Secure Allocation for Graph-Based Virtual Machines in Cloud Environments

1st Mansour Aldawood
Department of Computer Science
University of Warwick
Coventry, UK
m.aldawood@warwick.ac.uk

2nd Arshad Jhumka
Department of Computer Science
University of Warwick
Coventry, UK
h.a.jhumka@warwick.ac.uk

***Abstract*—Cloud computing systems (CCSs) enable the sharing of physical computing resources through virtualisation, where a group of virtual machines (VMs) can share the same physical resources of a given machine. However, this sharing can lead to a so-called side-channel attack (SCA), widely recognised as a potential threat to CCSs. Specifically, malicious VMs can capture information from (target) VMs, i.e., those with sensitive information, by merely co-located with them on the same physical machine. As such, a VM allocation algorithm needs to be cognizant of this issue and attempts to allocate the malicious and target VMs onto different machines, i.e., the allocation algorithm needs to be security-aware. This paper investigates the allocation patterns of VM allocation algorithms that are more likely to lead to a secure allocation. A driving objective is to reduce the number of VM migrations during allocation. We also propose a graph-based secure VMs allocation algorithm (GbSRS) to minimise SCA threats. Our results show that algorithms following a stacking-based behaviour are more likely to produce secure VMs allocation than those following spreading or random behaviours.**

***Index Terms*—Cloud computing, side-channel attacks, virtual machine allocation**

I. Introduction

The paradigm of cloud computing systems (CCSs) focuses on sharing the physical computing resources, e.g., CPU and RAM, among cloud users through virtualization technology. Virtualization allows cloud users to access the computing resources hosted on physical machines (PMs) through virtual machines (VMs) [1]. As a result, sharing physical computing resources leads to security threats among co-located VMs in CCSs [2].

In particular, we consider the side-channel attacks (SCAs), which can occur when a malicious VM shares the same PM with a target VM, a VM that contains sensitive information. The SCAs depends on collecting related information about the co-located VMs, such as execution time through a cache-based attack, then analysing the collected bits of information for profiling and attacking the target VMs. Subsequently, the impact of SCAs expands from software to hardware level. [3]. As such, and to overcome the SCAs, we focus on developing an algorithm that allocates the malicious and targets VMs on distinct PMs, i.e., the allocation algorithms avoid malicious co-location. Therefore, in this paper, we focus on producing a secure VMs allocation that avoids malicious co-residency.

In our previous work [4], we address the issue of identifying the VMs with malicious or normal behaviour; thus, isolating them from other VMs. Moreover, we assume that the cloud service provider (CSP) can analyse the behaviours of the VMs using a learning model and categorise them into three predefined classes, malicious, target and normal VMs. Then, allocate the VMs based on the outcome of the classification from the learning model. Classifying the VMs depends on detecting the malicious behaviour of cloud users by analysing the abnormal activities of their VMs, or other computing components that they utilised [5].

The main contributions of this paper are as follows: (i) we consider several VM allocation algorithms and investigate their ability to return a secure allocation potentially. The three classes of algorithms we study are (a) spreading, (b) stacking and (c) random VMs allocation behaviours. The spreading behaviour will attempt to spread the VMs across all the available PMs, e.g., round-robin fashion. On the other hand, the stacking will attempt to stack VMs on PMs, possibly resulting in fewer PMs being used, e.g., a Bin-Packing algorithm. Finally, the random allocation behaviour allocates the VMs to PMs randomly, as long as the resources are available at that PM. Moreover, (ii) we propose a secure graph-based stacking algorithm (GbSRS) to obtain a secure VM allocation in the cloud system. GbSRS follows a stacking-based behaviour, similar to bin-packing; however, bin-baking is not classified as a secure-aware algorithm which requires adaptation to be secure [6]. Our algorithm considers the outcome of the VMs behaviour analysis and the load correlation of the resources during the allocation. Additionally, (iii) we study the impact of several factors on the quality of the VM allocation, such as VMs arrival time, the proportion of specific VM types and the PMs resources structure (i.e., how their available resources vary during allocation). We simulate our allocation algorithms extensively under several suitable scenarios. Our results show that our proposed algorithm outperforms state-of-the-art VM allocation algorithms in terms of security.

The paper is structured as follows: We present a survey of the literature and related work in Section II. We present our system model and a formalisation of the problem we focus on in Section III. We develop our algorithms in Section IV, and

we present their results in Section V. We conclude the paper in Section VI.

