR.

Listing 3: "Haizea logs for Orange's VM Placement"

[2011-01-06 16:47:16.02]	LSCHED	Lease #2 has been requested.
[2011-01-06 16:47:16.02]	LSCHED	Lease #2 has been marked as pending.
[2011-01-06 16:47:16.06]	LSCHED	Scheduling lease #2 (1 nodes) — AR
[2011-01-06 16:47:16.06]	LSCHED	From 2011-01-06 16:48:07.00 to 2011-01-06 16:53:07.00
[2011-01-06 16:47:16.06]	LEASES	Trying to schedule lease from user #1 on node#1
[2011-01-06 16:47:16.06]	LEASES	Node not available: user #2 is already running a VM here.
[2011-01-06 16:47:16.06]	LEASES	Trying to schedule lease from user #1 on node#2
[2011-01-06 16:47:16.11]	LSCHED	Lease #2 has been scheduled.

Then the user Renault requests a VM. Renault does not have adversary thus the scheduler just applies the greedy policies. As described in listing 4, the scheduler chooses the first PC as both PCs have the same resources and all VMs use the same amount of resources. The VMs placement returned by OpenNebula is therefore as described in listing 5

As this experiment shows, the policy effectively forbids two adversary users from running VMs on the same PC. The downfall of this policy is that it can refuse a VM request if all available PCs have adversary VMs even when they have enough resources to host the requested VM. Thus, the overall consolidation of the cloud is reduced. Accordingly, this solution has scalability issues as with a growing number of users, the number of rules in the policy can grow exponentially. This can lead to a worst case scenario where every user is the adversary of everyone else, which leads to having dedicated PCs for each user. As we have explained earlier, the enforcing of this security policy must have an additional financial cost for the user requesting it. This cost can be weighed by the number of adversaries the user has set.

5.2 Placement with preemption

Without preemption capability, the scheduler will not be able to manage different levels of VM requests. For example, as explained in section 3, a best-effort lease can be suspended (preempted) by an AR lease. Using preemption to free resources for a high priority lease is common practice in resource provisioning schedulers, but in our case, freeing resources is not the only goal. We also have to suspend some adversary VMs to enforce our security policies.

The following example illustrates a situation where preemption is used in order to start an AR lease requested by a user who has adversaries on all the available PCs of the cloud. To show the preemption capability of the scheduler, we introduce an example where the user Orange submits an AR lease. At the same time, as described in listing 6, six VMs are already running on the cloud. Four of them are owned by the user SFR and two by Renault. All of those VMs are best-effort. VMs owned by SFR are spread across the two PCs of the cloud. Therefore, launching the VM requested by Orange will require the suspension of one or more VMs owned by SFR.

Listing 7 shows what happens when the Orange lease is submitted. First, the scheduler refuses to place the VM on either PCs (node #1 and #2) as there are adversary VMs running on them. However, as both PCs are identical, the first node is selected and all VMs that are adversaries with the VM contained in the requested lease are suspended. In this example,

VMs 635 and 637 are suspended.

Listing 8 describes the state of the cloud after the placement of the AR VM. It shows that the two VMs belonging to an adversary user of Orange on PC node #1 have been suspended. However, a third VM that was also running on the same PC was not preempted since it is owned by Renault who is not an adversary of Orange. So the VM is still running after the AR VM has been placed and started.

5.3 Placement with migration

With preemption capability but without migration capability, the scheduler is able to manage different levels of VM requests but is led to suspend some VMs when it could have migrated them to other PCs.

In this section we exhibit a situation similar to the previous example but in the case of our scheduler having both capabilities. Also the VMs requested by SFR are AR leases and cannot be preempted, and there is only one SFR VM (i.e. adversary VM) running on the PC node #1 whereas there are two of them on PC node #2.

Orange submits an AR lease. The state of the cloud at this moment is described in listing 9. The placement of the Orange lease requires that one or more SFR VMs are migrated to be able to start the requested VM. After establishing scores for both nodes, the scheduler determines that PC node #1 is the most suitable to receive the VM request (since it has the lowest number of adversaries). Accordingly, Haizea must migrate all the VMs that are adversaries of Orange: VM 3404 is then migrated and the lease request is started as shown in listing 11.

