Improving users' isolation in IaaS: Virtual Machine Placement with Security Constraints

Eddy Caron*†, Jonathan Rouzaud-Cornabas*‡
* *Université de Lyon. LIP Laboratory.*
UMR CNRS - † ÉNS Lyon - ‡ INRIA - UCB Lyon 5668.
France
Email: FirstName.LastName@ens-lyon.fr

***Abstract*—Nowadays virtualization is used as the sole mechanism to isolate different users on Cloud platforms. We will show that due to improper virtualization of micro-architectural components, data leak and modification can occur on public Clouds. Furthermore, using the same vector, it is possible to induce performance interferences, *i.e.* noisy neighbors. Using this approach, a VM can slow down and steal resources from concurrent VMs. We propose placement heuristics that take into account isolation requirements. We modify three classical heuristics to take into account these requirements. Furthermore, we propose four new heuristics that take into account the hierarchy of the Cloud platforms and the isolation requirements. Finally, we evaluate these heuristics and compare them with the modified classical ones. We show that our heuristics are performing at least as good as classical ones but are scaling better and are faster by a few order of magnitude than the classical ones.**

***Keywords*-Security, Isolation, Hierarchical VM Placement, Covert-channel, Cloud Computing, IaaS**

I. Introduction

Virtualization is now widely used in modern data centers. Thank to mature software stacks and the widespread availability of platforms all over the world, the Cloud is now available for many applications of different kinds. Security and performance are the main goal users want to achieve when porting applications over IaaS or PaaS platforms. Security has been proven to be sometimes difficult to obtain [1] and several issues have been raised in public Clouds and public domain virtualization software stacks. Several different kinds of attacks and security issues can be observed that may lower the impact of Clouds. On the performance side, the expectations are higher than what can be actually obtained on today's public Clouds. Shared Physical Machines lead to performance degradation that are not appropriate for high performance applications. Isolation is then a critical issue both for security and performance concerns.

In this paper, we present the limitation of using virtualization technology as the sole approach to isolate workloads and users within a Cloud. In highly secured environments, strong isolation is done by unshared resources environment for two tasks with different security clearance. It is still the strongest defense against covert-channels (and other attacks). But this approach eliminates most of the current public and private Clouds but also the way how virtualization is used. With the widespread usage of virtualization, the need of strong isolation in such environment becomes critical.

First, in Section II, we present the micro-architecture of modern computer. We also present the virtualization limitations and how they can be exploited to attack security and privacy in the Clouds. Moreover, we show that the same issue exists for performance. In Section III, we introduce a set of VM placement heuristics that take into account the security requirements. Furthermore, we propose a set of optimization for these heuristics to improve performance and consolidation. In the Section IV, we present our evaluation methodology and explain our results. Finally, in the Section V, we conclude and present our future work.

II. Micro-Architecture: Where Virtualization Failed

Since the last few years, the complexity of physical machines' hardware topology has increased dramatically [2]. The number of cores, shared caches, and memory nodes have completely changed the micro-architecture of computers. From a simple CPU architecture during the Pentium era, we have now access to complex multi-core, multi-level caches that can be specific to a core, shared between some or all. Symmetric Multithreaded Processors (SMP) brings another level of hierarchy. SMP is a mean to share the resources of a core between multiple logical processors.

With the increasing number of cores, scalability becomes a major issue. To address it, modern proces-

sors use non-uniform interconnects[1]. This technology is named a Non-Uniform Memory Access (NUMA) architecture. But memory is not the only resources to be accessed through these non-uniform interconnects, Input/Ouput Devices accesses are done in a similar manner (Non-Uniform Input/Output Access – NUIOA). In this type of architecture, some cores have faster access than others to some I/O devices and memory banks [3]. For example, Figure 1 shows the inner architectural components of five modern platforms. As one can see, depending on whether the data is stored in a directly connected memory bank or in a remote one, the access to it will need to passthrough one (or more) CPU. The same is true for the I/O devices. Accordingly, the placement of tasks on CPU and their related data on memory banks is critical to exploit performance on these modern architecture.

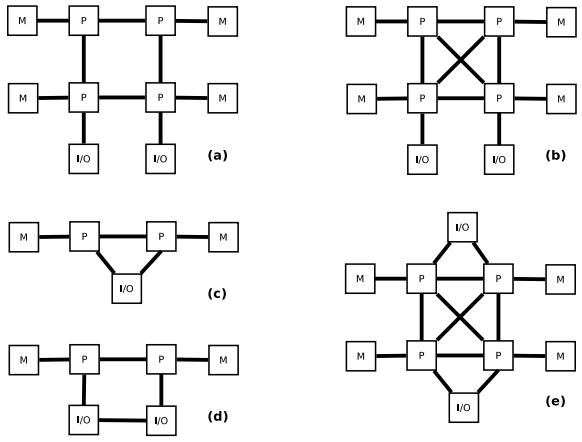

Figure 1. Interconnection of processors (P), memory (M) and I/O chipsets in some modern architectures [3]: (a) four AMD Istanbul processors; (b) four AMD Magny-Cours processors; (c) and (d) two Intel Westmere-EP processors; (e) four Intel Nehalem-EX processors.