II. Related Work

The paper aims to develop a secure VM allocation algorithm in CCSs to defend against SCAs by mitigating the chance of malicious co-residency. This paper is based on our previous work [4], where we studied the problem of the VMs allocation when they are independent of each other. However, in this paper, we will consider the dependent VMs, represented as graphs, to examine the effect of secure allocation on the graph-based allocation.

The areas that tackled SCAs focused on either finding a solution logically on the VMs level or physically on the hardware level. As such, some areas focus on VMs clustering during the allocation by grouping the VMs based on specific predefined criteria. For example, in [7], a group-based allocation policy was proposed to create a balance between optimizing the resources and obtaining a secure VM allocation. Moreover, in [8], they group the VMs based on risk factors obtained from network connectivity between the VMs or the PMs hosting them. While in [9], they group the VMs and allocate them based on conflict-free groups between VMs. In other words, a VM has a minimum conflict with the other VMs in which they share the same PM. Similarly, some solutions aim to allocate the VMs based on predefined security standards or controls [10]. Further, in some cases, based on a user security profile defined by the user [6]. For instance, [11] proposed a method that places the VMs allocation process mainly by maintaining the same security standard level of the shared VMs. Furthermore, the work in [12] aims to migrate the VMs while ensuring that the VMs hosted on the same PM share the same security level.

On other domains, [13] and [14] focus on finding an allocation, or migration, for the VMs to limit the time of VMs co-location. However, this requires the malicious VM to obtain a co-location with the target VM in a considerable amount of time. Thus, these algorithms try to minimise this time of VMs sharing the same PM, and therefore, migrate the VMs that trigger the time limit.

In other areas, in [15] and [16], they focused on modifying the hardware structure of the CCSs to minimise the leakage information through the side channel. Finally, some researches focus on developing an algorithm that manipulates the scheduling component of the cloud system. For example, in [17], an algorithm deliberately delays the VMs arrival to a specific time to reduce the chance of being allocated with malicious VMs. While in [18], their algorithm tries to migrate the VM frequently to keep tracking the target VMs difficult to the malicious VM. Furthermore, some considered factors that affect the VM allocation, such as VMs number [19]. Alternatively, some consider an optimisation-driven solution to tackle this issue and obtain a secure allocation [20].

III. Virtual Machine Allocation Model

In this section, we present the VMs allocation model that we assume in this paper. The model includes the PMs architecture and VMs relation, which we view as a graph-based architecture. First, the PMs architecture, or data centre topology, is modelled as a Fat-Tree architecture to represent the relation between VMs, PMs and network components [21]. Second, unlike our previous work in [4], in this paper represent the VMs relation as a weighted undirected graph, representing the relationships and interactions between VMs.

A. System (Physical) Model

We assume that the CCS is structured as a Fat-Tree architecture, with three layers of switches: a core switch S_c, a set of aggregation switches S_a and a set of edge switches S_e [22]. The edge switch $s \in S_e$ connects the PMs directly and also connects to upper linked switches S_a. An aggregation switch $s \in S_a$ distributes the communication links from uplinks and downlinks, while the core switch S_c connects the CCS to the outside world. The switches are the inside vertices of the tree, while the PMs are the tree's leaves.

Furthermore, the CCS consists of a set P of $(k+1)$ PMs, and each $PM_i \in P, 1 \leq i \leq k$ has the same set of resources, but in varying quantities, e.g., one PM may have more CPU cores or more memory size than another, i.e., we assume the system to be heterogeneous. We denote by $R(PM_i)$, the amount of physical resources available on machine PM_i. We assume a special PM, denoted by PM_0, upon which unallocated VMs reside. Additionally, there is a non-empty set V of VMs, and each $VM^j \in V$ may have a different set of resource type requirements, i.e., we assume the VMs to be heterogeneous, as for the PMs.

In this paper, we assume that all the resource needs of a $VM^j \in V$ can be met by any PM in the CCS. We denote by $N(VM^j)$, the amount of resources needed by VM^j. We also assume that $\forall\ 1 \leq i, j \leq k, i \neq j, PM_i$ is able to communicate with PM_j in the CCS through the connected switches, i.e., the CCS is connected.

B. VMs Allocation Model

The VMs allocation model we assume in this paper is that of a weighted undirected graph $G(V, E, L)$, where the V is the set of VMs, the set E of edges represents the relation between two VMs (VM^i, VM^j) and L is an edge labelling function that returns the label on an edge $e \in E$. We call such a model the VMs load correlation model (see Figure 1(C)). Such a model exists for each of the three types of VMs we assume, namely normal (set N), target (set T) and malicious (set M), under the following constraints:

- $V = T \cup M \cup N$
- $T \cap M = \emptyset \wedge T \cap N = \emptyset \wedge M \cap N = \emptyset$

1) VMs Type: Based on the learning model assumed in our previous paper [4], a VM will be classified as either being a target, malicious or a normal VM. Starting with the set of VMs (see Figure 1 (A)), we initially assume that all VMs of the same type form a fully connected graph, called a VMs Type graph, in that all VMs of the same type can potentially communicate with each other (see Figure 1 (B)).

