

Conflict-Based VM Placement Model for Cloud Computing

Department of Computer Science, University of Pretoria, Pretoria, South Africa

Abstract – Proper placement of Virtual machines (VMs) is a key component of Infrastructure as a Service (IaaS) for cloud service providers. Consequently, most VM placement algorithms focus on resource consolidation to improve placement efficiency and maximize on resource utilization both of which reduce energy consumption. Often times, resource consolidation results in tenants' VMs being bundled together in a group of few physical nodes. This increases the risk of placing VMs of potentially conflicting tenants on the same physical infrastructure, yet these would rather be placed on separate nodes in the physical infrastructure at minimum. The problem is that this causes a VM placement vulnerability in cloud computing. This vulnerability tolerates threats, such as Inter-VM attacks which enables malicious tenants to obtain unauthorised access to confidential data from other co-resident tenants. These type of threats demand new and novel countermeasures to be integrated into currently available VM placement algorithms. The solution proposed in this paper is based on a conflict-aware model, cognisant of the fact that tenants might have different degrees of conflict, resulting in different cost and risk implications. This model guarantees logical and physical isolation of conflicting tenants' VMs at different levels based on their conflict tolerance levels and it is validated using a worked out experiment.

1.0 Introduction

The capability to have VMs of multiple co-resident tenants on the same physical infrastructure makes a good business case for a public cloud service provider. Benefits, amongst others, include significant cost saving on hardware infrastructure and reduced energy consumption. However, the same capability increases the risk for unsuspecting tenants' VMs to share physical resources with malicious tenants' or competitors' VMs. This in turn increases the risk of confidential data leakage through inter-VM attacks. The threat of an inter-VM attack is most often overlooked due to its technical complexity. An inter-VM attack can be exploited by malicious tenants of public clouds to obtain unauthorised access to confidential data from VMs of unsuspecting co-resident tenants (Levitin, Xing and Dai, 2018).

Due of its technical complexity, this attack can easily go undetected for a long time (Levitin et al., 2018; Addya et al., 2017). This is because an inter-VM attack leverages on VM placement vulnerabilities inherent in the virtualization layer of cloud infrastructures. If it so happens that there are no proper triggers and alerts set in the virtual machine monitor (VMM) (Levitin et al., 2018), this attack would remain invisible on the physical layer of cloud computing and cloud administrators would not be able to detect it. This has been demonstrated, for example, in earlier work of Perez-Botero et al. (2013) and Ristenpart et al. (2009). Perez-Botero et al. (2013) demonstrate a malicious attacker exploiting vulnerabilities on a memory management unit in order to compromise the confidentiality of co-resident guest VMs' data. Ristenpart et al. (2009) on the other hand demonstrate a malicious user's guest VM exploiting vulnerabilities in the hypervisor to trick it to issue a command that could destroy another co-resident guest VM.

Quite often as demonstrated in earlier works of Perez-Botero et al. (2013) and Ristenpart et al. (2009) inter-VM attacks exploit covert channels that exist between co-resident tenants' VMs. Tenants with a malicious intent can also compromise a hypervisor to escalate their access privileges. Given that a hypervisor mediates all user access to shared resources, it becomes a target for attackers that want to gain unauthorised access to confidential data of all co-resident guest VMs. Consequently, some research work is focused on eliminating the hypervisor because it is argued to act as a single point of failure (Martins et al., 2017; Pearce, Zeadally and Hunt, 2013).

This paper does not advocate for the removal of the hypervisor. However, it acknowledges the technical complexity of dealing with inter-VM attacks and then makes an attempt to answer the research question; on how to improve VM placement and facilitate their physical isolation in order to contain data leakage threats posed by inter-VM attacks? The authors theorises that a proper placement of VMs, taking cognisance of risk exposure and cost implications from the tenants' perspective could help to minimise the impact of inter-VM attacks. The main contribution is a conflict-aware VM placement model called the Conflict-Based Control for Cloud (CBAC4C).

The remainder of the paper is structured as follows: Section II provides related work. Section III briefly discusses the Chinese Wall Model and then demonstrates its unsuitability to address the VM placement problems in the cloud and help prevent inter-VM attacks. Section IV presents CBAC4C, highlights and uses its rules for VM placements on the cloud a worked out example. Section VI briefly provides experimental details and empirical results that demonstrate the practicability and suitability of the CBAC4C model to address the VM placement vulnerability which is exploited by an inter-VM attack. Section VII concludes the paper.

2.0 Related Work

The VM placement problem in the context of cloud computing has been extensively studied by different researchers (Ferdaus et al., 2017; Filho et al., 2018; Quan, Wang and Ren, 2017; Levitin, Xing and Dai, 2018). Consequently, several research efforts attempt to address the VM placement problem in cloud computing, albeit from different view-points as captured and summarized in a number of surveys (Filho et al., 2018; Challita, Paraiso and Merle, 2017; Madhusudhan and Satish, 2017; Thulo and Eloff, 2017; Alnajdi et al., 2016; Kaur and Bhardwaj, 2016; Masdari et al., 2016 and Pires and Baran, 2015). The next subsection reviews each of these surveys. The main goal is to identify the research gap in existing VM placement literature.

2.1 Surveys on VM Placement

For example, Filho et al. (2018) provides a comprehensive review of the state-of-the-art on VM placement in the cloud highlighting open issues and challenges whilst reflecting on its relevancy in an increasing and demanding market. Filho et al. (2018) reviews and classifies different approaches that seek to address the VM placement problem in cloud computing in an effort to identify open issues and provide pointers for future solutions. Unfortunately, this work (i.e. Filho et al. (2018)) does not identify the security issue of VM placement. Therein, there is also no mention of cost or risk that are associated with the physical isolations of VMs belonging to conflicting tenants. Similar to Filho et al. (2018), the work of Alnajdi et al. (2016) provides a critical analysis of existing dynamic VM placement algorithms to outline open research challenges and provide directions for future work. This work (i.e. Alnajdi et al., 2016) also disregard the open issue of security associated with VM placements.

Challita et al. (2017) reviews VM placement literature that focuses on reducing power consumption, maximize resource utilization and avoid traffic congestion. Challita et al. (2017) mention security as one area that remains unresolved with respect to VM placement. Kaur and Bhardwaj (2016) reviews VM placement algorithms that focuses on resource consolidation. Kaur and Bhardwaj's work is focused on research efforts that attempt to minimize the number of physical nodes to allocate VMs, minimize VM allocation time and power consumption. Similar to the work of Kaur and Bhardwaj (2016) is the work of Usmani and Singh (2016) which also provides a comprehensive literature review of the state-of-the-art VM placement and resource consolidation techniques with the aim to minimize and improve energy consumption which they claim is increasing to unacceptable levels. The work of Madhusudhan and Satish (2017) focuses on reviewing and classifying VM placement algorithms that attempt to consolidate resources in order to maximize resource utilization and minimize energy consumption. Three of these surveys (i.e. Kaur and Bhardwaj (2016), Usmani and Singh (2016), and Madhusudhan and Satish, (2017)) fail to point out the impact of VM placement consolidation on the security implications thereof. Only the work of Challita et al. (2017) does so.

Thulo and Eloff (2017) reviews existing literature on VM placement algorithms. The findings thereof show that there is research gap on research efforts that consider the security aspect of VM placement (Thulo and Eloff, 2017). Therefore, and in order to close the gap, Thulo and Eloff goes further to investigate existing optimized VM placement algorithms that at least have a potential to be further augmented with security features. Masdari et al. (2016) reviews and classifies VM placement schemes based on their VM placement algorithm and evaluates their capabilities and objectives. However, and similar to the work of Thulo and

Eloff (2017), Masdari et al. also mention that there is research gap with regards to research work that addresses the security aspect of VM placement algorithms. Pires and Baran (2015) reviews and classifies VM placement literature with respect to QoS, energy efficiency, service level agreements and resource consolidation. Their findings show that some researchers are to a lesser extent starting to focus on the security issues of VM placement algorithms (Pires and Baran, 2015). However, this work also points to the fact that this issue is not being addressed at scale as expected.