The inner-topology of computer has a large impact on performance depending where the different process of an application are placed. For example, the DMA throughput can decrease by up to 42% when accessing a GPU from a distant NUMA node. Thus, exporting the topology to the VM is critical. Virtualized NUMA (vNUMA) is already available [4]. But there is not a one-to-one mapping between virtual and physical NUMA. Indeed, some physical NUMA can be shared through two vNUMA by VMs and two NUMA can be aggregated into one vNUMA. Accordingly, performance optimization can be buried by the vNUMA not really providing the same kind of hardware connections it exposes.

[1]HyperTransport for AMD and QPI for Intel.

A. Covert-Channel

Contrary to the memory and CPU, the micro-architectural component are not properly virtualized in modern platform. Therefore, the sharing of micro-architectural components can be used as covert channels to leak information between concurrent VMs. In [5], authors present new work on the creation of covert channels of communication between virtual machines through the L2 cache processors. Their goal is to quantify the rate of leakage through this channel. In previous work [1], three channels were studied: memory (0.006bps), disk (0.0005bps), and L2 cache (0.2bps). The purpose of [5] is to see if they can optimize the speed. In experimental testbeds, they were able to reach a rate of 223.71bps and between 1.27bps and 10.46bps on Amazon EC2. Other studies of covert channels within Clouds have been made in [6], [7].

In [8], authors present a new inter-VMs attack in the Cloud. Unlike previous works, they do not use the L2 cache CPU but the memory bus. This new channel allows them not to be limited to a set of cores sharing a cache but they can reach the entire physical machine. To do this, they observe the memory bus contention as covert channel. They are able to create a covert channel with a bandwidth of 746.8 bps (+/- 10.1) and between 343.5 bps (+/- 66.1) and 107.9 bps (+/- 39.9) on Amazon EC2. This improvement is at least a factor of 10 with approaches via the L2 cache.

B. Noisy Neighbors: Performance Interference

Virtualization is also used in Cloud Computing to provide performance isolation between multiple tenants. With a perfect performance isolation, no performance interference between two VMs must exist. Thus, noisy VM will have no impact on the performance of other VMs. Hypervisors implement resources allocation algorithms that fairly shares resources between different VMs. But these algorithms focus only on CPU time and memory capacity and do not take into account other resources [9]. Furthermore, current hardware virtualization does not provide mechanisms to limit micro-architectural components usage.

With the lack of strong isolation and in particular on micro-architectural components, the performance of a VM can suffer from interference coming from another one running on the same hardware. In [10], authors show that the performance loss can be up to 80% on a cache-sensitive benchmark because of the behavior of a collocated VM. But others works have shown the same type of performance loss can be achieved by

attacking shared memory [11] and [12]. Other works have studied the strong performance interference in Cloud Computing platforms [13], [14], [15]. As one can expect, all the shared resources (compute, storage and network) in Clouds are affected by interference issues.

The effectiveness of such interference could lead to a new class of attacks called Resource-Freeing Attacks (RFA) in [10]. *"A resource-freeing attack leverages these observations of contention to improve a* VM*'s performance by forcing a competing* VM *to saturate some bottleneck resources"*. The purpose of such attack would be to create interference that leads a VM to starve due to its inability to access specific micro-architectural components and thus freeing other resources. Accordingly, the VM that launches the attack could used these resources to boost its performance. Authors of [10] have shown they can gain a performance boost of up to 60% in lab and 13% on Amazon EC2.

C. Detection of Covert-Channel

Amazon provides a service that dedicates physical machines to an user: dedicated instances. But this service comes with a high price. [7] propose to continuously monitor memory access latencies and generates alarms when anomalies are detected. But memory probing incurs high performance overhead. Moreover, their approach has a high false positive rate.

VALID [16] is a specific language for security assurance. The goal is to verify the respect of security properties *i.e.* to detect violations of them. The language can express security goals such as isolation and generally express security properties based on information flow. Their approach is limited to detect isolation faults in network and is not able to do it for micro-architectures.

D. Improving Isolation within a Physical Machine

As we have presented in Section II-A, it is possible to create covert channels between VMs by using the lack of proper virtualization of micro-architectural components.