2) VMs Load Similarity: As a system administrator may wish for related VMs to be co-located, we propose a VMs load similarity metric to capture the similarity of two VMs in terms of resource requirements. A VM Type graph is thus refined into a VMs load similarity graph by labelling the edge between VMs by the similarity metric between the two VMs (see Figure 1 (C)) and only keeping the edge with the highest value for each vertex.

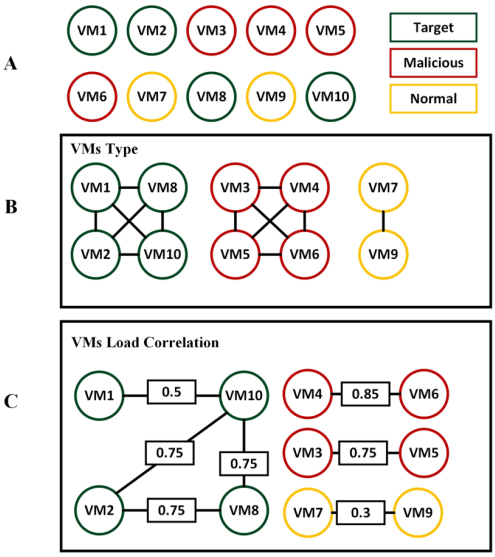

Fig. 1. The Model for VMs Type and Load Correlation.

The load similarity of two VMs, VM^i and VM^j, denoted by vm_λ is computed as follows:

$$vm_\lambda(VM^i, VM^j) = \frac{N(VM^i)}{N(VM^j)} \tag{1}$$

The VMs Type graph $G_{\lambda,\lambda\in\{T,M,N\}} = (V, E)$ is converted into a VM load similarity graph $G'_{\lambda,\lambda\in\{T,M,N\}} = (V', E', L)$ as follows:

- $V = V'$
- $L(u, v) = vm_\lambda(u, v)$
- $\forall(u, v), (u, v') \in E, v \neq v' \cdot (u, v) \in E' \Leftrightarrow L(u, v) \geq L(u, v')$

As such, VMs with high load similarity may be co-located under appropriate resource availability. Moreover, we model the VM allocation and VM migrations similarly to our previous work but with adding the changes related to the graph-based architecture. Furthermore, we consider the allocation A to be *secure* if $\forall m \in M, \forall t \in T : A(m) \neq A(t) \neq PM_0$, i.e., an allocation is secure if no malicious VM is co-located with a target VM. Similarity, the VMs migration is secure if both the start and the end allocations are secure. Consequently, we need to reduce the cost of a VM migration. As such, we define the $Cost_\Delta$ of VM migration $\Delta = Move(A_i, A_{i+1})$ as follows: Denoting by $MP(PM_i, PM_j)$, the shortest path between PM_i and PM_j in the CSS graph.

$$Cost_\Delta = \sum_{\forall v \in Move(A_i, A_{i+1}), A_i(v)=s \wedge A_{i+1}(v)=e} cost(MP(s, e)) \tag{2}$$

where $cost(MP(s, e))$ is the sum of delay of each switch and link on that path $MP(s, e)$.

IV. Algorithm for Secure Allocation

This paper aims to develop a secure allocation algorithm while minimising the cost of VMs migrations. Thus, we propose our graph-based security-aware heuristic, called Graph-based Secure Random Stacking (GbSRS), shown in Algorithm 1, as well as a migration algorithm called Graph-based VM Migration (GbM), to migrate the VMs securely while reducing the cost of VM migration. GbSRS is an extension of our previous algorithm called secure random stacking (SRS), where both algorithms follow the same process of producing as many secure allocations as possible; however, they have different allocations steps [4]. Thus, in this paper, we will only be explaining the changes from our previous work, which allocates the dependant VMs based on their graph-based connectivities.

A. Graph-based Secure Random Stacking (GbSRS)

The input of the GbSRS algorithm is the unallocated set of VMs, represented in a graph g, and the output is the secure allocation produced under the available resources, denoted as A. The GbSRS performs three attempts to allocate a given VM, vm_i. The first attempt, in line 1, will try to allocate the vm_i on the same PM, pm_j, of the connected VM from the load similarity graph. Each VM is connected initially with another VM by how similar they are, based on their required resources. The function $getConnectedVM(vm_i)$, in line 2, will be triggered to retrieve the VM with the highest load similarity of the connected VMs.

