Mitigating Multi-Tenancy Risks in IaaS Cloud Through Constraints-Driven Virtual Resource Scheduling

Khalid Bijon
Institute for Cyber Security
Univ of Texas at San Antonio
khalid.bijon@utsa.edu

Ram Krishnan
Institute for Cyber Security
Univ of Texas at San Antonio
ram.krishnan@utsa.edu

Ravi Sandhu
Institute for Cyber Security
Univ of Texas at San Antonio
ravi.sandhu@utsa.edu

ABSTRACT

A major concern in the adoption of cloud infrastructure-as-a-service (IaaS) arises from multi-tenancy, where multiple tenants share the underlying physical infrastructure operated by a cloud service provider. A tenant could be an enterprise in the context of a public cloud or a department within an enterprise in the context of a private cloud. Enabled by virtualization technology, the service provider is able to minimize cost by providing virtualized hardware resources such as virtual machines, virtual storage and virtual networks, as a service to multiple tenants where, for instance, a tenant's virtual machine may be hosted in the same physical server as that of many other tenants. It is well-known that separation of execution environment provided by the hypervisors that enable virtualization technology has many limitations. In addition to inadvertent misconfigurations, a number of attacks have been demonstrated that allow unauthorized information flow between virtual machines hosted by a hypervisor on a given physical server. In this paper, we present attribute-based constraints specification and enforcement as a mechanism to mitigate such multi-tenancy risks that arise in cloud IaaS. We represent relevant properties of virtual resources (e.g., virtual machines, virtual networks, etc.) as their attributes. Conflicting attribute values are specified by the tenant or by the cloud IaaS system as appropriate. The goal is to schedule virtual resources on physical resources in a conflict-free manner. The general problem is shown to be NP-complete. We explore practical conflict specifications that can be efficiently enforced. We have implemented a prototype for virtual machine scheduling in OpenStack, a widely-used open-source cloud IaaS software, and evaluated its performance overhead, resource requirements to satisfy conflicts, and resource utilization.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

SACMAT'15, June1–3, 2015, Vienna, Austria.
Copyright © 2015 ACM 978-1-4503-3556-0/15/06 ...$15.00.
DOI: http://dx.doi.org/10.1145/2752952.2752964
.

General Terms

Security

Keywords

Cloud IaaS; Virtual-Resource Scheduling; VM Migration; Multi-Tenancy; Constraint; VM Co-Residency Management

1. INTRODUCTION

Enterprises are increasingly driven by economics and flexibility to utilize computing resources provided by cloud infrastructure-as-a-service (IaaS) [32]. A major impediment to wider adoption of cloud IaaS stems from an enterprise's loss of direct control over their virtual resources in cloud IaaS relative to the customary level of control over physical (or virtual) resources in an enterprise-managed data center [22]. In cloud IaaS, the physical resources in a datacenter are logically arranged by the cloud service provider (CSP) and virtual resources are hosted on those logical collections of physical resources. This is illustrated in figure 1 where a rack, for example, is a collection of a specific set of physical servers and network hosts. Other resources such as physical storage volumes may be associated with those compute hosts in the rack. This is shown as physical resource to physical resource mapping (PR-to-PR) in the figure. The single and double-headed arrows indicate the usual "one-to" and "many-to" mappings respectively. Tenants obtain a number of separate pieces of virtual computing resources (or simply resources), e.g. virtual machines, virtual networks, etc., from the CSP. The cloud IaaS system should have suitable policy specification capabilities so that tenants can dynamically manage and arrange these resources to build a particular computing environment based on their needs. For instance, tenants can systematically control specific virtual hardware (e.g. RAM, disk) assignment to virtual machines based on certain properties (e.g. purpose, running work-load type, sensitivity). This is shown as virtual resource to virtual resource mapping (VR-to-VR) in the figure and several mechanisms have been proposed [10, 11] to manage it.

More significantly, in cloud IaaS, physical hardware is also shared by multiple virtual resources for maximizing utilization and reducing cost. IaaS public or community cloud providers allow multi-tenancy which multiplexes virtual resources of multiple enterprises upon same hardware. This includes co-location of virtual machines from different tenants on a single physical host, sharing physical disk storage, etc. This is illustrated as virtual resource to physical resource mapping (VR-to-PR) in figure 1. This raises many secu-

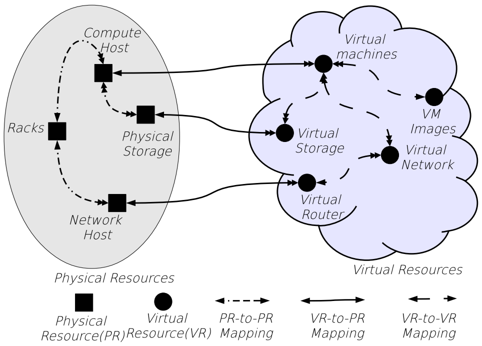


Figure 1: Cloud Resources Mapping Relation

rity and performance considerations for a tenant's workload in the cloud. For instance, a tenant's virtual machines can be attacked by co-located malicious virtual machines of an adversary tenant. Similarly, highly cpu-intensive co-located virtual machines may disrupt each other's expected performance. The work of Ristenpart et al [31, 36, 40, 41] has demonstrated such co-location vulnerabilities in real-world clouds. In particular, they show that preventing targeted co-location of virtual machines from different tenants on the same physical server is unlikely to be successful. Their conclusion is that "the best solution is simply to expose the risk and placement decisions directly to users" (i.e. tenants) [31]. The main objective of this paper is to address this goal where the tenants and the cloud system are able to schedule virtual resources on the physical resources consistent with high-level and fine-grained constraints.