For further security during the placement process, each lease keeps track of the migrations and preemptions it caused, i.e. the lists of VMs that had to be migrated or preempted to make room for it to start. Right before starting the lease request, the scheduler verifies that those VMs have indeed been preempted or suspended by querying OpenNebula to make sure that there has not been an issue enacting those commands. If the VIM i.e. OpenNebula indicates any anomalies in this verification (which may happen if an error or failure has occurred in the suspension or migration processes due to network, ACPI issues, etc...), the lease request is canceled to avoid any possibility of having two adversary VMs running in the same PC. This check may lead to some leases being rejected whenever an anomaly is detected in the enactment commands, but it adds further security to our model and closes any window of opportunity for an attacker to exploit such scenarios.

Listing 4: "Haizea logs for Renault's VM Placement"

[2011-01-06 16:49:07.02]	LSCHED	Lease #3 has been requested.
[2011-01-06 16:49:07.02]	LSCHED	Lease #3 has been marked as pending.
[2011-01-06 16:49:07.07]	LSCHED	Queued lease request #3, 1 nodes for 01:00:00.00.
[2011-01-06 16:49:07.11]	LSCHED	Next request in the queue is lease 3. Attempting to schedule...
[2011-01-06 16:49:07.11]	LEASES	Trying to schedule lease from user #4 on node#1
[2011-01-06 16:49:07.11]	LEASES	Trying to schedule lease from user #4 on node#2
[2011-01-06 16:49:07.12]	VMSCHED	Lease #3 can be scheduled on nodes [1] from 2011-01-06 16:49:17.00 to 2011-01-06 17:49:47.00

Listing 5: "VM Placement returned by OpenNebula"

ID	USER	NAME	STAT	CPU	MEM	HOSTNAME	TIME
613	sfr	sfr	runn	0	OK	node1 00	00:04:17
614	orange	orange	runn	0	OK	node2 00	00:02:59
615	renault	renault	runn	0	OK	node1 00	00:01:06

Listing 6: "Initial VM Placement returned by OpenNebula"

ID	USER	NAME	STAT	CPU	MEM	HOSTNAME	TIME
635	sfr	sfr	runn	0	OK	node1 00	00:03:14
636	sfr	sfr	runn	0	OK	node2 00	00:03:13
637	sfr	sfr	runn	0	OK	node1 00	00:03:12
638	sfr	sfr	runn	0	OK	node2 00	00:03:11
639	renault	renault	runn	0	OK	node1 00	00:02:06
640	renault	renault	runn	0	OK	node2 00	00:02:04

Listing 7: "Haizea logs for Orange's VM Placement with preemption"

[2011-01-06 17:16:48.02]	LSCHED	Lease #7 has been requested.
[2011-01-06 17:16:48.02]	LSCHED	Lease #7 has been marked as pending.
[2011-01-06 17:16:48.07]	LSCHED	Scheduling lease #7 (1 nodes) — AR
[2011-01-06 17:16:48.07]	LSCHED	From 2011-01-06 17:17:41.00 to 2011-01-06 17:22:41.00
[2011-01-06 17:16:48.07]	LEASES	Trying to schedule lease from user #1 on node#1
[2011-01-06 17:16:48.08]	LEASES	Node not available: user #2 is already running a VM here.
[2011-01-06 17:16:48.08]	LEASES	Trying to schedule lease from user #1 on node#2
[2011-01-06 17:16:48.08]	LEASES	Node not available: user #2 is already running a VM here.
[2011-01-06 17:16:48.08]	LSCHED	Must preempt leases [1, 3] to make room for lease #7
[2011-01-06 17:16:48.09]	LSCHED	Preempting lease #1...
[2011-01-06 17:16:48.09]	LSCHED	... lease #1 will be suspended at 2011-01-06 17:17:41.00.
[2011-01-06 17:16:48.10]	LSCHED	Preempting lease #3...
[2011-01-06 17:16:48.10]	LSCHED	... lease #3 will be suspended at 2011-01-06 17:17:41.00.
[2011-01-06 17:16:48.15]	LSCHED	Lease #7 has been scheduled.