The second attempt, in line 9, aims to allocate the unallocated VM to one of the PMs connected to the edge switch of the connected VMs. The function $getAllocatedConnectedVM(vm_i)$, in line 10, is triggered to return all the connected VMs, from the VMs type graph, that already allocated. This step aims to know the edge switch linked to the connected VMs, to retrieve all PMs connected to this edge switch. As such, in line 12, the function $getEdgeSwitch()$ will be triggered to return the edge switch of the connected VM. From this information, in line 13, the list of PMs linked to the selected edge switch will be retrieved and therefore considered for an allocation.

Afterwards, the list of PMs is sent a $getHighestFRPMs$ function to select the most two highest fullness ratio PMs only out of the available PMs. And the outcome is then the stored list of elected PMs, denoted as $ElectedPMsList$, as explained previously in [4]. Then, in line 16, the algorithm randomly selected the PMs among the elected PMs and assigned them as candidates for allocation. The function $getRandomPM(ElectedPMsList)$ is responsible for selecting a random PM among the list of the elected PMs and store as a candidate PM. Subsequently, if the candidate PM is suitable,

Algorithm 1 Graph-based Secure Random Stacking (GbSRS).

Input: $g = G(V, E, L)$
Output: A

1: **for** $vm_i \in g$ **do**
2: $vm_\lambda \leftarrow getConnectedVM(vm_i)$ {First Attempt}
3: **if** $vm_\lambda.getPM() \neq null$ **then**
4: $pm_j \leftarrow vm_\lambda.getPM()$
5: **if** $(pm_j.suitablePM(vm_i))$ **then**
6: $A \leftarrow Assign(vm_i, pm_j)$
7: **end if**
8: **end if** {Second Attempt}
9: **if** $A = null$ **then**
10: $vm_\tau \leftarrow getAllocatedConnectedVM(vm_i)$
11: **if** $vm_\tau \neq null$ **then**
12: $edge_i \leftarrow vm_\tau.getEdgeSwitch()$
13: $pms \leftarrow edge_i.getPMsList()$
14: $ElectedPMsList.add(getHighestFRPMs(vm_i, pms)$
15: **for** $Counter < ElectedPMsList.size()$ **do**
16: $pm_i \leftarrow getRandomPM(ElectedPMsList)$
17: **if** $(pm_i.suitablePM(vm_i))$ **then**
18: $A \leftarrow Assign(vm_i, pm_i)$
19: **end if**
20: **end for**
21: **end if**
22: **end if** {Third Attempt}
23: **if** $A = null$ **then**
24: Third attempt will be on aggregation switch
25: Last attempt steps is similar to GbSS algorithm
26: **end if**
27: **end for**
28: **return** A

it will be considered a host for the vm_i and added to the A, in line **??**.

The third attempt will repeat the same process as the second attempt but with PMs linked to the same aggregation switch. This step will increase the number of available PMs to be selected for an allocation. If these attempts failed to obtain a secure VM allocation, then the VM migration, will be triggered in order to produce another allocation. Finally, if all the above attempts failed, including the VMs migration, the GbSRS will try to allocate the vm_i on any PMs in the P regardless of graph connection constraints.

In this work, we consider the resources of the VMs and PMs are multidimensional, meaning each one of them has a set of resources representing them. For example, the resources are RAM, CPUs with their cores, storage, network switches and other resources are represented as. Thus, we calculated the multidimensional resources that performed in the FR function as follow:

$$FR = \frac{VM_{ram}}{PM_{ram}} + \frac{VM_{cpu}}{PM_{cpu}} \tag{3}$$

In Eq (3), we compare the upcoming VM with each PMs, considering the RAM and CPU specifications. If the result of the computation is equal to two, then the PM is a perfect match for the VM and this PM will be the first one selected for possible allocation.

B. Graph-based VM Migration (GbM)

Compare to the migration algorithm presented in our previous work [4], the GbM aims to minimize the number of VM migrations, hence the migration cost. As such, the algorithm finds a suitable PM for the migrated VMs randomly only if the $Cost$ of migration to the selected PM is less than Δ. We define the cost Δ in Eq 2 as the cost of VM migration from one PM to another, and where it should not exceed the predefined threshold.

V. Results and Discussion

This section will evaluate the proposed algorithm and compare it with existing ones, following a different VMs allocation behaviour. As stated earlier, our proposed algorithm follows a stacking-based behaviour; therefore, we compare it with state-of-the-art algorithms that follow a spread and random behaviour.