In this respect, even the leading IaaS service providers offer minimal support to their tenants. In particular, tenants have very little influence in how their resources are scheduled. Of course, certain coarse-grained and static preferences for disaster management are supported. For instance, the Amazon Web Services cloud infrastructure is hosted at multiple locations worldwide where a location comprises of multiple geographically isolated datacenters called a 'Region' [1]. Each 'Region' also has multiple, isolated locations known as 'Availability Zones'. As a client, a tenant can at best specify the 'Availability Zone' of its virtual resources and specify backup Availability Zones for a premium. This concerns engineering for fail-safety but does not concern co-location of a tenant's resources with those of others in a given physical server or a rack. This article explores a highly dynamic and fine-grained technique for scheduling virtual resources based on high-level constraints specified by tenants in order to mitigate security threats (several of which are discussed in [5]) due to the co-residency of virtual resources.

We present the design, implementation, and evaluation of an attribute-based constraints specification framework which enables tenants to express several essential properties of their resources as attributes and to specify values of those attributes that conflict for the purpose of co-locating virtual resources on a given physical resource. A constraints enforcement engine schedules virtual resources on the physical resources while respecting the conflicts specified using attributes of those resources. For instance, consider two attributes of a virtual machine: a `tenant` attribute that represents the owning enterprise of that virtual machine, and a `sensitivity` attribute that represents the sensitivity level of the data processed by that virtual machine. An example of a high-level co-location constraint is that virtual machines of different tenants may co-locate in a physical server as long as the `sensitivity` is not 'high'. As we will see, enforcing constraints in general in large-scale systems such as IaaS cloud is computationally inefficient. Moreover, it negatively impacts physical resource requirements and their utilization—directly impacting the bottom line of IaaS CSPs.

The contributions of this paper are summarized below which aligns with its general outline.

- We present a design of an attribute-based framework for specifying co-location constraints of virtual resources scheduled on given physical resources (section 2).

- Given that co-location constraints can drastically affect physical resource utilization, we propose a host optimization process while enforcing constraints (section 3). Note that, host optimization (i.e., optimizing the number of hosts necessary for scheduling the `vms` in a conflict-free manner) is an important requirement for achieving energy-efficient datacenter which is also a major concern for the CSPs for cost optimization [7]. We establish that, in general, the algorithms for host optimization while enforcing such constraints are NP-Complete (section 3). We demonstrate a subset of attribute conflicts that are of practical significance in varied application domains and cloud deployment scenarios (public, private, community, etc.), which can be efficiently enforced in polynomial-time (section 3).

- We develop a prototype of the conflict-free virtual machine scheduling framework in OpenStack [3] and rigorously evaluate the framework on various aspects, e.g., resource requirements, resource utilizations, etc. (section 4).

We analyze issues that arise due to the incremental changes of conflicts over time. A discussion of the security risks in this approach, some related works and conclusion are given in section 6, 7 and 8 respectively.

2. CONFLICT-FREE VIRTUAL RESOURCE SCHEDULING

Intuitively, an attribute captures a property of an entity in the system, expressed as a name:value pair. In the context of cloud IaaS, attributes can represent a virtual machine's owner tenant, sensitivity-level, cpu intensity-level of workloads, etc. For simplicity, we restrict the scope of the paper as follows. We confine our attention to virtual to physical resource mapping in the context of virtual machines and physical compute servers. Then we briefly discuss the possible extension of this approach to other virtual and physical resources. In the rest of the article we refer to physical compute host and virtual machine as `host` and `vm` respectively. We restrict the kind of constraint to "must not co-locate" constraint where the specified conflicts are co-location conflicts that state whether two `vms` can be co-located in the same `host` or not. In this section, we formally define the components of `hosts` allocation for the `vms`, which we refer as `host`-to-`vm` allocation, in the presence of various co-location conflicts. Note that, a `vm` may have multiple attributes each with its own values. Attribute value of a `vm` can be assigned either manually by a user or automatically by the system. For instance, when an enterprise user creates a `vm`, an appropriate value is assigned to the *tenant* attribute of the vm automatically whereas, the user may need to explicitly specify the value for a *sensitivity* attribute based on sensitivity of data processed in that vm. Developing administration models for such attribute assignments is beyond the scope of this paper. We assume that vms are assigned with proper attribute values. For our purpose, the values of an attribute can conflict with each other and the goal is to allow the vms to co-locate in same host only if their assigned attribute-values do not conflict. For general and in-depth understanding about various types of attribute conflicts we suggest to read the articles of Bijon et al [8, 9].

2.1 Scheduling Components Specification

The scheduling components include two sets called HOST and VM that contain the existing hosts and vms respectively. There are attributes of vm that characterize different properties of a vm and are modeled as functions. For each attribute function, there is a set of finite constant values that represents the possible values of that attribute. For our purpose, we assume values of attributes to be atomic.[1] Therefore, for a particular vm, the name of the attribute function maps to one value from the set. For convenience attribute functions are referred to as attributes. Also, values of an attribute can have conflicts with each other and these conflicts are specified in a conflict-set of the attribute. Conflicts are specified on values of each attribute independent of other attributes. Formally these components are defined as follows.