Listing 8: "VM Placement returned by OpenNebula after preemption"

ID	USER	NAME	STAT	CPU	MEM	HOSTNAME	TIME
635	sfr	sfr	susp	0	OK	node1 00	00:06:26
636	sfr	sfr	runn	0	OK	node2 00	00:06:25
637	sfr	sfr	susp	0	OK	node1 00	00:06:24
638	sfr	sfr	runn	0	OK	node2 00	00:06:23
639	renault	renault	runn	0	OK	node1 00	00:05:18
640	renault	renault	runn	0	OK	node2 00	00:05:16
641	orange	orange	boot	0	OK	node1 00	00:02:59

Listing 9: "Initial VM Placement returned by OpenNebula"

ID	USER	NAME	STAT	CPU	MEM	HOSTNAME	TIME
3404	sfr	sfr	runn	0	OK	node1	00 00:03:28
3405	sfr	sfr	runn	0	OK	node2	00 00:03:27
3406	renault	renault	runn	0	OK	node2	00 00:02:19
3407	renault	renault	runn	0	OK	node1	00 00:02:18
3408	sfr	sfr	runn	0	OK	node2	00 00:00:52

Listing 10: "Haizea logs for Orange's VM placement with preemption and migration"

[2011-04-15 10:03:12.02]	LSCHED	Lease #6 has been requested.
[2011-04-15 10:03:12.02]	LSCHED	Lease #6 has been marked as pending.
[2011-04-15 10:03:12.07]	LSCHED	Scheduling lease #6 (1 nodes) — AR
[2011-04-15 10:03:12.07]	LSCHED	From 2011-04-15 10:03:55.00 to 2011-04-15 12:03:55.00
[2011-04-15 10:03:12.08]	LSCHED	Must migrate leases to make room for lease #6
[2011-04-15 10:03:12.08]	LSCHED	Migrating lease #1 to node #2
[2011-04-15 10:03:12.08]	LSCHED	Migrating lease #1...
[2011-04-15 10:03:12.08]	LSCHED	... lease #1 will be migrated at 2011-04-15 10:03:55.00.
[2011-04-15 10:03:12.12]	LSCHED	Lease #6 has been scheduled.

5.4 Virtual Networks

To validate our virtual network isolation policy, we have created four virtual networks described in listing 12: each user has his own private vNet and in addition, Orange has a public network. Moreover, Orange has created the private network by expressing a trusted users policy i.e. a list of users who have the right to connect to the private vNet. This policy permits only the user Renault to request the access to the private vNet. In this experiment we demonstrate that Orange's adversary SFR cannot use Orange's private network where as the user Renault can.

As described in listing 13, SFR requests a VM that is connected to the public network of Orange. As nothing is against connecting adversaries on a public network, the VM request is accepted and started. The same logs listing applies when Renault requests VMs that will be connected to either vNets owned by Orange (private and public).

However, as described in listing 14, SFR requests a VM that is connected to the private network of Orange. The scheduler is aware that only Orange and Renault VMs can join this network, hence it refuses the VM request without even trying to schedule it.

5.5 Simulations

To test the scalability of our scheduler, we use the simulation mode that is included with Haizea. We run numerous tests on a cloud that is composed of 2 and 2,000 PCs. Then, we evaluate the time to place between 2 and 3,000 VMs. In our simulation configuration, a PC can not host more than 4 VMs. Of course, when trying to start 3,000 VMs on 2 PCs, the majority of the requested VMs were not refused. Furthermore, in some case, the test can require migrations e.g. 30 VMs on 20 PCs and other time it is not the case e.g. 30 VMs on 100 PCs.

As shown in the figure 4, the placement time depends on the number of PCs on the cloud. Indeed, as the scheduler needs to compute a score (through the greedy policy) for each

PC on the cloud, the placement time is correlated with the number of PCs. For a cloud composed of 2 PCs, the scheduler places a VM in 100ms. But, when it tries to place 300 VMs, the scheduler loses a large amount of time to try, without success, to place VM. Thus, the failure of the placement (only 8 VMs are placed and 292 are refused) explains the higher placement time for 300 VM on 2 PCs. Furthermore, for the case where we try to place 3,000 VMs on 2 PCs (not display in the figure), the average placement time is even worse, around 6.75sec. The observations are the same for 20, 200 and 2,000 PCs.