Moreover, We utilise a simulation environment called a Network CloudSim, an open-source cloud simulation environment specified for network-related simulations in the cloud environment [23]. Further, we utilised the Azure VMs traces published by the Microsoft team, which contains the VMs workload of Azure [24]

Furthermore, we study the effect of different VM allocation behaviours on obtaining a secure VMs allocation under certain situations. These situations are the arrival of specific VMs impacts on the secure allocation. And, the amount of VMs type impacts or whether the structure of PMs (heterogeneity) resources and how they differ from each other can affect the secure allocation.

Additionally, following our previous work [4], we compared our algorithm behaviour, stacking-based behaviour, against allocation algorithms behaviours such as spreading and random, explained in [25] and [26], respectively. This section will refer to the spreading behaviour as Round Robin (RR), while the random behaviour as (Rand), for simplicity. Furthermore, we added a unique behaviour that combines the spreading and random behaviours, called Previously Selected Servers First (PSSF), explained in [27].

A. Simulation Setup

We design the cloud data centre of this experiment as a Fat-Tree architecture with three levels, as in [22]. The root level, which is the core switch, is linked to a sublevel of aggregation switches. These aggregation switches are linked to a sublevel of edge switches. In which these edge switches are linked to the PMs. Finally, the PMs are hosting the VMs based on the allocation Algorithm behaviour. The following sections will explain the setup in more detail.

TABLE I
TYPE OF VMS ARRIVALS

Type	*Order Detail*
MTN	S(MTN)-G(M)-S(MTN)-G(T)-S(MTN)-G(N)-S(MTN)
NMT	S(NMT)-G(N)-S(NMT)-G(M)-S(NMT)-G(T)-S(NMT)
TNM	S(TNM)-G(T)-S(TNM)-G(N)-S(TNM)-G(M)-S(TNM)

1) Arrival of VMs: We consider three VMs arrival times, i.e., when the VMs are arrived and allocated on a PM. The goal is to show the effect of VMs arrival time, based on its type, on the overall malicious co-residency for each algorithm class. As described in Table I, the three arrival times are MTN, NMT and TNM. MTN captures the case where the malicious VMs (M) arrived first, then the target VMs (T), and finally the normal VMs (N). The same concept applies to the other two types.

Furthermore, the order detail in Table I, shows how the VMs exactly are arrived in each experiment. To demonstrate, in the MTN arrival type, the VMs in any experiment is divided into seven groups; each group will arrive in the same sequence described in the table under the order detail column. For example, out of seven groups, the first group is S(MTM), where the S refer to a Single VM arrival. Here, the malicious, target and normal VM will arrive in a single alternative sequence. Precisely, single (M)- then single (T)- then single(N), then the process is repeated until this group of VMs arrived. Then the second group, G(M), means that a group of malicious VMs will arrive second. Then the third group will repeat the same order as the first group. After that, the fourth group G(T), means that a group of target VMs will arrive fourth. Then the rest of the seven groups will arrive until the last VM arrives. The same concept will apply to the other two types of VM orders, NMT and TNM. The motivation for configuring the VMs arrival in such order, specificity, starting with single VMs followed by a group of VMs instead of the opposite, makes the allocation more challenging for the algorithms. Moreover, to mimic the real-world scenario of VMs arrival as much as possible.

2) Switches and PMs Structure: In this paper, we structured the linkage between PMs and switches: A maximum of four PMs are connected to each edge switch. Each aggregation switch is connected by a maximum of two edges, which means that each aggregation switch can be connected by eight PMs maximum. Finally, all the aggregation switches are connected to one core switch. The number of PMs in each experiment will accommodate the required resources of the VMs. Moreover, we designed the available resources to be limited compare to the demanded resources. Hence, it makes it challenging for the algorithms to find a secure allocation. Additionally, as we presented in our earlier work [4], we consider three types of PMs structure, or level of PMs heterogeneity, High, Medium and Low heterogeneous PMs, which indicates how much the PMs are different from each other concerning the available resources.

3) VMs Structure: As stated earlier, we have utilised the VMs traces of the Azure cloud data centre; however, we only use one VM from each user id, resulting in 6687 VMs for each experiment. This step aims to produce a different set of demanded resources for each arriving VM, i.e., heterogeneous VMs. Also, it makes the allocation of the VMs more challenging to the proposed algorithms. After collecting the data set from Azure traces, we then utilise the VM information and create the VMs on Network CloudSim simulation. Then, we arrange the VMs according to their created time and assign a type for each VM, either T, M or N. Moreover, we consider the possibility of each VM type number in each experiment. In other words, how many target, malicious or normal VMs in each experiment. For example, from 1 to 7 in Figure 2, the experiments have a different VMs structure based on VM type. In the first experiment, the number of target VMs is higher than malicious and normal VMs, while in the second experiment, the number of normal VMs is high than target VMs, and normal VMs. Until reaching the seven experiments, which an indication of the end of VMs type number possibilities examine. Afterwards, from 8 to 14, the same possibilities of VMs type number are repeated. Consequently, a different VM structure based on VM type is considered to identify whether a specific amount of VM type affects the overall secure allocation.