In summary, the covered surveys reflect that the VM placement problem in cloud computing has been extensively studied from different view-points. However, the covered surveys all point to the lack of research efforts that focuses on the security implications of VM placement algorithms. It is good to note that there are some isolated research efforts that have been identified as moving towards covering this research gap. However, these are still insufficient and wide apart. The next subsection discusses some of these research efforts.

2.2 Security-aware VM Placement

Albeit the insufficient research efforts that investigate security-related implications of VM placement algorithms, there are some isolated and fragmented efforts. For example the work of Levitin et al. (2018); Thulo and Eloff (2017); Ahamed (2016) and Shetty, Yuchi and Song, (2016) which investigate data leakage threats as a result of security-related vulnerabilities in VM placement algorithms.

Levitin et al. (2018) models data security and survivability requirements to address an inter-VM attack that exploits a co-resident based data vulnerability to leak or corrupt data held in a co-resident target's VM. Levitin et al proposes a data replication technique that partitions a tenant's data into multiple blocks and randomly distribute them in multiple servers to enhance security. This work (i.e. Levitin et al., 2018) goes further to create multiple replicas for each block to improve data survivability in a cloud that is subject to inter-VM attacks. This is a good approach to deal with data corruption. However, it comes at a high cost in terms of the underlying infrastructure that holds the multiple replica data blocks. Furthermore, the proposed random placement of data blocks to different servers does not guarantee non-co-residence of an attacker and target VM. Therefore, the proposal therein cannot be argued to really address the problem of data leakage through a targeted inter-VM attack.

Thulo and Eloff (2017) proposes a solution that uses optimized traffic and network-aware VM placement algorithms as a baseline and incorporate security features with a goal to achieve an optimized security-aware VM placement algorithm. However, this study ends before it can reflect how this augmentation is to be done and without mentioning the actual security features that are to be considered. Ahamed (2016) follows the approach of Thulo and Eloff (2017) which proposes a solution that adds security features on optimal existing VM placement algorithms. Ahamed starts from an assumption that VMs have vulnerabilities and then profiles each VM according to its associated vulnerabilities as depicted in Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (CVSS) databases. The security profiles are then ranked and used in what they refer to as a security-aware and energy efficient VM placement solution. This approach places what they consider vulnerable VMs together and those that are not so vulnerable together.

Shetty, Yuchi and Song, (2016) follows a similar approach to that of Ahamed (2016) and evaluates the vulnerability of VMs based on the National Vulnerability Database (NVD). The probability of risk for each VM is calculated based on its connections and dependencies with other VMs. The resultant evaluation is used to quantify the overall security risk of physical hosts. This is directly proportional to the total sum of vulnerabilities of all VMs hosted in each physical host. The proposed security-aware VM placement algorithms therein (i.e. Shetty et al., 2016) ensure that VMs with high risks are placed in low survivability hosts and separate from those with a low risk which are placed in high survivability hosts. This is argued therein to somehow eliminate the possibility of placing highly vulnerable VMs or 'bad neighbours' on physical hosts that host low risk VMs.

In summary, only a few researchers are investigating the security implications of VM placements on the cloud. There is a big need for more research efforts in this space. For example, there is a need to look at the placement of VM belonging to conflicting tenants.

2.3 Conflict-aware VM Placement

There are some research efforts that have already looked at the issue of placing VMs in a manner that would handle conflicting tenants' requirements. The work of Mashayekhy et al. (2014); Si et al. (2013); Yu et al. (2014); Narwal, Kumar and Sharma (2016); Kwiat et al (2015); Han et al. (2015), (2013) and (2014) have already made some stride on this topic.

Mashayekhy et al. (2014) considers data protection requirements in terms of restricting VM co-residence and co-location in 'trust restrictions' and 'disclosure restrictions'. This paper investigates data leakage threats posed by inter-VM attacks on conflicting tenants sharing the same provider's cloud infrastructure. Mashayekhy et al. (2014) then proposes cryptographic mechanisms to solve the problem. However, reliance on cryptographic solutions alone cannot address the inter-VM problem. Cryptographic solutions provide an added layer of security but cannot stop an inter-VM attack from leaking encrypted data.

Si et al. (2013) proposed a security awareness VM placement scheme (SVMPS). This scheme proposes two conflict-of-interest relations, namely "aggressive conflict of interest relation" (ACIR) and "aggressive in ally with relation" (AIAR). Both relations are based on Brewer and Nash's Chinese Wall model and are used to enhance isolation between conflicting tenants of the same cloud provider. However, the isolation is limited to the host level. The work of Yu et al (2014) seems to be extending the work of Si et al. (2013). It also proposes a security-aware VM management scheme based on the CWM. However, this work only ensures that VMs of conflicting users are placed on different physical nodes. This approach partly addresses the problem but it is still insufficient to achieve a more desirable solution which could ensure an even wider separation of conflicting tenants' VMs. This work also fails to address the different degrees of CoI which would introduce some flexibility to the proposed model.

Narwal et al. (2016) make use of game theory to compare different VM placement policies and determine one that minimizes an attacker's possibility of co-residence with other conflicting tenants. The main parameters in this game theory model are attackers and defenders. The attackers tries to co-locate as many of their malicious VMs with as many target VMs as possible (i.e. increase efficiency and coverage of their co-residence placement). The defenders use a set of VM placement policies instead of one to do the placement. This is such that when a VM placement request comes, the defender randomly selects a VM placement policy with a pre-defined probability from the set of all policies to do the placement. This work also maintains a workload balance and minimum power consumption (Narwal et al., 2016).

Kwiat et al (2015) uses game theory to demonstrate the interdependency between users of the cloud. This interdependency is such that a vulnerability in a VM of one user affects other co-resident tenants' VMs and the controlling hypervisor. Kwiat et al. (2015) assumes that every users is rational and make decisions that maximizes their payoff. The end goal is to minimize the negative effects of the interdependency of users' co-located VMs. Even though Kwiat et al. (2015) somehow addresses the security issues of the cloud, it does not address the prevalent issue of conflict of interest amongst co-located VMs.

Han et al. (2014) propose a VM placement policy that places a new VM to a physical node that already has the highest number of VMs. This is aimed at resource consolidation. Han et al. (2015) extends Han et al. (2013) and (2014) with a mathematical formulation of the solution to mitigate the threat of inter-VMs attacks on co-residents whilst satisfying constraints in workload balance and power consumption. In an attempt to prevent an attacker from starting too many VM instances in order to improve an attacker's efficiency and coverage as discussed in Han et al. (2013), Han et al. (2015) put all VMs of a tenant on the same physical host. This approach helps to control inter-VM attacks since at the host level there would be no co-residence of malicious and non-malicious VMs. However, for a normal tenant, having all your VMs instantiated on a single host creates a single point of failure.

Bagnasco, Vallero and Zaccolo (2017) propose a fair scheduling service (FaSS) for OpenNebula cloud infrastructure. The FaSS solution prioritizes VM placement requests based on an initially assigned weight and historical resource usage. The FaSS is designed similar to Haiza scheduling algorithm (Molto and Caballer, 2016) and they both support OpenNebula cloud infrastructure. Both of them interface with the existing scheduler without interfering with its underlying code and logic. However, both these algorithms do not address the issue of VM isolation to minimize in a cost effective manner the risk of confidential data leakage posed by inter-VM attacks.

In summary, the overview of related work has highlighted a number of shortcomings of existing approaches:

- The surveys reflects that there is insufficient research work that focuses on security-aware VM placement algorithms.
- The covered research on security-aware VM placement is not sufficient to full address the security aspect of VM placement on the cloud.
- The literature on conflict-aware VM placement does not address the need for solutions that consider different levels of physical isolation
- The literature on conflict-aware VM placement does not take into account the cost and risk implications thereof.
- Existing work on conflict-aware VM placement seems to be taking a rigid approach (i.e. either you are in conflict or not) to managing CoI without considering the varying degrees of conflict

The paper at hand extends the work of Dlamini et al. (2014) and Ratsoma et al. (2015), which proposes a conflict-aware placement of VMs. The next section briefly describes the Brewer & Nash's CWM as it forms the basis for the proposed solution in this paper.