In [17], authors propose to improve the isolation of virtual machines while minimizing the loss of consolidation. The approach consists in offering better refinement within this isolation. They propose two algorithms: cache hierarchy aware core assignment and page coloring based cache partitioning. It limits the risk of breaking the isolation between virtual machines without having to use multiple physical machines. But their approach introduces a very large overhead that renders it inapplicable on real world Cloud platforms.

In [18], authors propose a protection that allows the hypervisor to counter attacks on CPU caches. Their approach has an overhead equivalent to 3 % loss of memory and cache. In addition, it brings an overhead of 5.9 to 7.2 % in terms of computation. Other works around software mechanisms to increase performance isolation exist for other resources: disk [19], memory bandwidth [20], and network [21].

The NoHype concept [22], [23] consists in removing the virtualization layer while retaining the key features enabled by virtualization. They limit covert-channel by enabling one VM per core but as we have show, other covert-channels exist within the micro-architecture components. Moreover, they are not able to do fair sharing on I/O and memory bus. Thus the performance isolation between VMs with NoHype remains weak.

E. Discussions

As we have shown in this section, the micro-architecture of modern platforms is evolving very fast. From a single CPU processor few years ago, we have now access to massively parallel platforms with complex hierarchy of micro-architectural components.

The current trend of security in Clouds is to use virtualization as the sole mechanism to enforce security and performance isolation between users and their VMs. But the lack of proper isolation of micro-architectural components lead to the ability of creating covert-channels between these VMs. It has been shown that it is possible to use these covert-channels to extract sensitive information such as cryptographic keys. But the security isolation is only one fold of a two folds issue. Performance isolation is also critical for Clouds. Indeed, if a VM is not able to efficiently use its available resources due to a noisy neighbor, it can lead to availability issue. As we have shown, a VM can applied such noisy neighbor behavior to slowdown a collocated VM. That leads the collocated VM to release resources and the VM who launches the attacks is able to extract more performance from the Cloud.

Motivation: As we have shown depending of the micro-architectural components shared between VMs, it is more or less effective to create covert-channels. The same is true for performance isolation. The sharing (or not) of micro-architectural components brings to complex security and performance trade-offs. Providing the tenant with the ability to specify the level of sharing his VMs can accept for micro-architectural components would give the ability to the user to configure the quality of isolation for each of his VMs. In the rest of the paper, we propose an approach to provide adaptable isolation. We show how it can be ported to any resource management system at node level (hypervisor) and at Cloud level.

III. Resource Allocation with Security Requirements

To help guide the placement of virtual machines, the main approach is to use constraints which reflect the goals expressed by the user and the entity that operates the infrastructure. By expressing the placement problem as Constraint Satisfaction Problems (CSP), it is possible to use a solver to resolve it. Constraints can represent different goals such as energy saving [24]. But solving large problems is time consuming thus using a decentralized and hierarchical approach can decrease the computation of a solution [25], [26]. The principle of using a linear program or a linear system to solve the problem of placement of virtual machines in the Cloud with constraints has been studied numerous times as in [27], [28]. The constraints can modelize numerous goals such as performance and availability [29]. These goals are then translate into collocation and anti-collocation constraints. As for the CSP approach, linear program approaches do not scale. To improve the performance of VM placement algorithm, hierarchical representation allows to reduce the search space [29]. Avoiding cache and processor resource contention through scheduling algorithms is a well studied approach [30], [31], [32]. At the cluster level, numerous works have also been done [33], [34]. The purpose of these approaches is to increase performance isolation between processes.

The current issue is to have efficient and scalable algorithms that are able to work at the scale of multi-datacenters Cloud infrastructure. CSP or linear program approaches do not scale well. Indeed, placing VMs on Clouds can be formalized as a multi-dimensional bin-packing. Therefore, the complexity of solving such problem is NP-Complete. Heuristics allow to have almost optimal solutions with a reduced complexity. In this paper, we propose to use such heuristics. They are able to take into account isolation requirements while placing VMs on a multi-datacenter Clouds or any infrastructures described through the distributed system model.

A. User Isolation Requirement

The purpose of the user isolation requirement is to enforce isolation between users.

The requirement has a scope where it applies *i.e.* a level in the hierarchy of the distributed systems. For example, if the requirement's scope is PM, it will apply to all the level below the PM *e.g.* NUMA and socket and at the PM level. We differentiate 3 types of isolation requirement: alone, friends and enemies. With alone requirement, the user specifies that only his VMs can shared the resources at a given level (and all the above ones). With friends one, the user specifies a list of users allowed to share resources with him. And with ennemies, the user specifies a list of users that must never share resources with him.

To verify these requirements, we introduce two functions. The first one checks if a node and the VMs already allocated on it respects the user's requirements. Furthermore, the functions also verifies if all the nodes above the checked node also verifies the user's requirements. The second one verifies if a new VM respects the user's requirements of all VMs already allocated on a given PM.