B. Result of malicious co-residency

In the Figures 2 to 4, we compare our algorithm, GbSRS, that follows a stacking-based behaviour with RR, Rand and PSSF. The objective is to calculate the percentage of PMs with malicious co-residency, denoted as (M_{pms}), as explain in [4].

In general, the algorithms GbSRS and Rand perform better than the PSSF and RR in most situations. This initial observation is an indication that the spreading behaviour allocation often leads to higher levels of malicious co-residency than the stacking or the random behaviour. Arguably, the Rand algorithm will always produce a different allocation in each simulation run due to its random behaviour. However, the M_{pms} are lower than anticipated in most situations, which is good for this behaviour. The possible reason for this outcome is the data centre design for the experiment and how the VMs are allocated. It influences the Rand algorithm to behave partially in a stacking-based manner while randomly selecting from the PMs available in each aggregation switch.

Moreover, the M_{pms} of the spreading behaviour algorithms, RR and PSSF, are often higher, because the VMs spreading across the entire available PMs, and the chance for a malicious user to obtain a malicious co-residency is higher. The more the malicious user has many VMs, the more the chance of malicious co-residency occurs. In other cases, the amount and spreading of target VMs help increase the M_{pms}, even if the malicious user has a fewer number of available VMs.

In addition, our proposed algorithm is performing as well as the Rand algorithm. Definitely, with more available PMs, the M_{pms} will be lower, and while with limited available PMs, the M_{pms} could perform worse in some situations. However, the lower M_{pms} is due to the stacking behaviour conducted during

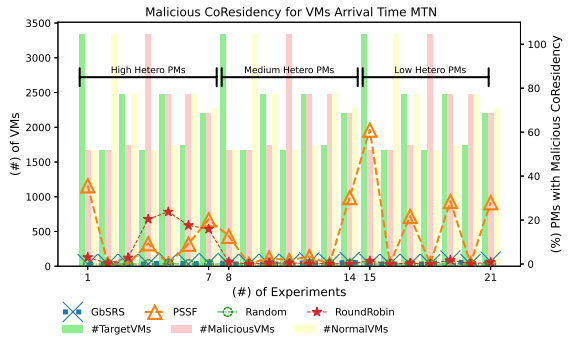

Fig. 2. M_{pms} for VMs Arrival Time MTN

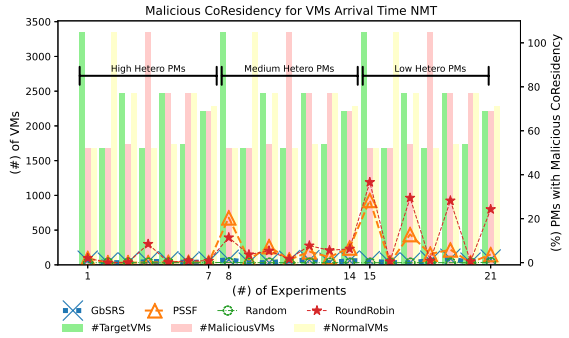

Fig. 3. M_{pms} for VMs Arrival Time NMT

the allocation. As such, the algorithm's behaviour leads to stack the connected VMs as much as possible while leaving available PMs freely for future allocation. Even if the malicious user can increase the VMs amount, it will be challenging to achieve malicious co-residency. Furthermore, in RR and PSSF, the spike of the amount malicious or target VMs could potentially lead to higher M_{pms}, in some cases, the spike of both together. Because it occupied the available PMs, depending on the time of VMs arrival, that potentially needed for obtaining a secure allocation.