- HOST is the finite set of hosts (physical servers).
- VM is the finite set of vms.
- Each host ∈ HOST has a capacity, represented as a function called W_{HOST}, that maps a host to a value greater than 1.0 to a maximum value of the host capacity[2]. The capacity restricts the number of vms that a host can contain based on the accumulated capacity of the vms. Value of it for a host remains constant unless explicitly modified, e.g., increasing RAM size.
- Similar to the capacity of host, each vm ∈ VM has a capacity represented by a function called W_{VM} where, W_{VM} : VM → k where $0.0 < k \leq 1.0$. Also, capacity of a vm remains constant unless explicitly modified.
- $\mathsf{ATTR}_{\mathsf{VM}}$ is the set of attribute functions of vm.
- For each $att \in \mathsf{ATTR}_{\mathsf{VM}}$, the domain of the function is the VM and the codomain is the values of *att* written as SCOPE_{att} which is a set of atomic values. Formally, *att*: VM $\rightarrow \mathsf{SCOPE}_{att}$, for each $att \in \mathsf{ATTR}_{\mathsf{VM}}$.

The values in SCOPE_{att} of an $att \in \mathsf{ATTR}_{\mathsf{VM}}$ that conflict with each other is specified as a relation called ConSet_{att}. ConSet_{att} is reflexive and symmetric, but not transitive. Hence, each element in ConSet_{att} is an unordered pair. For each $att \in \mathsf{ATTR}_{\mathsf{VM}}$, ConSet_{att} is defined as follows.

- ConSet_{att} is the set of conflicts of the values of each *att* $\in \mathsf{ATTR}_{\mathsf{VM}}$. Formally,
  $\mathsf{ConSet}_{att} \subseteq \{\{\text{x,y}\} \mid \text{x}\neq\text{y and x,y} \in \mathsf{SCOPE}_{att}\}$

Part I in figure 2 shows two attributes, *tenant* and *sensitivity*, and their respective scopes. Some conflicts among values of *tenant* and *sensitivity* attributes are also shown representing conflicts among their values. For instance, $\{\{\text{tnt}_1, \text{tnt}_2\}, \{\text{tnt}_2,\text{tnt}_3\}, \{\text{tnt}_4, \text{tnt}_6\}\}$ in ConSet_{tnt} specifies that vms of tnt_1 and tnt_2, tnt_2 and tnt_3, and tnt_4 and tnt_6 conflict with each other and, hence, cannot be co-located. Also, part IV shows an example of attribute assignment for vms. For instance, for vm1, *tenant*(vm1) = tnt_3 and *sensitivity*(vm1) = high. Also note that the value 0.6 denotes the capacity requirement of that vm. That is, W_{VM}(vm1)=0.6.

2.2 Conflict-Free Host to VM Allocation

Given that the ConSet_{att} specifies conflicting values for an attribute $att \in \mathsf{ATTR}_{\mathsf{VM}}$, the conflict-free host to vm allocation is concerned about allocation of a host to a group of vms that do not conflict with each other. There are 4 steps in this process as illustrated in figure 2. Step 1 is to *partition* the values of each attribute (i.e., SCOPE_{att} of an *att* $\in \mathsf{ATTR}_{\mathsf{VM}}$), into a family of subsets where the elements in each subset do not conflict with each other. We refer to such *partition* as "Conflict-Free Partition of Attribute-Values".

Definition 1. **(Conflict-Free Partition of Attribute-Values)** *A* conflict-free partition of attribute-values *of each att* $\in \mathsf{ATTR}_{\mathsf{VM}}$ *is specified as* $\mathsf{PARTITION}_{att}$ *that partitions the values in* SCOPE_{att} *where the values of each element in* $\mathsf{PARTITION}_{att}$ *do not conflict with each other, i.e., for each* $x \in \mathsf{PARTITION}_{att}$ *and for each* $y \in \mathsf{ConSet}_{att}$, $|x \cap y| \leq 1$

We can state that, for an attribute *att*, a $\mathsf{PARTITION}_{att}$ partitions SCOPE_{att} where (1) $\mathsf{PARTITION}_{att}$ does not contain $\emptyset$, (2) elements in $\mathsf{PARTITION}_{att}$ are pairwise disjoint, (3) the union of the elements in $\mathsf{PARTITION}_{att}$ is SCOPE_{att}, and (4) the values in a set-element of $\mathsf{PARTITION}_{att}$ do not conflict with each other, i.e. no more than one value from that set-element belongs to the same element in ConSet_{att}.

Part II in figure 2 shows examples of conflict-free partitions, $\mathsf{Partition}_{tenant}$ and $\mathsf{Partition}_{sensitivity}$, for ConSet_{tenant} and $\mathsf{ConSet}_{sensitivity}$ given in part I. For example, $\{\text{tnt}_1, \text{tnt}_3, \text{tnt}_6\}$ in $\mathsf{Partition}_{tenant}$ means these values do not conflict with each other. Note that, there can be multiple candidate $\mathsf{PARTITION}_{att}$ for a given ConSet_{att} of an attribute *att* $\in \mathsf{ATTR}_{\mathsf{VM}}$. Section 3 shows that the selection of an appropriate $\mathsf{PARTITION}_{att}$ is important for the host optimization.