The figure 5 presents the average placement time based on the number of PCs on the cloud. Contrary to the results presented in the figure 4, we only take into account the tests where all the VMs have been placed. Thus, the results do not take into account time wasted in trying to place VMs on an infrastructure without ressources available. The average placement time is binded linearly with the number of PCs. Thus, we can predict the placement time for a given cloud. Furthermore, it proves the scalability of our scheduler for a large cloud.

Based on these observations, we need to find way to optimize the placement process to include more parallelism. Indeed, if we can place multiple VMs at the same time, we can reduce the average placement time. But, it brings concurrency issues that must be studied. Moreover, these are simulations results that need to be validated by real world experiments.

6. RELATED WORKS

Entropy [7] is a virtual machine scheduler dedicated to consolidate a cluster. Entropy's goal is to reduce the number of physical machines that are running VMs to reduce energy consumption. Moreover, it takes into account the cost of VM migrations to avoid unnecessary ones. This is done by taking into account the amount of memory used by the VM. Indeed, the live migration time duration for a VM is linked to its memory usage. But Entropy is dedicated toward one goal, energy consumption, and does not take into account

Listing 11: "VM Placement returned by OpenNebula after migration"

ID	USER	NAME	STAT	CPU	MEM	HOSTNAME	TIME
3404	sfr	sfr	runn	0	0K	node2	00 00:05:13
3405	sfr	sfr	runn	0	0K	node2	00 00:05:12
3406	renault	renault	runn	0	0K	node2	00 00:04:04
3407	renault	renault	runn	0	0K	node1	00 00:04:03
3408	sfr	sfr	runn	0	0K	node2	00 00:02:37
3409	orange	orange	boot	0	0K	node1	00 00:01:24

Listing 12: "List of virtual Network"

ID	USER	NAME	TYPE	BRIDGE	P	#LEASES
15	sfr	sfr	Ranged	br0	N	0
16	orange	orange	Ranged	br0	N	0
17	orange	orange_public	Ranged	br0	Y	0
18	renault	renault	Ranged	br0	N	0

Listing 13: "Haizea logs for the placement of Renault's and SFR's VMs"

[2011-01-06 17:01:58.05]	LSCHED	Lease #1 has been requested.
[2011-01-06 17:01:58.05]	LSCHED	Lease #1 has been marked as pending.

Listing 14: "Haizea logs for SFR VM placement with Orange private network"

[2011-01-06 17:02:18.01]	LSCHED	Lease #2 has been requested.
[2011-01-06 17:02:18.02]	LSCHED	User_id 2 is not allowed to deploy leases on network 16
[2011-01-06 17:02:18.02]	LSCHED	Lease #2 has not been accepted