Mostly, the effect of PMs heterogeneity structure on the M_{pms} is higher for all algorithms when it is low, i.e., the set of available resources for the PMs are the same. Thus, it

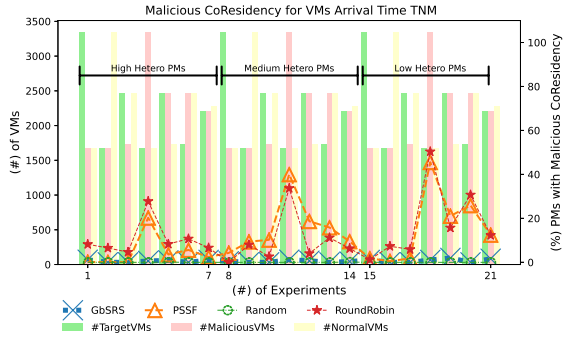

Fig. 4. M_{pms} for VMs Arrival Time TNM

is an indication that to obtain a relatively secure allocation, it is favourable to avoid allocating the VMs on the PMs with the same structure. On the other hand, the medium and high structure generally yields lower M_{pms} for some of the algorithms. A possible reason for having either high or low M_{pms} on the three PM structure is that each algorithm fits a VM into a PM. Unfitted (VM to PM) allocation may skip a perfect allocation due to spreading or random behaviour of the algorithm, which leads to an increase in the M_{pms}. Our algorithm GbSRS uses a fullness ratio function that calculates the available space of PM compares to the demanded resources from the VM. Then selected the PMs that mostly fitted to a particular VM. Thus, it is a calculated relation between each VM with the available PMs.

Overall, in RR and PSSF, the M_{pms} is relatively higher when the majority of either the malicious or target VMs arrive firstly in MTN and TNM, respectively. The reason is that the VMs, for example, the malicious ones, are initially spread across the available PM. Then, after some time, when the target VMs arrive, fewer spaces will be available for secure allocation. While our algorithm, which will stack the VMs to allow more spaces for future VMs, thus, reducing the M_{pms}. In addition, the M_{pms} for PSSF is worse when the majority of target VMs arrives after the malicious ones. On the other hand, in Figure 3, when the majority of malicious and target VMs arrives last, the M_{pms} is lower than other arrivals. This situation refers to the fact that any target or malicious VMs can be allocated with normal VMs. The migration process could potentially affect the M_{pms} for this situation. Because most normal VMs arrive first, which gives the algorithms a possibility for VM migration, a free space is available for either a malicious VM or a target, which result in reducing the chance of M_{pms}.

C. Result of VMs Migration

In Figure 5, we study the effect of VMs migration of the allocation algorithms, denoted as Mig_{vms} [4]. The VMs migration indicates the cost of VMs migration for each algorithm; the $Cost$ of VMs migration is explained in Eq (2). In other words, the higher percentage of Mig_{vms} leads to a higher cost of VMs migration.

Overall, RR, PSSF, and Rand algorithms are having a higher Mig_{vms} compares to our proposed algorithm. The migration Mig_{vms} reflect the cost percentage for each algorithm, where it is clear that allocating the VMs in a stacking-based manner leads to a lower cost of VMs migration. Therefore, GbSRS performs better than other algorithms due to allocating the VMs using the fullness ratio function. This stacking-based behaviour produces a fitted allocation for each VM, thus minimising the need for migration and VMs migration cost. On the other hand, PSSF and RR algorithms show a similar behaviour because they share similar allocation behaviour, spreading the VMs. This leading to a significant amount of VMs migration, hence, a high cost of migrating the VMs. In contrast to the Rand, it shows an irregular pattern of Mig_{vms} for the situation due to its randomness behaviour. Nevertheless, it has a high cost for

VMs migration in most cases examined, similar to RR and PSSF.

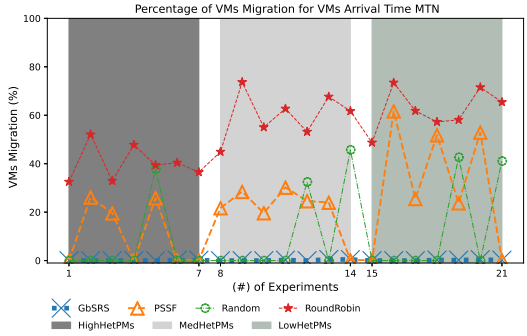

Fig. 5. Mig_{vms} for VMs Arrival Time MTN

VI. Conclusion

This paper developed a model and algorithms that consider the dependent VMs represented as graphs to examine their effect on obtaining a secure VMs allocation on a graph-based architecture data centre. We consider the load similarity of the VMs and the type classification of the dependent VMs during the allocation. Furthermore, we evaluated the graph-based VMs allocation behaviours on obtaining a secure allocation and investigated some factors that affect producing malicious co-residency. Our results show that the data centre topology impacted the overall outcomes of malicious co-residency and the VM arrival times have also a significant impact on obtaining a secure allocation. In future work, we will be extending the study related to the proposed learning model framework that classifies the VMs based on their activities into specific types and allocates them subsequently. With the advancement of machine learning tools and detailed datasets related to cloud activities, it becomes possible to formulate and develop a model for secure VMs allocation accordingly.