3.0 Chinese Wall Model (Brewer & Nash, 1989)

The CWM is a specialized access control policy that addresses issues of confidentiality in domains that are sensitive to conflict of interests (CoI). A CoI is associated with an inadvertent leakage of confidential data to competitors. The CWM defines impenetrable walls that segment data and enclose it in mutually disjoint CoI classes to avoid inadvertent access and leakage of confidential data to competitors.

The CWM starts off as a free choice model and ends with tight restrictions on subsequent access requests. For example, at the beginning all subjects are free to choose any object. However, once the initial choice has been made, the subject can only access already accessed objects, or those in the same dataset and the ones belonging to different COI classes. This model is based on a set of rules that defines how subjects can access data in a manner that completely avoids conflicts of interest between competitors. The model rigidly enforces access such that there is no flexibility for partial conflicts. Furthermore, it is sensitive to the history of past access. Below is a discussion on the CWM property rules.

3.1 Chinese Wall Model Rules

The CWM has two distinct yet closely related rules. These are the simple security and * - property rule.

3.1.1 A Simple Security Rule

This rule states that a subject may be granted access to an object only if the object:

- is in the same company dataset as the object already accessed i.e. inside the same wall of conflict based on the assumption that a company belongs to only one *CoI*
- belongs to a different company in a different *CoI* class. For example, if a subject accesses an object from Bank A he or she will subsequently be blocked from accessing any objects from Bank B or BANK C. However, the subject is free to access any objects from any company in any other conflict class.


3.1.2 *-Property Rule


This rule pertains to write access which is permitted only if:

- access is granted by the simple security policy
- no object can be read which
 - is in a different company dataset than the one for which write access is requested
 - contains information that is not sanitized

3.2 Applying the CWM to prevent confidential data leakage in the cloud

The diagram (figure 1) illustrates how the CWM could be used to place VMs belonging to different companies from different CoI classes on a public cloud infrastructure. Assume that we have three functional business domains (automobiles, banks and clothing) which basically translates to three CoI classes i.e.

CoI_A → {Audi, BMW} →  blue triangle representing its VMs

CoI_B → {Abs, Bob, Chi} →  orange triangle representing its VMs and

CoI_C → {Alex, Boo, Cho} →  green triangle representing its VMs

Using the CWM, all VMs that belongs to a specific company are placed in the same physical node (PN) or nodes. For example, figure 1 shows that VMs belonging to Audi which an element of CoI_A can only be placed in PN₁ or PN₂ and all those associated with BMW are placed in PN₃ or PN₄. The co-location of a tenant's VM using CWM would greatly improve the disk seek time and help speed up access to VMs. This will address the latency which is associated with accessing VMs in different PNs. However, and given this setup, the best physical isolation of directly conflicting companies' VMs (i.e. Audi and BMW) can only be a cluster apart. Hence, it would be easy for an adversary to map out the cloud infrastructure in order to gain co-residence with their target. For an attacker to map out the cloud infrastructure and identify their target, they only need to find out the number of clusters in the CSP. Once that is done, they then need to get at least one tenant in each cluster to get to know where their target VM is hosted.

The CWM was designed to control access requests from consultants working on different datasets from different companies. Hence, this model would work well to control employees' access to tenants VMs. It could be easily used to prevent a CSP's employees from accessing VMs from conflicting tenants. For example, once an employee has access to Audi's VM, subsequent access to BMW's VM would be denied. However, access to any of the Banks or CDs would be granted. Moreover, after the initial access, any access to another company's VM within the same CoI is denied. This practically means that, one employee can have at most three VMs from companies in different CoI. This would be an ideal case if CSPs were using this approach to control their employees' access to tenants' VMs.

Furthermore, the CWM performs badly when it comes to resource utilization. For example, assume that a company from each CoI make successive requests to place their VMs in the cloud infrastructure. Assuming these three require VMs to host public data with minimal or no risk. This does not necessarily need any physical isolation boundaries. Under the current conditions, the CWM policy would place each of the VMs in separate PNs within the different CoI classes. Such a scenario would result in a cloud infrastructure that has VMs scattered all around. For example, it would happen that instead of using one PN to host these, the CWM would host them in three separate PNs. This is not an ideal case if one considers energy utilization. Hence, for a CSP to ensure maximum utilization of resources, it should be possible to place these different requests on the same PN; provided that there is sufficient storage capacity. The above short-comings reflect the unsuitability of the existing CWM to appropriately place tenants' VMs on the cloud. Hence, the next section discusses a Conflict-Based Control for Cloud Computing (CBAC4C) – a model based on the CWM but tailored for VM placement in cloud computing (i.e. OpenNebula cloud infrastructure).

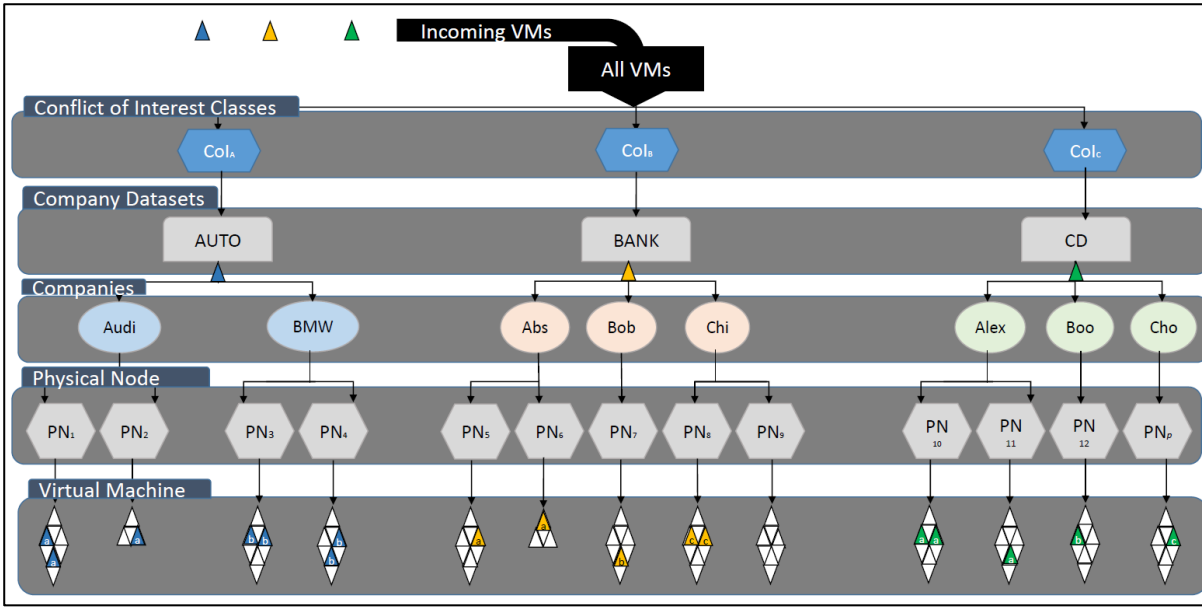


Figure 1: An Example that illustrates the unsuitability of the CWM in VM placement on the cloud

4.0 CBAC4C – A VM Placement Model Based on the Chinese Wall Model

4.1 Description of the CBAC4C Model

The CBAC4C model extends the CWM policy where each tenant belongs to only one CoI class. In a similar manner, this paper also retains the concept of a CoI as a grouping of tenants from the same business domain as defined for CWM in the above section. Furthermore, it is assumed that tenants' data sets can be assigned to one or more VMs. Based on these assumptions and the shortcomings identified in the current body of knowledge, the requirements for applying the revised CWM in a cloud environment using our CBAC4C are as follows:

Requirement 1: Focus on infrastructure as a service (IaaS).

Requirement 2: Provide VM isolation based on different degrees of CoI, referred to as conflict tolerance level (CTL).

Requirement 3: Minimise the risk of data leakage caused by inter-VM attacks in a cost-effective way and based on the CTL. The next sections start off by modelling a CoI relationship which is not modelled in the covered related work. This relationship modelling is important to clarify the concept of CoI as used within this paper.