Step 2 combines the *conflict-free partitions of attribute-values* of all attributes. We define a *conflict-free segment* that consists one element of $\mathsf{PARTITION}_{att}$ of each attribute $att \in \mathsf{ATTR}_{\mathsf{VM}}$. We will see later that vms, mapped to a conflict-free segment, do not conflict with that of others, hence, can co-locate. Note that a vm can get any value from the scope of an attribute. Therefore, conflict-free segments should be generated in such a way so that it can map all possible assigned values to the attributes of the vms. A cartesian product of the $\mathsf{PARTITION}_{att}$ for all $att \in \mathsf{ATTR}_{\mathsf{VM}}$ generate all possible segments of conflict-free values of the attributes that can a vm based on its assigned attribute values.

[1] An example of an atomic attribute is *sensitivity* where the values are high, medium and low. A vm can only get one of the three values for *sensitivity*. However, some cases might require set-valued attributes for which a vm may take multiple values. For our purpose, we only consider atomic attributes, however, it can easily extend to set-valued one.

[2] Multi-dimensional weights of a host, e.g., RAM, CPU, can be reduced to one single normalized weight. In OpenStack [3], hosts are mapped to a single weight which is calculated by the *weighted_sum* method that takes weighted average of different metrics of a host e.g., RAM, workload.