References

[1] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research*, 2009, pp. 101–111.

[2] W. A. Jansen, "Cloud hooks: Security and privacy issues in cloud computing," in *2011 44th Hawaii International Conference on System Sciences*. IEEE, 2011, pp. 1–10.

[3] M.-M. Bazm, M. Lacoste, M. Südholt, and J.-M. Menaud, "Side channels in the cloud: Isolation challenges, attacks, and countermeasures," 2017.

[4] M. Aldawood, A. Jhumka, and S. A. Fahmy, "Sit here: Placing virtual machines securely in cloud environments." in *CLOSER*, 2021, pp. 248–259.

[5] M. Rabbani, Y. L. Wang, R. Khoshkangini, H. Jelodar, R. Zhao, and P. Hu, "A hybrid machine learning approach for malicious behaviour detection and recognition in cloud computing," *Journal of Network and Computer Applications*, vol. 151, p. 102507, 2020.

[6] V. Natu and T. N. B. Duong, "Secure virtual machine placement in infrastructure cloud services," in *2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2017, pp. 26–33.

[7] V. D. Long and T. N. B. Duong, "Group instance: Flexible co-location resistant virtual machine placement in iaas clouds," in *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 2020, pp. 64–69.

[8] J. Han, W. Zang, S. Chen, and M. Yu, "Reducing security risks of clouds through virtual machine placement," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2017, pp. 275–292.

[9] K. Bijon, R. Krishnan, and R. Sandhu, "Mitigating multi-tenancy risks in iaas cloud through constraints-driven virtual resource scheduling," in *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, 2015, pp. 63–74.

[10] M. Bahrami, A. Malvankar, K. K. Budhraja, C. Kundu, M. Singhal, and A. Kundu, "Compliance-aware provisioning of containers on cloud," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 2017, pp. 696–700.

[11] E. Caron, A. D. Le, A. Lefray, and C. Toinard, "Definition of security metrics for the cloud computing and security-aware virtual machine placement algorithms," in *2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*. IEEE, 2013, pp. 125–131.

[12] F. Ahamed, S. Shahrestani, and B. Javadi, "Security aware and energy-efficient virtual machine consolidation in cloud computing systems," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 1516–1523.

[13] Q. Sun, Q. Shen, C. Li, and Z. Wu, "Selance: Secure load balancing of virtual machines in cloud," in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 662–669.

[14] A. Agarwal and T. N. B. Duong, "Secure virtual machine placement in cloud data centers," *Future Generation Computer Systems*, vol. 100, pp. 210–222, 2019.

[15] R. Sprabery, K. Evchenko, A. Raj, R. B. Bobba, S. Mohan, and R. H. Campbell, "A novel scheduling framework leveraging hardware cache partitioning for cache-side-channel elimination in clouds," *arXiv preprint arXiv:1708.09538*, 2017.

[16] M. A. Will and R. K. Ko, "Secure fpga as a service—towards secure data processing by physicalizing the cloud," in *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 449–455.

[17] M. Berrima, A. K. Nasr, and N. Ben Rajeb, "Co-location resistant strategy with full resources optimization," in *Proceedings of the 2016 ACM on Cloud Computing Security Workshop*, 2016, pp. 3–10.

[18] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against vm-colocation attacks in clouds," in *IFIP international information security conference*. Springer, 2012, pp. 388–399.

[19] Y. Qiu, Q. Shen, Y. Luo, C. Li, and Z. Wu, "A secure virtual machine deployment strategy to reduce co-residency in cloud," in *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017, pp. 347–354.

[20] J. Han, W. Zang, L. Liu, S. Chen, and M. Yu, "Risk-aware multi-objective optimized virtual machine placement in the cloud," *Journal of Computer Security*, vol. 26, no. 5, pp. 707–730, 2018.

[21] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[22] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 51–62.

[23] S. K. Garg and R. Buyya, "Networkcloudsim: Modelling parallel applications in cloud simulations," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE, 2011, pp. 105–113.

[24] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 205–218.

[25] T. Balharith and F. Alhaidari, "Round robin scheduling algorithm in cpu and cloud computing: a review," in *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*. IEEE, 2019, pp. 1–7.

[26] Y. Azar, S. Kamara, I. Menache, M. Raykova, and B. Shepard, "Co-location-resistant clouds," in *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security*, 2014, pp. 9–20.

[27] Y. Han, J. Chan, T. Alpcan, and C. Leckie, "Using virtual machine allocation policies to defend against co-resident attacks in cloud computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 95–108, 2015.