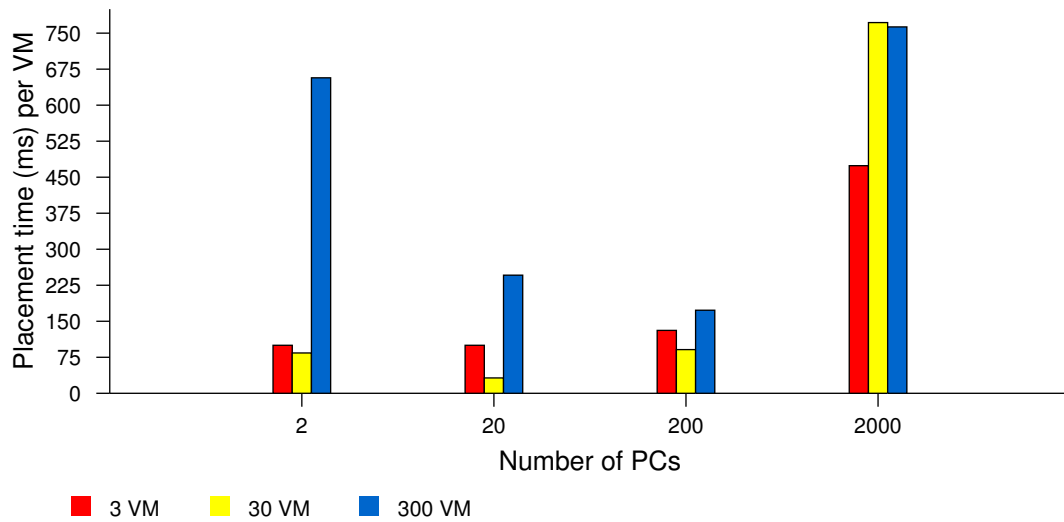


Figure 4: Average placement time for a VM

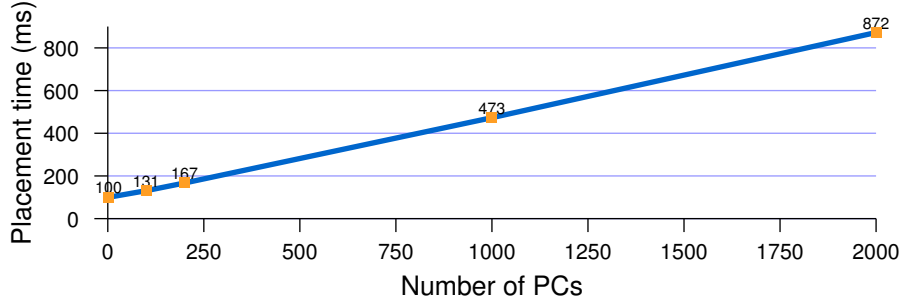


Figure 5: Average placement time for a successful VM placement

heterogeneous architectures. Furthermore, it can not take into account user objectives like our isolation policy.

Snooze [5] is another virtual machine scheduler dedicated toward reducing the energy consumption by minimizing the number of physical machines in the cloud. It uses a heuristic approach to find a placement solution. But, contrary to Entropy, their solution is not centralized. Snooze is scalable and fault-tolerant. But as Entropy, it is dedicated to one goal (energy consumption) and cannot take into account other ones.

In [19], the authors present a two-level cloud scheduler. First, the VM provisioning process places applications into a set of VMs. Then, the VMs are placed on physical machines. To solve the provisioning and placement problems, they use a constraint solver. Their placement time is between 800ms and 1,000ms. Moreover, the user's objectives are limited to one metric such as average response time of requests. They are not able to express isolation policies.

In [1, 22, 2], the authors present a software dedicated to the management of VMs migration for the consolidation of a cloud. The scheduler's goal is to choose which VM to migrate and when. As explained in [3], a local migration uses only few milliseconds whereas it uses 1-2 sec on a WAN [18]. This migration time can be reduced by the use of an InfiniBand network as explain in [8]. But they do not take into account isolation policies provided by users.

7. CONCLUSIONS

This paper discusses the problem presented in [16] where scheduling algorithms can be used to place VMs on the same PCs which leads to the risk of cross-VM attacks between adversary users in cloud environments. We show that cross-VM attacks are indeed possible and could be exploited to violate integrity, confidentiality and to cause availability issues with VMs. We concluded on the needs of a better isolation between users on the cloud. Then, we have presented the cloud infrastructure used in our experiments and justified why it is vulnerable to these cross-VM attacks.

We introduced a new security-aware scheduler that solves the issues discussed in previous sections between users in a cloud environment. This scheduler provides a way of placing and migrating VMs based on security policies that enable the

users of the cloud to express their isolation needs. We also introduced finer-grained virtual network access controls to allow a defined set of users to share a virtual network.

We have implemented the policies, algorithms and modified placement, migration and virtual networks on the Haizea scheduler which allowed us to test the efficiency of our approach with the simulation capability of Haizea but also in real world settings with the OpenNebula cloud infrastructure. Finally, we showed that currently, no other solution can leverage the risk of cross-VM attacks within clouds or allows the same kind of security policies we introduced in this paper.