4.2 CoI Relationship in CBAC4C

A CoI relation is defined as follows:

- A TenantA cannot be in conflict with itself. This implies that a CoI relationship is not reflexive.
- If TenantA is in conflict with TenantB; the reverse is also true; that TenantB is also in conflict with TenantA. This implies that a CoI relation is mutually inclusive. Therefore, based on this property, a CoI relationship is symmetric.
- If TenantA is in conflict with TenantB who is in conflict with TenantC. Despite the relationship between TenantA and TenantB; and that relationship between TenantB and TenantC, this does not necessarily imply that TenantA is in conflict with TenantC. Hence, a CoI relationship is not transitive.

Therefore, this paper models a CoI relationship as neither an equivalence nor a binary relationship.

4.3 Risk, Physical Security and Cost

The proposed model takes cognisance of the fact that tenants can have different degrees of conflict with other tenants. Hence, the risk (R) of confidential data leakage also varies depending on the degrees of conflict. For example, co-residence of VMs that belong to two directly competing tenants (i.e. those that are in direct conflict of interest with one another), poses the highest risk, R.

Ideally, VMs belonging to tenants that are in direct conflict could be placed far apart. However, CSPs are likely to charge a high cost C for non-conflicting placement of tenants' VMs and relative low cost C for flexible co-resident placement with conflicting tenants' VMs. Hence, a physical separation S is determined for implementing CTLs, indicating how much physical separation a tenant requires between its VMs and that of its competitors. This is followed by determining a cost value C for implementing a CTL. A risk exposure level R is directly proportional to and associated with a CTL. Therefore, the physical separation of VMs requires a modification to the physical architecture of the cloud in the following manner; VM, PN, clusters (Clu), data centre's (DC) and geographical locations (Loc) as proposed in Eloff et al. (2014):

$$VM \subseteq PN \subseteq Clu \subseteq DC \subseteq Loc$$

This means that a company's dataset is held in a VM which is held in a PN that exists in a Clu. A Clu is found within DCs and DCs are contained in Locs. This tree-like structure is a one to many relationship between Loc –DC; DC – Clu; Clu – PN and PN – VM. This means that our solution requires a matching underlying cloud tree-structure. The next section discusses the CBAC4C VM placement policy based on the requirements and the above cloud structure.

4.4 CBAC4C Rules

This model adopts the access rules of the CWM policy and changes them to VM placement policy rules. The idea is to place VMs in a conflict-aware manner on the cloud infrastructure. These rules considers R, S and C when placing VMs. The four rules are as follows:

Rule 1: PNs can be shared between non-conflicting businesses.

This rule makes it possible for VMs belonging to non-conflicting businesses to share the same PN (i.e. if there is sufficient storage space). The assumption is that this poses no risk for confidential leakage through inter-VM attacks. This is a similar proposition to the one in the CWM policy which ensures that a consultant may have access to datasets belong to companies that are not in conflict with each other.

A VM can be placed in a PN if the following three conditions hold:

- The target PN has sufficient storage capacity to hold the incoming VM
- The target PN is empty
- All existing VMs in the target PN belong to a different CoI than that of the incoming VM (there are no conflicts)

Rule 2: Companies that are in conflict with each other might opt to run their VMs on the same PN.

This rule enables a flexible and cost effective way for companies to host their data on the cloud. For example, it might happen that a company that is in direct conflict with another has public data to host in a VM which is co-resident in a PN that already hosts a competitor's VM. Such a scenario is catered for by the following flexible rule which allows conflicting companies to share a physical node at a low cost. Hence, the rule is as follows:

A VM can be placed co-resident with that of a conflicting tenant if the following conditions hold:

- The target PN has sufficient storage capacity to hold the incoming VM

- If the target PN has at least one existing VM that belong to the same CoI as that of the incoming VM.

Rule 3: Companies that are in conflict with each other might opt to run their VMs on different physical nodes but on the same cluster.

Following the same reasoning of flexibility and cost effectiveness as in rule 2, the cost of placement on different nodes would be more expensive than that of co-hosting on the same physical node. This is related to the risk posed on the VMs. The further apart we place VMs of conflicting businesses, the better. However, this gets more expensive in terms of the cost as the risk of confidential data leakage diminishes as we place VMs of conflicting businesses further apart. Hence, the rule is as follows:

A VM can be placed in the same Clu with that of a conflicting company but in different PNs; if the following conditions hold:

- The target PN in the target Clu has sufficient storage capacity to hold the incoming VM
- The target PN in the target Clu is empty and with sufficient storage capacity to hold the incoming VM
- If the target PN in the target Clu has existing VMs, none of these must belong to the same CoI as that of the incoming VM.

Rule 4: Companies that are in conflict with each other might opt to run different VMs on different physical nodes, on different clusters but same data centre

The cost of placement on different clusters would be more expensive than that of co-hosting on the same cluster. Hence, the rule is as follows:

A VM can be placed in the same DC with that of a conflicting company but in different PNs from different Clus; if the following conditions hold:

- The target PN in the target Clu and DC has sufficient storage capacity to hold the incoming VM
- The target PN in the target Clu within the target DC is empty and with sufficient storage capacity to hold the incoming VM
- If the target PN in the target Clu within the target DC has existing VMs, these must belong to a different CoI class from that of the incoming VM.

Rule 5: Companies that are in direct conflict with a high risk exposure with each other must run their VMs on different physical nodes, different clusters and different data centres.

At an even higher degree of conflict VMs from conflicting businesses could be placed in different PNs hosted in different Clus and different DCs. A VM can be placed to be in a different DC within the same location with that of a conflicting company but in different PNs from different Clus; if the following conditions hold:

- The target PN in the target Clu and DC has sufficient storage capacity to hold the incoming VM
- The target PN in the target Clu within the target DC is empty and with sufficient storage capacity to hold the incoming VM
- If the target PN is in a different DC but same Loc with VMs of its conflicting partners, none of these must belong to the same CoI as that of the incoming VM.

The highest degree of conflict is demonstrated in Rule 6 below.

Rule 6: Companies that are in direct conflict might in extreme cases with a very high risk exposure decide to run their VMs as far apart as different geographical locations.

A VM can be placed to be in a different Loc inside a different DC, a different Clu and a different PN with that of a conflicting company. This is one of the most expansive options but the risk of confidential data leakage is at its bare minimum at this level. A VM can be placed in this category if the following conditions hold:

- The target PN in the target Clu and DC within the target Loc has sufficient storage capacity to hold the incoming VM
- The target PN in the target Clu within the target DC held in the Loc is empty and with sufficient storage capacity to hold the incoming VM
- If the target PN is in a different Loc as that of its competitor, none of the VMs within the Loc must belong to the same CoI as that of the incoming VM.

Note: Rule 2, 3, 4 and 5 refer to the cost aspect of physically placing a VM. This paper models the cost associated with the proposed placement in such a manner that the further apart we place VMs of conflicting companies, the lower the risk of confidential data leakage. However, this comes with a cost implication to the companies but it is profitable for the cloud service provider. Hence, in Rule 2, 3, 4, and 5, we provide that we can allow conflicting businesses to co-reside their VMs with that of conflicting businesses. This basically stems for the fact that it is not all cloud hosted data that a company would want to keep away from its competitors. This model is able to address the challenges of CWM on VM placement on the cloud. The next section discusses an example on how the above rules are actually implemented and put into practice for conflict-aware VM placement on the cloud.

4.5 Example of CBAC4C VM Placement

This section uses figure 2 below to demonstrate how CBAC4C does VM placement on a cloud infrastructure. The example considers a fictitious cloud service provider that has its DCs distributed in three geographical Locs. For illustration and simplicity purposes, the example has a one-to-one mapping between Locs and DCs. However, from DC downwards, we preserve the one-to-many relationship. Building on the scenario raised in figure 1, assume that we have three functional business domains (automobiles, banks and clothing) which basically translates to three CoI classes i.e.