B. Heuristics

In this section, we will first present two optimizations than can used by all VM placement heuristics. Then, we present heuristics that do not take into account the hierarchy in the section. We present a first fit heuristic that takes into account the hierarchy. Finally, we present best fit heuristics that take into account the hierarchy.

1) Optimizations: A function, *Check Capacity*, is used by the heuristics to verify that a node of the distributed system has enough resources for a VM. To avoid to search in a branch of the hierarchy if no node within it contains enough resources to start a VM, we introduce the optimization $betterFit$. Indeed, a level in the hierarchy can have enough free aggregated resources of a specific capacity but these resources are shared between PMs. For example, a cluster can have 20 free CPU cores but with only a maximal block of 4 available on a PM. Thus, without $betterFit$, the algorithm will try to find a PM that can host the VM that requests 8 CPU cores but will failed as none contains enough resources. With $betterFit$, the algorithm will know that there is no block of CPU cores larger than 4 and thus it can avoid to iterate through all the PMs of the cluster. Therefore, $betterFit$ avoids to search in branch where there is no solution.

Except for the FirstFit heuristics, all the VM placement heuristics presented here return a list of PMs. Thus, we need to sort this list to select the PM that is the best to start the VM. All the PMs returned by the VM placement heuristics fit the capacity and isolation requirements, the sorting algorithm does not change anything about it. But the sorting algorithm helps to improve the quality of placement by selecting the best fitting PMs in a list of fitting PMs. Moreover, the same heuristic can be used to sort all the nodes in the hierarchy and not only the PMs. Indeed, in some heuristics, we need to sort a list of nodes at other level *e.g.* clusters within the hierarchy. We use two different sorting heuristics. $sortBestFit$

applies best fit *i.e.* it places at the beginning of the list the nodes that have just enough resources for the VM. It helps to improve the consolidation of the nodes while respecting the resources and isolation requirements. $sortBestFitAppsAffinity$ is more advanced version that takes into account already placed VMs. In addition of best fit, it puts at the beginning of the list the nodes where VMs from the same application are already running. Then the nodes that has a father, *i.e.* a node at a higher level in the hierarchy, where VMs from the same application are running. And recursively, it does so for every upper levels. For example, it will put at first two PMs where VMs from the same application are already running and will sort one from the other one by applying the best fit heuristic. Then, it will put the PMs that belong to a cluster where VMs from the same application are running and applies best fit heuristic between them. Therefore, it helps to consolidate the application by placing the VMs belonging to an application close to another. Thus, it helps to minimize the latency between VMs of an application. Moreover, by grouping the VMs of an application on a sub-part of the hierarchy, it helps to improve the number of allocated VMs by avoiding to spread an application in the hierarchy. Indeed, if an application is spread in the distributed system and required an isolation requirement with a scope at the upper level of the hierarchy, it can lock a large part of the distributed system for one application.

2) Heuristics not Hierarchy-Aware: In this paper, we present new heuristics that take into account the hierarchy of distributed systems such as cloud. But, we want to compare our heuristics with ones that do not take into account hierarchy. Even if they do not take into account the hierarchy, they respect the capacity and isolation requirements. The $FirstFit$ heuristic takes the list of all the PMs on the distributed system. It iterates trough it until it finds a node that fits the capacity and isolation requirements. The issue with $FirstFit$ is that it always iterates on a fixed list of PMs. To improve it, $FirstFitShuffle$ does a shuffle on the list of PMs before iterating on it. The $BestFit$ heuristic takes the list of all the PMs on the distributed systems. It returns a list that contains all fitting PMs. This list can then be sorted using one of the sorting algorithms presented previously.

3) BestFit: $HierarchicalAwareBestFit$ (HABF) takes into account the hierarchy. It applies one of the two sorting algorithms at each level of the hierarchy. First, it sorts using one of the two sorting algorithms all the root of the distributed systems and selects the best fitting one. Then it does the same with each child of the best fit node. Recursively, it reaches a subset of PMs, *e.g.* the PMs belonging to a cluster that fits the capacity and isolation requirements and returns this list. The list can be then sorted using one of the two sorting algorithms. All the best fit hierarchy-aware heuristics presented here including this one are based on a Breath First Search algorithm.

4) BestFit with Properties: Contrary to the previous one, $HierarchicalAwareBestFitwProperties$ (HABFP) starts to use best fit only when it reaches a level in the hierarchy that is equal to the higher level expressed by the isolation requirements. Thus, it will iterate through all the branches of the tree at the beginning. By doing so, the heuristic is able to choose the best fit node at a specific level in the hierarchy and is not limited to the children of the best fit node of each level. Unless the higher level expressed by isolation requirements is the higher level of the hierarchy, the heuristic will iterate through more nodes but it can find a best fit nodes that is behind a non best fit nodes. For example, an application specifies an isolation requirements at the cluster level. If using the first version of the best fit hierarchy-aware heuristic, it will select a cluster in best fit site. With this heuristic, it will iterate through all the sites and all the clusters belonging to each site. It will then select the best fit cluster.