Our first direction of future work will be to further evaluate the effectiveness of the presented scheduling approach. To obtain a complete view, a larger amount of VM requests and scenario combinations must be evaluated. We will also work on more extensive experimentations to test the scalability of our solution. Moreover, we want to evaluate our scheduler in a multi-site and geographically distributed cloud. Our second goal is to have a better understanding of the reduction of consolidation due to our policy: the purpose would be to propose a pricing scheme to bill our isolation facility in the clouds. Thirdly, we will work on implementing this isolation policy in other components to avoid covert channels within the cloud in order to prevent any attempt of breaking it. Also, we want to extend our policy to include more user's objectives to cover other aspects of security within clouds. For example, the user must be able to express the geographical localisation of his VMs. It is important as the laws change from a country to another one as described in [6, 12].

8. REFERENCES

- [1] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Michele Messori. Dynamic load management of virtual machines in cloud architectures. In *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 201–214. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-12636-914.
- [2] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*,

- pages 119–128. IEEE, 2007.
- [3] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.
 - [4] Nurmi D. Wolski R. Grzegorzczak C. Obertelli G. Soman S. Youseff L. Zagorodnov D. The eucalyptus open-source cloud-computing system. *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, 0:124–131, 2009.
 - [5] Eugen Feller, Louis Rilling, Christine Morin, Renaud Lottiaux, and Daniel Leprince. Snooze: A scalable, fault-tolerant and distributed consolidation manager for large-scale clusters. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, GREENCOM-CPSCOM '10, pages 125–132, Washington, DC, USA, 2010. IEEE Computer Society.
 - [6] James A. Hall and Stephen L. Liedtka. The sarbanes-oxley act: implications for large-scale it outsourcing. *Commun. ACM*, 50:95–100, March 2007.
 - [7] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 41–50, New York, NY, USA, 2009. ACM.
 - [8] Wei Huang, Qi Gao, Jiuxing Liu, and Dhabaleswar K. Panda. High performance virtual machine migration with rdma over modern interconnects. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 11–20, Washington, DC, USA, 2007. IEEE Computer Society.
 - [9] Sotomayor B. Keahey K. Foster I. Combining batch execution and leasing using virtual machines. *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 87–96, 2008.
 - [10] Khaled M. Khan and Qutaibah Malluhi. Establishing trust in cloud computing. *IT Professional*, 12:20–27, September 2010.
 - [11] Keith W. Miller, Jeffrey Voas, and Phil Laplante. In trust we trust. *Computer*, 43:85–87, October 2010.
 - [12] M. Mowbray. The fog over the grimpen mire: cloud computing and the law. *Script-ed Journal of Law, Technology and Society*, 6(1), 2009.
 - [13] J. Murakami. A hypervisor IPS based on hardware assisted virtualization technology. *Black Hat USA 2008*, 2008.
 - [14] Siani Pearson and Azzedine Benameur. Privacy, security and trust issues arising from cloud computing. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 693–702, Washington, DC, USA, 2010. IEEE Computer Society.
 - [15] Ravi Sandhu, Raj Boppana, Ram Krishnan, Jeff Reich, Todd Wolff, and Josh Zachry. Towards a discipline of mission-aware cloud computing. In *Proceedings of the 2010 ACM workshop on Cloud computing security workshop*, CCSW '10, pages 13–18, New York, NY, USA, 2010. ACM.
 - [16] Thomas Ristenpart Eran Tromer Hovav Shacham Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, 2009.
 - [17] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13(5):14–22, September 2009.
 - [18] Franco Travostino. Seamless live migration of virtual machines over the man/wan. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.
 - [19] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Proceedings of the 2009 Ninth IEEE International Conference on Computer and Information Technology - Volume 02*, CIT '09, pages 357–362, Washington, DC, USA, 2009. IEEE Computer Society.
 - [20] A. Bussani J.L. Griffin B. Jansen K. Julisch G. Karjoth H. Maruyama M. Nakamura R. Perez M. Schunter A. Tanner L. van Doorn E.V. Herreweghen M. Waidner and S. Yoshihama. Trusted virtual domains: Secure foundation for business and it services. *Research Report RC 23792*, IBM Research, November 2005.
 - [21] R. Wojtczuk. Subverting the Xen hypervisor. *BlackHat USA*, 2008.
 - [22] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923 – 2938, 2009. Virtualized Data Centers.