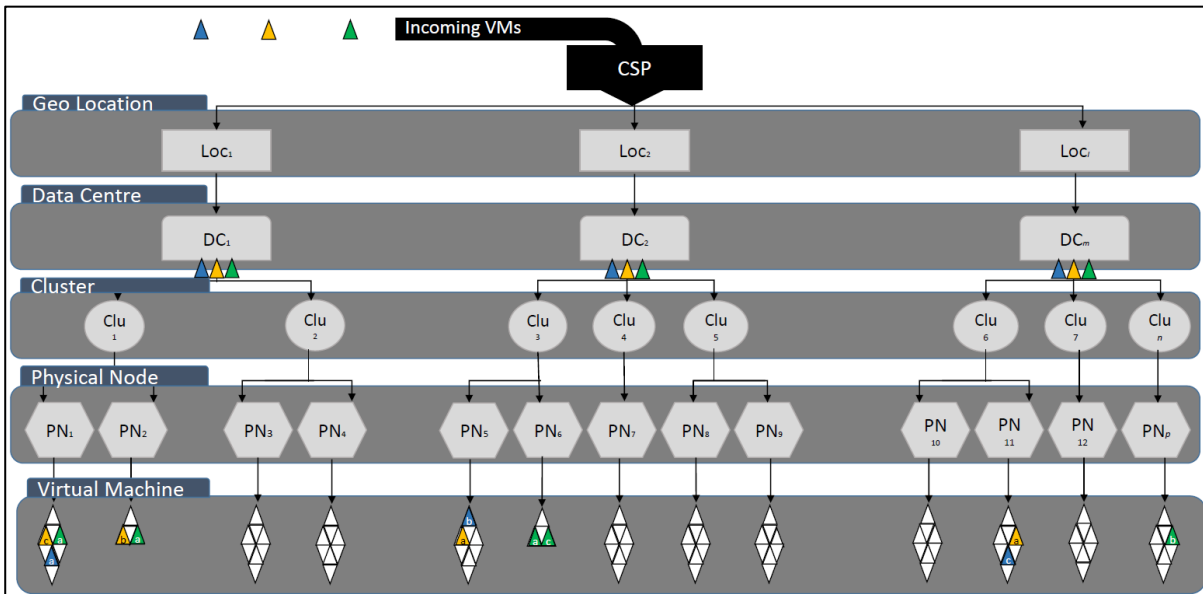


Figure 2: An Example that illustrates VM placement on the cloud using CBAC4C

$CoI_A \rightarrow \{Audi, BMW\} \rightarrow$ blue triangle representing its VMs

$CoI_B \rightarrow \{Abs, Bob, Chi\} \rightarrow$ orange triangle representing its VMs and

$CoI_C \rightarrow \{Alex, Boo, Cho\} \rightarrow$ green triangle representing its VMs

Using CBAC4C rules ensure that VMs belonging to that same CoI could still be allowed to share the same PN. For example, we have a scenario as depicted in PN_1 i.e. $Loc_1.DC_1.Clu_1.PN_1$ which implements Rule 1. To illustrate Rule 2, both VMs that belong to Alex and Cho who are in direct conflict with each other are sharing the same PN, i.e. $Loc_2.DC_2.Clu_3.PN_6$. Using Rule 3, VMs that belong to Bob and Cho who are in direct conflict with each other are sharing the same Clu, i.e. $Loc_1.DC_1.Clu_1$ and using different PNs i.e. PN_1 and PN_2 . In order to implement Rule 4, consider Audi having a VM in $Loc_1.DC_1.Clu_1.PN_1$ and BMW placing theirs in $Loc_1.DC_1.Clu_1.PN_3$ or PN_4 . Taking, the extreme i.e. using Rule 6, a VM from Audi is located in $Loc_1.DC_1.Clu_1.PN_1$ and that of its direct conflicting partner is hosted in $Loc_1.DC_m.Clu_6.PN_{11}$.

Taking a close look at figure 2, the varying degrees of conflicts are well addressed. Furthermore, the VMs are almost evenly spread across the cloud infrastructure which has a significant impact on resource utilisation. However, if the CBAC4C model is stringently applied to totally avoid conflicts, the size of the cloud in terms of the number of Clus, DCs and Locs would be directly proportional to the conflicting tenants. In that case our solution would not be scalable for practical implementation. More importantly though, our solution can significantly address the problem of confidential data leakage from competitors in direct conflict if Rule 6 and Rule 1 are rigorously enforced. Rule 2 – 5 are somehow flexible, but provides a cost-effective approach to VM placement where businesses are allowed to take a calculated risk to have their ‘not so mission critical’ VMs co-resident with that of their conflicting partners.

5.0 Empirical results

A number of experiments were conducted on a live OpenNebula cloud infrastructure running a KVM hypervisor to test and evaluate the effectiveness and efficiency of the proposed conflict-aware VM placement algorithms. The choice of the live OpenNebula cloud environment is based on the fact that it matches and is a direct fit of the proposed cloud structure i.e. adopted in this paper;

$$VM \subseteq PN \subseteq Clu \subseteq DC \subseteq Loc$$

Furthermore, the choice for OpenNebula cloud infrastructure is motivated by the comparison and analysis in Llorente (2013); Lynn et al., (2015), and Suganya and Kannan (2018). Llorente (2013) compares four main open-source players in the cloud computing ecosystem i.e. Eucalyptus, CloudStack, OpenStack and OpenNebula. These are analysed and compared based on their ability to adapt to datacentre virtualization that can be customized to provide differentiated cloud services. The results therein (Llorente, 2013) are in favour of OpenNebula which demonstrates a high level of flexibility and highly virtualization of cloud services. Lynn et al. (2015) and Suganya and Kannan (2018) use a feature-based comparison to illustrate the different offerings without necessarily ranking them. Hence, based on its suitability to the proposed tree-structure and its flexibility, the authors have chosen to experiment with OpenNebula for this research.

5.1 OpenNebula Cloud Infrastructure

Figure 3 shows an architecture for OpenNebula cloud infrastructure which comprises of three layers i.e. tools, core and drivers. The tools layer provides graphical interfaces (GUI - Sunstore) that allow users to interact with and manage VM instances. These include the command-line interface through which a user may enter commands. This also contains the cloud servers’ interface which manages external cloud servers. Lastly, the tools layer contains a scheduler which manages VM placement. This paper focuses on extending the work of the scheduler using the CBAC4C. Hence, CBAC4C is placed within the scheduler in red to show that it is an additions tool to ensure conflict-aware VM placement.

The second layer contains Application Programming Interfaces (APIs) that offers built-in methods that offers core functionality of the OpenNebula. This comprises of openNebula cloud API (OCA) which uses ruby and java bindings. The XML RPC API which uses XML bindings.

The third layer comprises of the core OpenNebula functionality that provides virtualization, monitoring, storage, images, networking and authentication features. The virtualization subsystem interacts with the KVM hypervisor in the PNs and is responsible for each step of a VM lifecycle. The monitoring subsystem execute a set of static probes provided by OpenNebula to collect VMs’ and PNs’ status, their basic performance indicators and capacity consumption and send it to the front-end node for processing.

The storage subsystem uses datastores (which is any storage medium e.g. storage area network (SAN) or direct attached storage) to store VMs' disk images. For the experiments, this paper made use of direct attached storage. This meant that when VMs were deployed, their images had to be transferred from the datastore to the different PNs. The images subsystem is used to setup operating systems or datablocks images for virtual machine instances. The networking subsystem provides flexible and customizable network services that makes it possible to integrate network requirements of datacentres. Finally, the authentication subsystem provides different authentication mechanisms such as SSH, X509, LDAP, AD and built-in authentication systems.