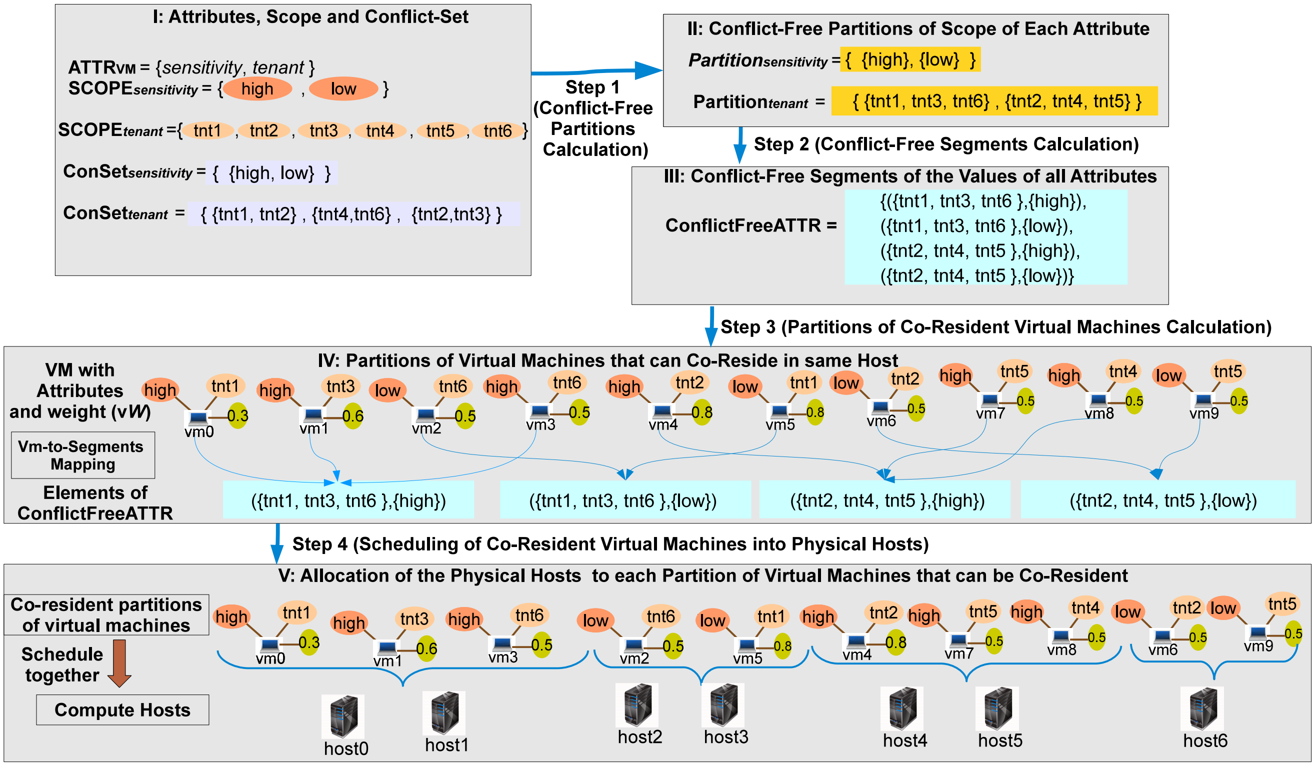

I: Attributes, Scope and Conflict-Set
ATTRVM = {sensitivity, tenant }
SCOPEsensitivity = { high , low }
SCOPEtenant ={ tnt1 , tnt2 , tnt3 , tnt4 , tnt5 , tnt6 }
ConSetsensitivity = { {high, low} }
ConSettenant = { {tnt1, tnt2} , {tnt4,tnt6} , {tnt2,tnt3} }
Step 1 (Conflict-Free Partitions Calculation)
II: Conflict-Free Partitions of Scope of Each Attribute
Partitionsensitivity ={ {high}, {low} }
Partitiontenant = { {tnt1, tnt3, tnt6} , {tnt2, tnt4, tnt5} }
Step 2 (Conflict-Free Segments Calculation)
III: Conflict-Free Segments of the Values of all Attributes
ConflictFreeATTR =
{({tnt1, tnt3, tnt6 },{high}),
({tnt1, tnt3, tnt6 },{low}),
({tnt2, tnt4, tnt5 },{high}),
({tnt2, tnt4, tnt5 },{low})}
Step 3 (Partitions of Co-Resident Virtual Machines Calculation)
IV: Partitions of Virtual Machines that can Co-Reside in same Host
VM with Attributes and weight (vW)
high tnt1 vm0 0.3
high tnt3 vm1 0.6
low tnt6 vm2 0.5
high tnt6 vm3 0.5
high tnt2 vm4 0.8
low tnt1 vm5 0.8
low tnt2 vm6 0.5
high tnt5 vm7 0.5
high tnt4 vm8 0.5
low tnt5 vm9 0.5
Vm-to-Segments Mapping
Elements of ConflictFreeATTR
({tnt1, tnt3, tnt6 },{high})
({tnt1, tnt3, tnt6 },{low})
({tnt2, tnt4, tnt5 },{high})
({tnt2, tnt4, tnt5 },{low})
Step 4 (Scheduling of Co-Resident Virtual Machines into Physical Hosts)
V: Allocation of the Physical Hosts to each Partition of Virtual Machines that can be Co-Resident
Co-resident partitions of virtual machines
Schedule together
Compute Hosts
high tnt1 vm0 0.3
high tnt3 vm1 0.6
high tnt6 vm3 0.5
low tnt6 vm2 0.5
low tnt1 vm5 0.8
high tnt2 vm4 0.8
high tnt5 vm7 0.5
high tnt4 vm8 0.5
low tnt2 vm6 0.5
low tnt5 vm9 0.5
host0
host1
host2
host3
host4
host5
host6

Figure 2: Conflict-Free vm-host Allocation

Definition 2. **(Conflict-Free Segments of the Values of Attributes)** *The* conflict-free segments of the values of attributes *is a set, called ConflictFreeATTR, of n-tuples where $n = |ATTR_{VM}|$ and each tuple is a result of the cartesian product of $PARTITION_{att}$ of all $att \in ATTR_{VM}$, i.e.,* $ConflictFreeATTR = \prod_{att \in ATTR_{VM}} PARTITION_{att}$

Each element $conFval \in$ ConflictFreeATTR is an ordered pair which is written as $\langle X_{att_1}, ..., X_{att_n} \rangle$ where $\{att_1, ..., att_n\} =$ ATTR$_{VM}$ and $X_{att_i} \in$ PARTITION$_{att_i}$. We assume that elements of each $conFval \in$ ConflictFreeATTR can be accessed by the notation $conFval[att]$ for each $att \in$ ATTR$_{VM}$.

Part III in figure 2 shows an example ConflictFreeATTR which is produced from the Cartesian product of conflict-free partitions Partition$_{tenant}$ and Partition$_{sensitivity}$. A tuple ({tnt$_1$, tnt$_3$, tnt$_6$},{high}) is an element in ConflictFreeATTR since {tnt$_1$, tnt$_3$, tnt$_6$} and {high} are members of Partition$_{tenant}$ and Partition$_{sensitivity}$ respectively.

Step 3 partitions the set VM such that vms of each element of the partition can be co-located. This is achieved by partitioning VM in a way such that each element of the partition can be mapped to an element of ConflictFreeATTR.

Definition 3. **(Co-Resident Partition of VM)** *The* Co-Resident Partition of VM, *specified as CoResidentVMGrp, is a partition of VM where the assigned values to att ∈ ATTR$_{VM}$ of all* vms *in an element of the partition map to the same segment in ConflictFreeATTR, i.e.,*
for all $X \in CoResidentVMGrp$ and for all $vm_i \neq vm_j \in X$,
$$\bigvee_{conFval \in ConflictFreeATTR} SetResidence(vm_i, vm_j, conFval, ATTR_{VM}))$$
where, $SetResidence(vm_i, vm_j, conFval, ATTR_{VM}) =$
$$\bigwedge_{att \in ATTR_{VM}} (att(vm_i) \in conFval[att] \wedge att(vm_j) \in conFval[att])$$

CoResidentVMGrp partitions VM if vms in an element of CoResidentVMGrp are assigned to the values, for all $att \in$ ATTR$_{VM}$, that belong to the same element in ConflictFreeATTR.

Part IV in figure 2 shows an example of CoResidentVMGrp calculation of 10 vms where vms are mapped to different elements of ConflictFreeATTR based on their attributes. For instance, vm1 is mapped to the segment ({tnt$_1$, tnt$_3$, tnt$_6$},{high}) since it is assigned with 'tnt$_3$' and 'high' for *tenant* and *sensitivity* attributes. Also, vm1 and vm3 belong to the same partition of CoResidentVMGrp since they are both mapped to the segment ({tnt$_1$, tnt$_3$, tnt$_6$}, {high}).

Finally, step 4 allocates hosts for the vms of each partition in CoResidentVMGrp. A host cannot contain vms from multiple partitions of CoResidentVMGrp. Also, combined capacity of the allocated vms must satisfy the capacity (W_{HOST}) of the host. Therefore, for each partition of vms in CoResidentVMGrp, multiple hosts might be required depending on the combined weight of the vms in that partition.