5) BestFit Large: As we previously state, a best fit node can be behind a not best fit node at a upper level in the hierarchy. Thus, doing best fit at each level of the hierarchy can have bad effects by hiding some nodes. $HierarchicalAwareBestFitLarge$ (HABFL) still applies a best fit hierarchy-aware heuristic but it iterates through all the valid nodes at each level. It will not iterate through part of the system that is not fitting capacity and isolation requirements. Thus, contrary to the non hierarchical-aware heuristics as presented in the Section III-B2, it does not have to iterate through all the PMs.

6) BestFit Large with Hierarchical Sorting: $HierarchicalAwareBestFitLargeSort$ (HABFLS) is a variant of $HierarchicalAwareBestFitLarge$. $HierarchicalAwareBestFitLargeSort$ returns an already sorted list but this list is sorted differently than HABFL. The list is hierarchically sorted such as the first PM on the list are the ones that are the best fitting PMs with the best fitting ancestors. For example, first will be the sorted list of PMs of the best fitting cluster of the best fitting site. Then the best fitting node of the second best fitting cluster of the best fitting site, etc..

7) Summary: We summarize the different heuristics in the table III-B7. For the heuristic complexity (without

optimizations), we use the following formalism. In this case, the complexity is the worst case for finding (or not) a fitting node for a given VM, *i.e.* browsing all the nodes of the platform. A platform is composed of i hierarchical levels, with 1 the lower one, *i.e.* PM and i the higher one, *e.g.* region. At each level, the maximal number of nodes attached to a node of higher level is MN_i. Accordingly, the maximal number of nodes N_i at a level i is $N_i = MN_i \times N_{i-1}$. The global number of nodes at any levels is $N = \forall i, MN_i \times N_{i-1}$. For the sake of simplicity, we use NPM to express the number of PMs. Accordingly, $NPM = MN_O \times N_1$. For example, to model the EC2 platform [2], we use 5 hierarchical levels (region, zone, cluster, rack, PM) with MN_i respectively equals to 7, 3, 5, 4 and 96. In this case, $NPM = 40320$. As one can see in the table III-B7, the theoretical complexity of hierarchy-aware heuristics is a little bit higher than the classical ones, *i.e.* an increasing of complexity of 1.4%. But, we will show in the Section IV that in practice, the hierarchical-aware heuristics are few orders of magnitude faster than classical ones.

8) Inside a Physical Machine: For the moment, the heuristic to select resources inside a PM is limited to select specific CPU core. It can be extended for memory and other resources such as I/O devices. To select a set of cores inside the selected PM, we use the $HierarchicalAwareBestFitwProperties$ heuristic.

IV. Experimentation

In this section, we will first present how we generate simulation scenario then how our simulation scenario are run. Then, we present our simulation results that compare the different heuristics we introduced in the previous section.

A. Scenario

In this section, we describe how we have generate our scenario. These scenario must be plausible with a bit of randomness. The users join and leave the platform based on some basic market rules. At the beginning, the platform has a few early adopters. Then, it gains momentum with an ever increasing number of users joining the platform. Finally, it has reach its limit of attractiveness and few new users are joining it. Accordingly, we approximate the number of users on the infrastructure using a poisson distribution. Furthermore, we use two types of models to simulate the elasticity of applications. The first one is called Uniform, it adds and removes VM (s) at each new application step of the simulation *e.g.* every hours by picking an amount to add/delete randomly. The second one is called AMR, it uses the elasticity model introduced in [35].

We limit our experimentation to applications with only one isolation requirement. It is not a limitation of our algorithms but it is a good first evaluation of them. Indeed, we want to know the impact of isolation requirements on consolidation and on placement time thus combining different requirements will complexify the task. For each application, we randomly select an isolation type (and the related list of friends or ennemies users if needed) and a level at which it applies in the infrastructure.

On top of the isolation requirement, we need for each new application to generate the different parameter of an application (elasticity model, number of VMs at the beginning, isolation requirements). Then, we just need to launch the application *i.e.* schedule the VMs required at the beginning and launch the elasticity function. It runs continuously and based on the elastic model used, adds or removes VMs.

B. Results

We have generate 5 scenarios: 3 (scenario #1), 30 (#2) and 300 (#3) new users per day on a Cloud using the Grid'5000 platform and 3 (#4) and 30 (#5) new users per day on a Cloud using the EC2 platform. We run the 5 scenarios for for each of the 7 placement heuristics and the two options BetterFit (BF) and AppsAffinity (AF). NOBF and NOAF respectively indicat that we do not use BF and AF options.