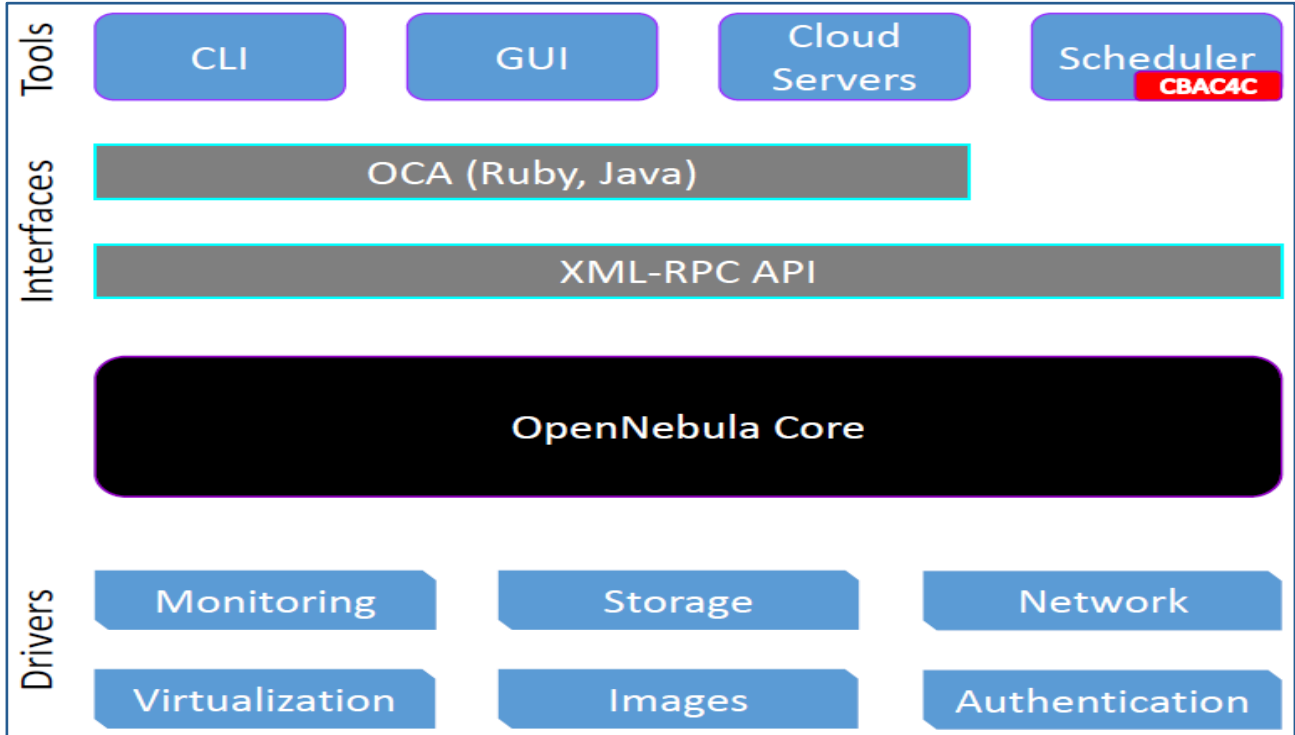


Figure 3: A Conflict-aware Opennebula Cloud Architecture (Adapted from the OpenNebula Project, 2018)

Since, the focus of this paper is on the VM placement, the authors will not discuss these any further. For further discussion on each of the traditional OpenNebula systems and subsystems the reader is directed to the OpenNebula Project (2018) page. The next section focuses on the scheduler and its associated traditional VM placement policies. The discussion then move to show how the authors have implemented the CBAC4C within the scheduler.

5.2 OpenNebula Scheduler

The OpenNebula cloud infrastructure comes with a standard first-in first-out (FIFO) match-making scheduler (mm-sched) which is responsible for VM placement (Podolnikova, 2016). A traditional mm_sched implements a ranking scheduling policy. The ranking scheduling policy comes with five different options of VM placement algorithms i.e. 0 - packing algorithm, 1 Striping algorithm, 2 – Load-aware algorithm, custom algorithm (this is a placeholder for user defined placement algorithms) and 4 – fixed algorithm (OpenNebula Project, 2018). These are briefly discussed in table 1.

The authors made use of the custom option to implement the conflict-aware VM placement rules which are stipulated in a set of four algorithms. The first two algorithms are used for initial VM instance placement and the last two are user for migrating an already existing one.

5.3 Experiment Setup

The experiment were setup on three machines all running Ubuntu 16.04 LTS with the following specification. The front-end node runs an Intel core i7-6700 at 3.4 GHz, has eight cores, 7.7 GiB memory and about 980 GB direct access storage to use for the VM placement. The first worker node runs an intel core i7-2600 at 3.4GHz, with eight cores, 7.8 GiB memory and about 1,4TB direct access storage. The

second worker node runs an Intel Pentium dual core E2180 at 2 GHz and with about 100 GB direct access storage. This setup gave us about 2,5 TB direct access storage which basically meant that our VM instances had to be very small.

5.3 Evaluation and Discussion of Experimental Results

The authors have defined a set of four VM instances i.e. small, medium, large and extra-large for OpenNebula cloud. This is similar to the naming convention in Amazon Web Services (AWS) IaaS. Table 1 shows the corresponding number of VM instances that were used in each of the experiments conducted on CloudSim with a CTL of 0 - 4 for all VM placements.

Table 1:- Existing VM Placement Policies in OpenNebula's scheduler and their Brief Description (OpenNebula Project, 2018; Podolnikova, 2016)

Policy Option	VM Placement Algorithm	Short description
0	Packing	This algorithm packs VMs in a minimum number of PNs. The algorithm ranks PNs based on their VM load. It prioritizes those PNs with more VMs running. This is to reduce VM fragmentation.
1	Striping	This algorithm spreads VMs in PNs with maximum available resources. This algorithm ranks PNs based on the maximum available resources. It prioritize PNs with less VMs.
2	Load-aware	This algorithm places VMs in PNs with less load to maximize available resources. It prioritizes PNs with more CPU available.
3	Custom	Uses a custom or user defined rank. This is a user defined algorithm that could be added.
4	Fixed	This algorithm requires the user to manually sort the PNs based on a user-set priority.

Figure 4 is a diagrammatic illustration of how this paper uses CBAC4C to extend the traditional mm-sched with the conflict-aware VM placement algorithms.

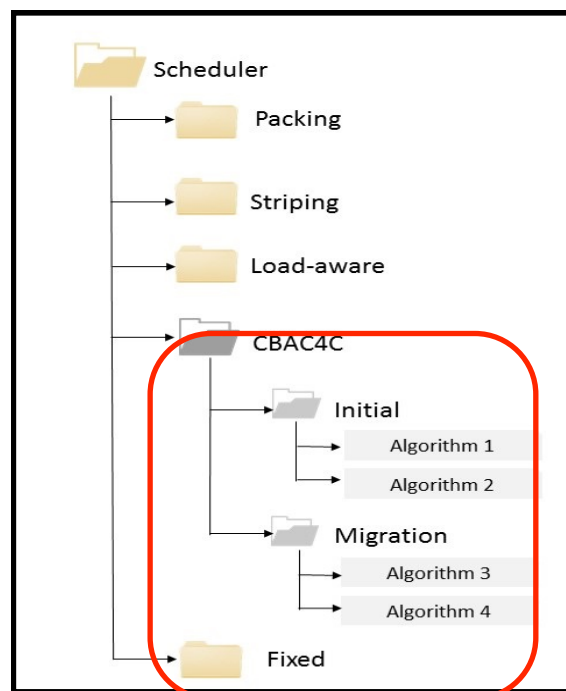


Figure 4 :- Extension of mm_sched with CBAC4C

CTL 0 referring to an instance that does not have any conflict restrictions. This is for example a VM instance that holds public data. CTL of 4 refers to a VM that holds Top secret data which has the highest conflict restrictions. The following table shows the number of VM instances that were used in our live OpenNebula cloud. For experiment one in OpenNebula cloud, there were 5 small, 5 medium, 10 large and 25 extra large VM instances. Experiment two had 6 small, 15 medium, 20 large and 43 extra large VM instances. Experiment three had 6 small, 12 medium, 34 large and 60 extra large VM instances. And experiment four had 8 small, 18 medium, 40 large and 65 extra large VM instances. This is summarized in table 2.