Definition 4. **(Conflict-Free Host to VM Allocation)** *Given VM, HOST, ATTR$_{VM}$, CoResidentVMGrp, W_{HOST} and W_{VM}, the* Conflict-Free Host to VM Allocation *is a mapping function called allocate that finds a set of* hosts*, $HOST' \subseteq HOST$, to allocate all* vm *∈ VM where the* vms *that reside in a* host *form a subset of an element of CoResidentVMGrp such that their combined weight does not exceed the weight of* host*, i.e., $allocate : HOST' \hookrightarrow \mathcal{P}(VM)$ where, if $chost \in HOST'$ and $allocate(chost) = lvm$, then,*
$$lvm \subseteq VM \wedge \bigvee_{x \in CoResidentVMGrp} lvm \subseteq x \wedge (\sum_{vm \in lvm} W_{VM}(vm)) \leq W_{HOST}(cs)$$

Part V in figure 2 shows an example of Conflict-Free Host to VM Allocation where the total number of vms is 10 and they are partitioned into 4 co-resident sets. Note that, here, Host0 and Host1 are allocated to one co-resident partition of vms containing {vm0, vm1, vm3} since their combined weight is more than the weight of a single host.

2.3 Conflict-Free Scheduling of Other Virtual Resources to Physical Resources

The process of sections 2.1 and 2.2 can also apply for the scheduling of other virtual resources (shown in figure 1) with the following modifications.

In physical storage to virtual storage allocation, two sets VM and HOST, defined in section 2.1, are substituted by sets VS and PS that specifies virtual storage volumes and physical volumes in the system respectively. Similar to the capacity functions of vm and host, two functions can be defined for virtual and physical resources that can map their respective capacities where the capacity can be a single metric calculated by weighted sum of different properties of a storage system. Such properties include size, storage i/o speed, etc. Now, similar to the $\mathsf{ATTR}_{\mathsf{VM}}$, a set can represent the attributes of the virtual storage volumes. Also, ConSet_{att} and definition 1-4 can be modified accordingly for the physical storage to virtual storage allocations.

A similar approach can be followed to derive the network host to virtual router allocation. Here, two sets called NH and VR can specify network hosts and virtual routers in the system respectively. Now the capacity could be the limit of network bandwidth of a network host and the bandwidth of a virtual router. One motivation of scheduling virtual router across different network hosts is for load-balancing of the network traffic and ensuring availability. Here, similar to the virtual machines, necessary attributes of the virtual routers can be generated. ConSet_{att} and definition 1-4 can be modified for network host to virtual router allocations.

3. OPTIMIZATION PROBLEM DEFINITION AND SOLUTION ANALYSIS

In this system, the specified conflicts restrict certain vms from co-locating in the same host. Hence, some hosts cannot schedule vms that conflict with currently scheduled vms in these hosts, despite having the required capacity. That increases the required number of hosts than a system without conflicts. Hence, it is desirable to schedule vms in a way that minimizes the number of hosts while satisfying the conflicts leading to an optimization problem.

Definition 5. (Host Optimization Problem) *The* Host optimization problem *seeks to minimize the number of* hosts *in the mapping, allocate : HOST′ $\hookrightarrow \mathcal{P}$(VM), specified in* Conflict-Free Host to VM Allocation (Definition 4).

This section investigates algorithms for definition 1 through 4 in order to solve the Host Optimization Problem.

3.1 MIN_PARTITION: Minimum Conflict-Free Partitions of Attribute-Values

More than one $\mathsf{PARTITION}_{att}$ can be generated for a given ConSet_{att}. In figure 2, for the given ConSet_{tenant}, candidate $\mathsf{Partition}_{tenant}$ sets could be {{tnt1, tnt3}, {tnt2, tnt6}, {tnt4, tnt5}} and {{tnt1,tnt3,tnt6},{tnt2,tnt4,tnt5}} with 3 and 2 elements in the sets respectively. Here, each element of a $\mathsf{PARTITION}_{att}$ contains the conflict-free attribute-values of *att*. Number of elements in $\mathsf{PARTITION}_{att}$ affects the total number of conflict-free segments (definition 2) where the vms mapped to same conflict-free segment can co-exist. A partition, with minimum number of elements, reduces the number of conflict-free segments. It also reduces the elements in CoResidentVMGrp that also minimizes the required number of hosts. We call such a partition as MIN_PARTITION.

Finding a MIN_PARTITION is similar to the graph-coloring problem that partitions the vertices of a graph G(V,E) into minimum color classes so that no two adjacent vertices, such as {v1,v2} ∈ E, fall in the same class. Graph-coloring problem is NP-Complete given that graph coloring *decision* problem, called k-coloring, is NP-Complete, which states that given a graph G(V, E) and a positive integer k $\leq$ |V|, can the vertices in V be colored by k different colors?

We show that MIN_PARTITION is NP-Complete by showing that the MIN_PARTITION *decision* problem, which we refer to as K_PARTITION, is NP-Complete. The problem states that given SCOPE_{att} and ConSet_{att} of an $att \in \mathsf{ATTR}_{\mathsf{VM}}$, and a positive integer k $\leq |\mathsf{SCOPE}_{att}|$, can the values in SCOPE_{att} be partitioned into k sets?

Theorem 1. *K_PARTITION is NP-Complete.*

Proof. We prove that K_PARTITION is NP-Complete by polynomial-time reduction of k-coloring to K_PARTITION.

An *instance* of k-coloring is a graph G(V, E) and an integer k. We construct $\mathsf{SCOPE}_{att} \leftarrow$ V and $\mathsf{ConSet}_{att} \leftarrow$ E and feed SCOPE_{att}, ConSet_{att}, and k to K_PARTITION. The complexity of this conversion is |V| × |E|.