1) Consolidation: We have compute four indicators for the consolidation: average CPU consolidation, average CPU consolidation per PM, average memory consolidation and average memory consolidation per host. For this metric, the higher is the better. They are computed by taking into account the PMs where VMs are placed. We compute the ratio of the amount of resources used against the total amount of resources on all the PMs for average CPU and memory consolidation. We do the same per PM and compute an average for average CPU and memory consolidation per PM. For reasons of space in this paper, we only display the global memory consolidation in the Figure 2 but the others are showing the same trend. As one can see, the impact of BF and AF optimizations is low especially for hierarchical heuristics. The same is true for all the other metrics. Accordingly, in the following figures, we will not show the value for each combination of optimizations but just the median of all combinations and the error bar.

As shown in the Figure 2, on the Grid'5000 platform, our hierarchical heuristics perform as good as classical

[2]http://aws.amazon.com/ec2/

Short Name	Name	Hierarchicy Aware	Complexity	Complexity EC2
FF	FirstFit	✗	NPM	40320
FFS	FirstFitShuffle	✗	NPM	40320
BF	BestFit	✗	NPM	40320
$HABF$	HierarchicalAwareBestFit	✓	N	40873
$HABFP$	HierarchicalAwareBestFitwProperties	✓	N	40873
$HABFL$	HierarchicalAwareBestFitLarge	✓	N	40873
$HABFLS$	HierarchicalAwareBestFitLargeSort	✓	N	40873

Table I
SUMMARY OF THE DIFFERENT HEURISTICS AND THEIR COMPLEXITY

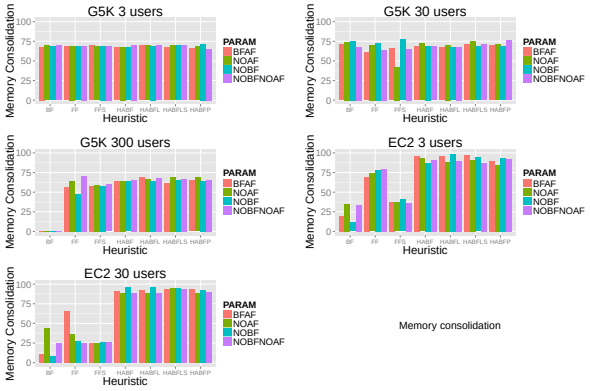

Figure 2. Memory consolidation per Host

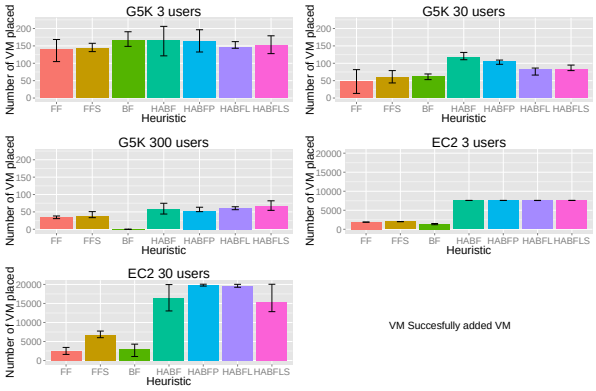

Figure 3. VM successfully added for each scenario

one. Except for the case where 300 new users arrived each day, in this case, the classical BestFit heuristic is not able to run fast enough and the experimentation failed. On the EC2 platform, our hierarchical heuristics perform a lot better than the classical one. Indeed, they are able to take into account the hierarchy of the platform and improve the overall consolidation. Whereas, classical ones only see the platform as a long list of PMs. Accordingly, our heuristics scale better than classical ones on large scale and hierarchical platforms. Futhermore, all our hierarchical heuristics perform equally from a consolidation point of view.

2) Successfully added VMs*:* This metric highlights the number of VMs that have been successfully placed by each heuristics. For this metric, the higher is the better. As shown on the Figure 3, our heuristics perform a little bit better, on average, than classical one on Grid'5000 platforms. But on EC2 platforms, they are performing at least 2 times better than the classical ones. Accordingly, our heuristics permit to place between 2 and 4 times more VMs on the same platform than the classical ones. This is strongly link with the consolidation metrics as placing more VMs on a the same platform required to improve a better consolidation.