Table 2:- VM Instances in OpenNebula Cloud Platform

	VM Instance Classes in OpenNebula			
Experiment No.	Small	Medium	Large	Extra Large
1	5	5	10	25
2	6	15	20	43
3	6	12	34	60
4	8	18	40	65

In this setup the number of VM instances are quite small. This is because in a live cloud environment, the number of instances is constrained by the size of the available underlying physical infrastructure of the cloud. The host machines or worker nodes were limited to approximately 980 GB in total. However, the number of VM instances is not expected to have so much of an effect on the generalization of the results and the conclusions drawn thereof.

Table 3:- Specification of VM Instances in OpenNebula

	Specification of VM Instance Classes in CloudSim			
	Small	Medium	Large	Extra Large
Storage (GB)	1	2	4	8
RAM (MB)	128	512	768	1024
No. of CPU	1	1	1	1
No.of vCPU	1	1	1	1

Table 3 shows the specification of the VM instances that were used for OpenNebula cloud platform. This table shows that a small VM instance is 1 GB, medium VM instance is 2 GB, a large VM instance if 4 GB and an extra large VM instance is 8 GB. The RAM is 128, 512, 768 and 1024 MB for small, medium, large and extra large VM instances respectively. Regardless of the size, each instance is allocated one CPU and one vCPU. Figure 4 shows the results of placing the actual VMs at CTL of zero on an OpenNebula cloud. At CTL of zero VMs can be placed anywhere as long as there is sufficient storage capacity. In this experiment, the small VM instances were placed between 18 - 34 seconds. The medium VM instances took a turnaround time between 32 - 57 seconds. Large instances took between 42 – 70 seconds and extra-large VMs were places within 48 - 82 seconds. The results further show that the placement time increases with the size of the VMs. The results in experiment two seems to defy odds in that the time it takes to place 10 large VM instances is almost the same as that of placing 25 extra-large VMs.

We can postulate that it might be that the large VM instances were placed in different physical host and most of the large VM instances were placed on the same physical host. However, this is strange and it becomes the odd one out, more so when considering the next results. Furthermore, the results in this experiment shows that decreasing the number of VM instances also decreases the time it takes to do the actual placement. This is demonstrated in experiment two and three where the number of VM medium instances are reduce from 15 to 12 and the placement turnaround time also gets reduced from approximately 40 to 38 seconds. Moreover, there is a somehow peculiar result between experiment two and three for the small VM instances. For some strange and unknown reason, the number of VMs that are being placed are the same, i.e. six but the result is different. Such deviations require further study.

Figure 5 shows the results of placing VMs at CTL zero to four. The small VM instances were placed between 18 - 35 seconds. The medium VM instances took a turnaround time between 31 - 45 seconds. Large instances took between 42 – 66 seconds and extra-large VMs were placed within 48 - 68 seconds. As demonstrated in figure 10.7, the results in figure 10.8 also show that the placement time increases with the size of the VMs. Moreover, the results also reflects a linear increase with the increase in CTL. There seems to be a steady increase as the CTL increases.

Figure 6 also shows that small VM instances took between 25 - 55 seconds to be placed. Medium VM instances took a turnaround time between 40 - 59 seconds. Large instances took between 47 – 66 seconds and extra-large VMs were placed within 45 - 79 seconds. These results reflect an insignificant increase between CTL zero and one; for medium and large VM instances. However, for small and extra-large instances the difference is more significant. At CTL two, the difference in turnaround time between medium and large seems to be very small, with medium instances taking more time than large instances. This is another odd result which requires further analysis to correctly determine its cause.

Figure 7 shows that small VM instances are placed within a timeframe of 24 - 43 seconds. Medium VM instances are placed from a minimum of 24 to a maximum of 62 seconds. Large VM instances took 60 seconds minimum and 73 seconds maximum. Lastly, extra-large VM instances took 69 – 87 seconds. The results of figure 7 reflect a relatively linear increase right across all VM instance classes (i.e. small, medium, large and extra-large) and the different CTLs.

Figure 8 shows that small VM instances are placed within a timeframe of 34 - 53 seconds. Medium VM instances are placed from a minimum of 57 to a maximum of 80 seconds. Large VM instances took 70 seconds minimum and 89 seconds maximum. Lastly, extra-large VM instances took 82 – 108 seconds. The results of figure 8 are similar to those of figure 10.9. They both reflect a relatively linear increase right across all VM instance classes and the different CTLs.

In summary and without any loss of generality, we can deduce that size of the VM instances has a direct effect on the time it takes to do the actual placement. From the results, it can also be deduced that the high the CTL the longer it takes to do the actual placement. Furthermore, when doing the actual placements of VMs, it was noted that the VM instance templates were created on the front-node and had to be transferred via the network for deployment to the respective physical nodes. This does add some latency to the response time. Hence, it would be better if the VM instances were created on the nodes where they are to be placed. In comparison, the results of an actual VM placement on an OpenNebula cloud environment to that of CloudSim (simulated environment) clearly shows that a simulated environment produces result with a significant error margin. For example, CloudSim placed 150 instances of extra large VMs in less than 30 seconds, yet doing the actual placement on OpenNebula with less than half instances took more than 80 seconds. Having done the experiments on an OpenNebula cloud, it would be interesting to compare our results to other cloud computing environments e.g. CloudStack, Eucalyptus, and OpenStack. However, such a comparison is left as part of future work. The next section concludes the paper and make recommendations for future work.

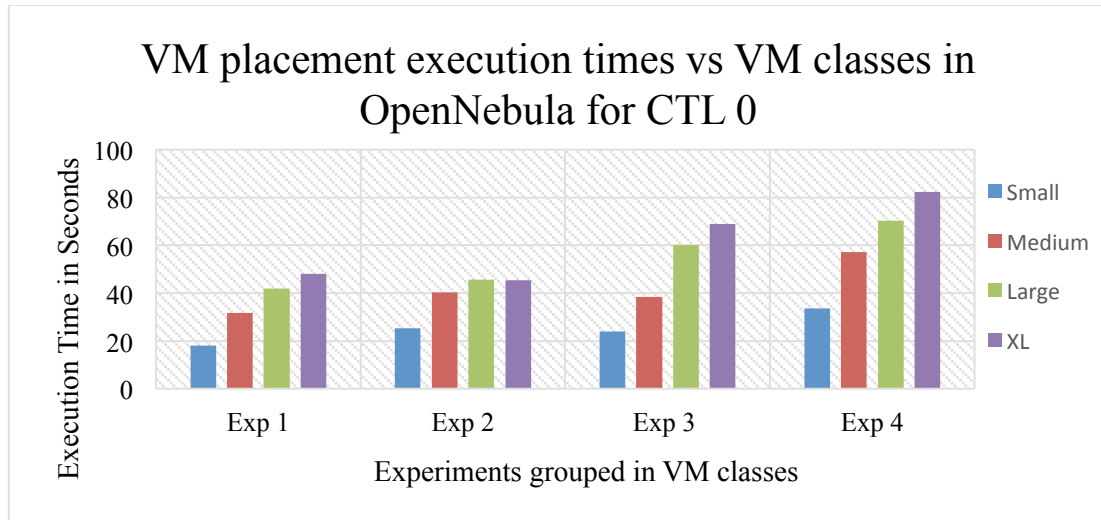


Figure 4:- VM Placement Performance Across the Different VM classes in OpenNebula at CTL 0