Now we show that an *yes instance* of k-coloring maps to an *yes instance* of K_PARTITION and vice versa.

$\Longrightarrow$ Assume G is an *yes instance* of k-coloring and there exists a set of colors C of size k in G. Thus, for all u ∈ V, color(u) ∈ C and for any u, v ∈ V, color(u)=color(v) only if {u, v} ∉ E. Also, for all u ∈ SCOPE_{att}, u belongs to cfs ∈ CFS where #CFS is k, and for any u, v ∈ SCOPE_{att}, u, v belongs to the same cfs ∈ CFS, if {u, v} ∉ ConSet_{att}. Thus, G is an *yes instance* of K_PARTITION.

$\Longleftarrow$ Assume SCOPE_{att}, ConSet_{att} is an *yes* instance of K_PARTITION and there exists a family of CFS of size k. Thus, for all u ∈ SCOPE_{att}, u belongs to a cfs ∈ CFS, and for any u, v ∈ SCOPE_{att}, u, v belongs to the same cfs ∈ CFS, if {u, v} ∉ ConSet_{att}. Thus, the vertices in same cfs ∈ CFS can be colored by the same color and there will be k number of colors to color all the vertices in G. Thus, G is an *yes instance* of k-coloring.

Thus, K_PARTITION is NP-Complete. □

Therefore, MIN_PARTITION is also NP-Complete. However, there are a number of *approximate* graph-coloring algorithms that can be applied to MIN_PARTITION. The algorithms are approximate in the sense that they may not provide the minimum size of $\mathsf{PARTITION}_{att}$, i.e., MIN_PARTITION may not be optimal. This is useful, although not optimal, because the conflicts are still satisfied. In appendix A, we discuss approximate algorithms for graph-coloring and their applications to MIN_PARTITION. We also develop an exact algorithm, shown in algorithm 1, based on backtracking. The complexity of this algorithm is NP since it is an adaptation of the general backtracking algorithm for the graph-coloring [6]. However, for attributes whose size of the scope is small enough (e.g. *sensitivity*), the algorithm computes the partition relatively fast. In algorithm 1, the MAKE_PARTITION procedure is called with scope SCOPE_{att} of an attribute $att \in \mathsf{ATTR}_{\mathsf{VM}}$, ConSet_{att}, and a partition

Algorithm 1 Conflict-Free Partition using Backtracking

```
procedure CHECK_VALIDITY(attval, $\mathsf{ConSet}_{att}$, CSet)
    for all $\text{attval}_i \in$ CSet do
        if {attval, $\text{attval}_i$} $\in \mathsf{ConSet}_{att}$ then
            Return False
        end if
    end for
    Return True
end procedure
procedure MAKE_PARTITION($\mathsf{SCOPE}_{att}$, $\mathsf{ConSet}_{att}$, $\mathsf{PARTITION}^k_{att}$)
    if attval $\in \mathsf{SCOPE}_{att}$ then
        for all par $\in \mathsf{PARTITION}^k_{att}$ do
            if CHECK_VALIDITY(attval, $\mathsf{ConSet}_{att}$, par) then
                par = par $\cup$ attval
                if MAKE_PARTITION($\mathsf{SCOPE}_{att}$-{par},
                $\mathsf{ConSet}_{att}$,$\mathsf{PARTITION}^k_{att}$) then
                    Return True
                end if
                par = par $-$ attval
            end if
        end for
    end if
    Return False
end procedure
```

$\mathsf{PARTITION}^k_{att}$ that can contain k elements. It uses a recursive backtracking algorithm that tries all possible combinations of k partitions and returns true if there is a valid conflict-free k partition of a given ConSet_{att}. Before adding an attribute value to a partition, MAKE_PARTITION calls CHECK_VALIDITY that verifies if the attribute value to be added is indeed free of conflict with respect to ConSet_{att}. In section 4, we analyze the performance of this algorithm for various sizes of attribute scopes and conflict sets.

Certain graphs such as perfect graphs have polynomial graph-coloring solutions. We identify that certain restricted versions of attribute conflict specification generates such graphs. For example, like in a Chinese-Wall policy, an organization can have a conflict-of-interest with certain other organizations. For instance, all banking tenants of a *CSP* may have a conflict-of-interest with each other. Similarly, all the oil-company tenants may conflict. A *CSP* can generate an attribute called *tenant* that represents a particular tenant name in the system, e.g, bank-of-america, and the values of *tenant* can be categorized into mutually disjoint conflict-of-interest classes. The conflict-set generates disjoint cliques of attribute values which can be solved in polynomial-time [18]. Appendix B discusses several such restricted conflicts.

3.2 ConflictFreeATTR Generation

This is a trivial algorithm that calculates the values of ConflictFreeATTR specified in definition 2. The algorithm takes as input $\mathsf{PARTITION}_{att}$ for all $att \in \mathsf{ATTR}_{\mathsf{VM}}$, and returns ConflictFreeATTR which is a Cartesian product of $\mathsf{PARTITION}_{att}$ for all att. It also stores the calculated ordered tuples in ConflictFreeATTR. The complexity is $O(n \times m)$ where n and m are the size of $\mathsf{ATTR}_{\mathsf{VM}}$ and $\mathsf{PARTITION}_{att}$.