3) VM *placement duration:* In this section, we study the impact of the different heuristics and options on the duration of the VM placement. The lower is the better. The duration on the Figure 4 is displayed in nanoseconds. Except for the first experiment (3 users per day on the Grid'5000 platform), the value for the classical BestFit heuristic is not displayed because it is at least 100 times higher than the worst performing one and renders the figure unreadable if plotted. Contrary to the consolidation and successfully added VMs metrics, our hierarchical heuristics perform better even on the Grid'5000 platform. For example on the fifth experiment, our heuristics perform between 1,000 and 10,000 faster than the two classical FirstFit heuristics. Consequently, our heuristics are able to have at least the same consolidation and successfully added VMs than classical ones and being faster by at least two orders of magnitude. Finally, one can see than HABFP and HABFLS are slower than the two other ones. This is due to a larger space search in average. Indeed as previously explained, these two heuristics start to cut branch in the search space later than the two other.

4) Impact of requirements on placement duration: This section concludes the study of our experimentation.

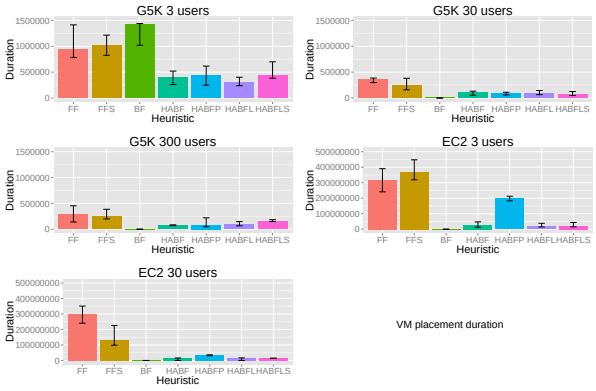

Figure 4. Average duration of placing a new VM

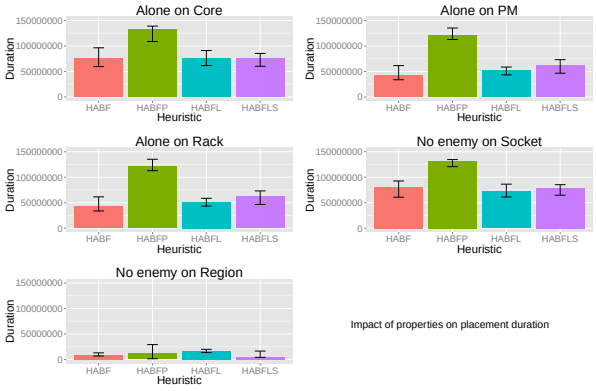

Figure 5. Average duration of placing a new VM with different level and type of isolation

Here, we take a look on the impact of isolation requirements on the placement duration. We only study it for our heuristics as the other ones does not intrinsically take into account the isolation requirements. Futhermore, for a reason of space, we only show the results for a small subset of possible isolation requirements and for only one experiment (EC2 with 30 new users per day). We wanted to know if requiring a low (core) or high (datacenter) level of isolation or a different type of isolation (alone, friend, enemy) will have impact on the placement duration. As one can see on the Figure 5, the type of isolation is not strongly related to the placement duration. On the contrary, the level of isolation has an impact on the placement duration. The lower the level of isolation is the slower the placement duration time is. Indeed, with a high level of isolation, the heuristics are able to reduce the search space earlier than for the requirements that require low level of isolation. Finally, as in the previous section, HAFP and HABFLS are slower than the two other ones.

V. Conclusion

We have seen that a large range of possible and real-world attacks on Cloud platforms are due to improper isolation. We have shown that the existing method to improve isolation (from a data and performance points of view) are limited to only detecting them or induce a large overhead. Accordingly, we propose to enhance VM placement heuristics with isolation requirements.

We have introduce the concept of user's isolation requirements for hierarchical and distributed large scale platform. Then, we have explained how these requirements can be used inside VM placement heuristics. Furthermore, we have introduce 4 new heuristics that use the hierarchical nature of the platform.

From experiments, we show that our heuristics perform at least as good as classical ones. Furthermore, on large-scale platforms such as EC2, our heuristics are performing few orders of magnitude better than the classical ones. Finally, we highlight the correlation between VM placement duration and the level of isolation required by a VM.

In the future, we plan to take into account the possible correlation between consolidation and the amount of successfully placed VMs. We work on a improved application model with a larger scale of security requirements. We plan to extend our heuristics to support the whole scope of the requirements and applications expressed through it. Finally, we will extend our heuristics to allocate network resources as well.

References

[1] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09. New York, NY, USA: ACM, 2009, pp. 199–212.

[2] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, feb. 2010, pp. 180 –186.

[3] B. Goglin and S. Moreaud, "Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters," in *CASS 2011: The 1st Workshop on Communication Architecture for Scalable Systems, held in conjunction with IPDPS 2011*. Anchorage, AK: IEEE Computer Society Press, May 2011.