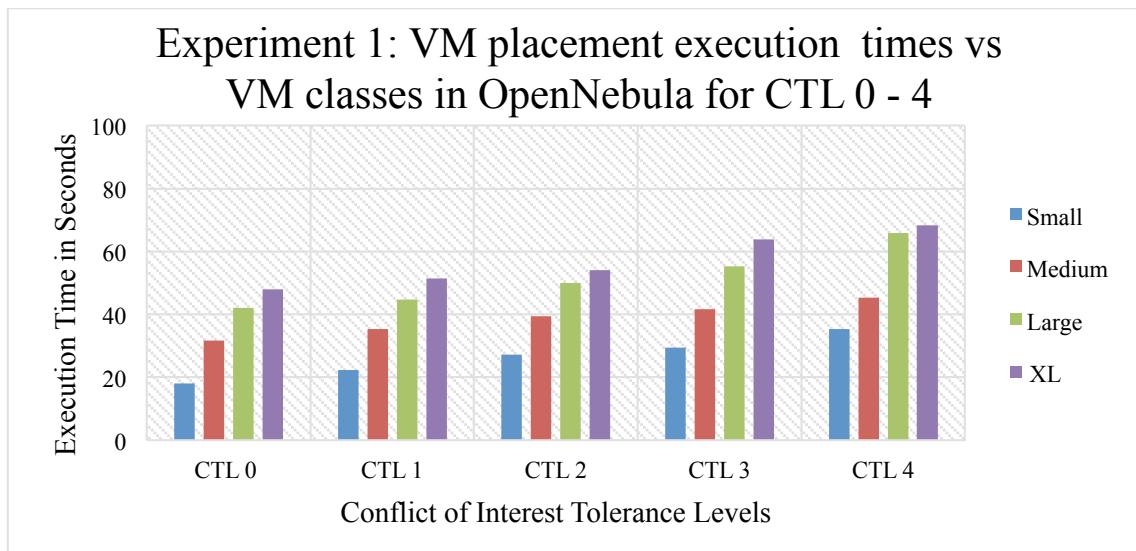


Figure 5:- Experiment 1 - VM placement performance across the different VM classes in OpenNebula at CTL 0 – 4

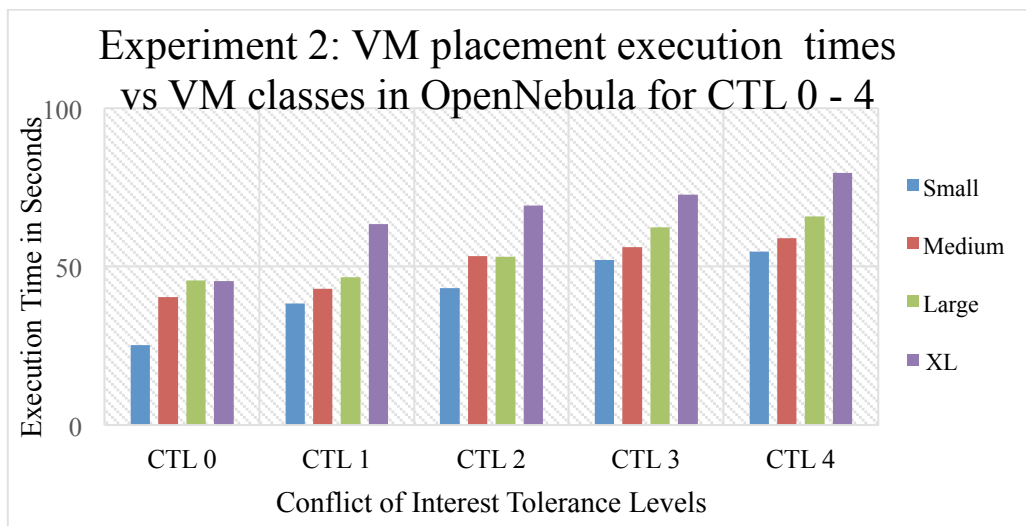


Figure 6:- Experiment 2 - VM placement performance across the different VM classes in OpenNebula at CTL 0 – 4

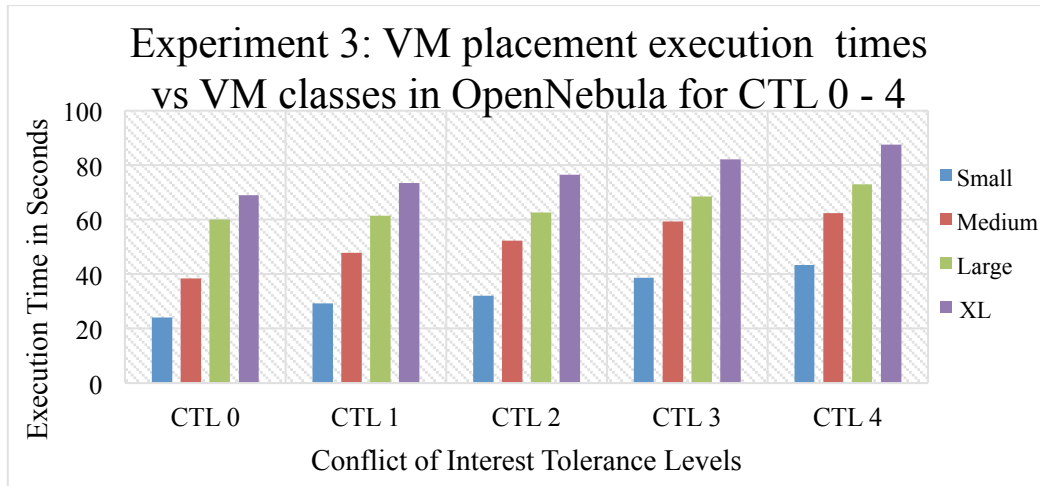


Figure 7:- Experiment 3 - VM placement performance across the different VM classes in OpenNebula at CTL 0 – 4

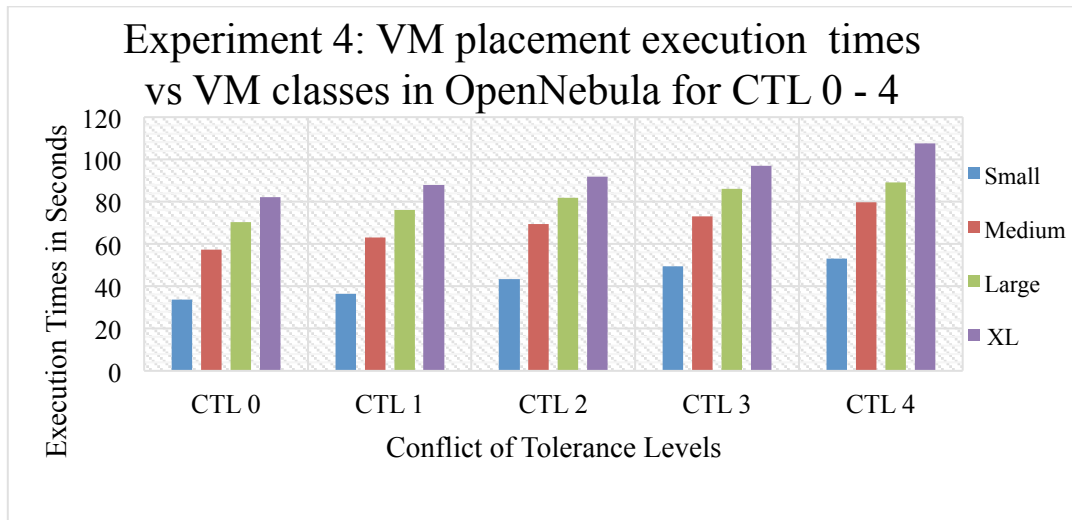


Figure 8:- Experiment 4 - VM Placement Performance Across the Different VM classes in OpenNebula at CTL 0 - 4

5.0 Conclusion

Virtualization raises a VM placement vulnerability for cloud computing. This vulnerability allows a malicious tenant to use an inter-VM attack to leak confidential business data from other co-resident tenants. This paper illustrated the unsuitability of the famous Brewer and Nash's CWM to solve the VM placement problem and contain inter-VM attacks. However, it borrows and uses the CoI concepts from the CWM. Furthermore, it adds different degrees of conflict to propose the CBAC4C model that is well suited for VM placement on the cloud. The aim of introducing different degrees of conflict is to help tenants and CSPs make informed and well-calculated VM placement decisions that factor in their security profile – balanced against cost constraints and risk appetite. The results show that VM placements on the cloud are directly affected by the CTLs, the number of competing tenants and the population size of already placed VMs. The results show that the higher the CTL, the higher the cost of placement.

Future work will consist of a more detailed empirical analysis of our proposed CBAC4C on other cloud infrastructures like Eucalyptus, OpenStack etc.

References