[4] Q. Ali, V. Kiriansky, J. Simons, and P. Zaroo, "Performance Evaluation of HPC Benchmarks on VMware's ESXi Server," in *Proceedings of the 2011 international conference on Parallel Processing*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 213–222.

[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 *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, ser. CCSW '11. New York, NY, USA: ACM, 2011, pp. 29–40.

[6] K. Okamura and Y. Oyama, "Load-Based Covert Channels Between Xen Virtual Machines," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC '10. New York, NY, USA: ACM, 2010, pp. 173–180.

[7] Y. Zhang, A. Juels, A. Oprea, and M. Reiter, "HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis," in *Security and Privacy (SP), 2011 IEEE Symposium on*, may 2011, pp. 313 –328.

[8] Z. Wu, Z. Xu, and H. Wang, "Whispers in the Hyper-space: High-speed Covert Channel Attacks in the Cloud," in *the 21st USENIX Security Symposium (Security'12)*, August 2012.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[10] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, "Resource-Freeing Attacks: Improve your Cloud Performance (At your Neighbor's Expense)," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 281–292.

[11] T. Moscibroda and O. Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 18:1–18:18.

[12] S. K. Barker and P. Shenoy, "Empirical Evaluation of Latency-Sensitive Application Performance in the Cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, ser. MMSys '10. New York, NY, USA: ACM, 2010, pp. 35–46.

[13] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, july 2010, pp. 51 –58.

[14] G. Wang and T. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –9.

[15] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 460–471, Sep. 2010.

[16] S. Bleikertz and T. Groß, "VALID: A Virtualization Assurance Language for Isolation and Deployment," *IEEE Policy*, 2011.

[17] H. Raj, R. Nathuji, and A. Singh, "Resource management for isolation enhanced cloud services," *CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security*, p. 77, 2009.

[18] K. Taesoo, M. Peinado, and G. Mainar-Ruiz, "System-Level Protection Against Cache-based Side Channel Attacks in the Cloud," in *Proceedings of the 21st Usenix Security Symposium*, ser. USENIX Security'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 1–16.

[19] A. Gulati, A. Merchant, and P. J. Varman, "mClock: Handling Throughput Variability for Hypervisor IO Scheduling," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–7.

[20] B. Verghese, A. Gupta, and M. Rosenblum, "Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors," *SIGOPS Oper. Syst. Rev.*, vol. 32, no. 5, pp. 181–192, Oct. 1998.

[21] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance Isolation for Cloud Datacenter Networks," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–1.

[22] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized Cloud Infrastructure without the Virtualization," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 350–361, Jun. 2010.

[23] J. Szefer, E. Keller, and R. Lee, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in *ACM Conference on Computer and Communications Security*, 2011.

[24] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud, "Autonomic Virtual Resource Management for Service Hosting Platforms," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8.

[25] E. Feller, L. Rilling, C. Morin, R. Lottiaux, and D. Leprince, "Snooze: A Scalable, Fault-Tolerant and Distributed Consolidation Manager for Large-Scale Clusters," in *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, dec. 2010, pp. 125 –132.

[26] E. Feller, L. Rilling, and C. Morin, "Energy-Aware Ant Colony Based Workload Placement in Clouds," in *The 12th IEEE/ACM International Conference on Grid Computing (GRID-2011)*, Lyon, France, Sep. 2011. [Online]. Available: http://hal.inria.fr/inria-00626042/en/

[27] J. L. L. Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments," in *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, july 2011, pp. 1 –7.

[28] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud Brokering Mechanisms for Optimized Placement of Virtual Machines Across Multiple Providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358 – 367, 2012.

[29] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-aware Virtual Machine Placement," in *Services Computing (SCC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 72–79.

[30] M. Bhadauria and S. A. McKee, "An Approach to Resource-Aware Co-Scheduling for CMPs," in *Proceedings of the 24th ACM International Conference on Supercomputing*, ser. ICS '10. New York, NY, USA: ACM, 2010, pp. 189–199.

[31] A. Merkel, J. Stoess, and F. Bellosa, "Resource-Conscious Scheduling for Energy Efficiency on Multicore Processors," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 153–166.

[32] S. Zhuravlev, S. Blagodurov, and A. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 129–142, Mar. 2010.

[33] J. Li, M. Qiu, J. Niu, W. Gao, Z. Zong, and X. Qin, "Feedback Dynamic Algorithms for Preemptable Job Scheduling in Cloud Systems," in *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, ser. WI-IAT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 561–564.

[34] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy Aware Consolidation for Cloud Computing," in *Proceedings of the 2008 conference on Power aware computing and systems*, ser. HotPower'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 10–10.

[35] C. Klein and C. Perez, "An RMS for Non-predictably Evolving Applications," *Cluster Computing, IEEE International Conference on*, vol. 0, pp. 326–334, 2011.