3.3 Co-Resident VM Partitions Generation

This algorithm takes ConflictFreeATTR and VM sets as input, creates a family of sets, called CoResidentVMGrp (definition 3), where each set contains a subset of vms that can co-reside. The number of sets in CoResidentVMGrp is equal to the number of elements in ConflictFreeATTR, where the algorithm maps an element of ConflictFreeATTR to an element in CoResidentVMGrp and the mapping is one-to-one and onto. The vms that map to the same element in ConflictFreeATTR belong to the same partition. The complexity of this algorithm is $O(\mathsf{VM} \times \mathsf{ConflictFreeATTR} \times \mathsf{ATTR}_{\mathsf{VM}})$. This algorithm works for both *offline* and *online* versions of VM scheduling. In offline, the total number of vms is fixed and are given before the algorithm runs. In *online*, the scheduling request for a vm arrives one at a time. For both versions, the algorithm takes one vm and maps it to an element in ConflictFreeATTR and adds the vm to a corresponding element in CoResidentVMGrp.

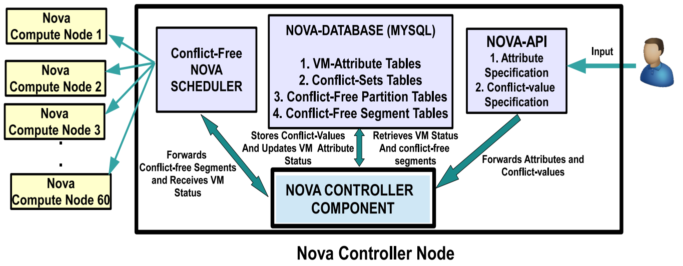

Nova Compute Node 1
Nova Compute Node 2
Nova Compute Node 3
Nova Compute Node 60
Conflict-Free NOVA SCHEDULER
NOVA-DATABASE (MYSQL)
1. VM-Attribute Tables
2. Conflict-Sets Tables
3. Conflict-Free Partition Tables
4. Conflict-Free Segment Tables
NOVA-API
1. Attribute Specification
2. Conflict-value Specification
Input
Forwards Conflict-free Segments and Receives VM Status
Stores Conflict-Values And Updates VM Attribute Status
Retrieves VM Status And conflict-free segments
Forwards Attributes and Conflict-values
NOVA CONTROLLER COMPONENT
Nova Controller Node

Figure 3: Experimental Setup in OpenStack

3.4 Scheduling VMs to Hosts

This algorithm takes CoResidentVMGrp, and schedules the vms that belong to each element in CoResidentVMGrp, together in one or more hosts. For vms of each element in CoResidentVMGrp, this process might need one or more hosts based on the combined capacity of the vms. If the total capacity exceeds the capacity of a single host then it will need multiple hosts. This scheduling problem is similar to the bin-packing [16] problem which is NP-Hard. However, there are a number of known heuristic approaches that can be applied here [30]. The scheduling of vms in an optimal way based on capacity is orthogonal to MIN_PARTITION since MIN_PARTITION is solved before this scheduling begins.

4. IMPLEMENTATION AND EVALUATION

We implement and evaluate our conflict-free vm to host scheduling framework. Since our work concerns scheduling, to conduct realistic experimentation, we need exclusive access to a large-scale cloud infrastructure with 100s of physical hosts to meaningfully study resource requirements and its utilization. First, we setup an IaaS cloud environment using a set of 5 physical machines (each of them is a Dell-R710 with 16 cores, 2.53 GHz and 98GB RAM). We now treat each of the vms that this cloud provides as a physical host. These vms are configured with 4 cores and 3 GB of RAM. We now create a DevStack-based cloud framework [2], a quick installation of OpenStack ideal for experimentation, using those vms as physical hosts to create a virtual cloud for the purpose of experimentation. Now, we create the second-level of vms to get a virtual IaaS cloud and the configuration of these vms are varied based on the experiment we perform.

We implemented our host-to-vm scheduling on the testbed described above. Figure 3 illustrates our experiment setup. In OpenStack, the component that takes care of vm management and scheduling is the Nova service. We created a cloud cluster with 61 hosts where one of them is the Nova controller node and another 60 are the Nova compute nodes.

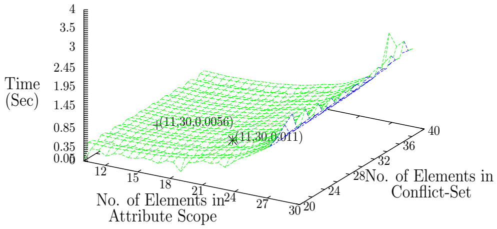


Figure 4: Required Time for Small Scope and Conflict-Set

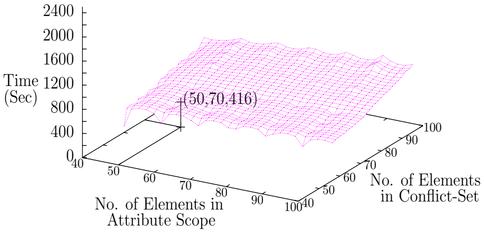


Figure 5: Required Time for Large Scope and Conflict-Set

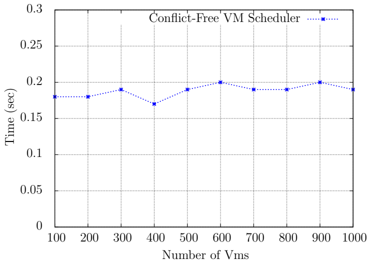


Figure 6: Latency for Conflict-free Scheduling

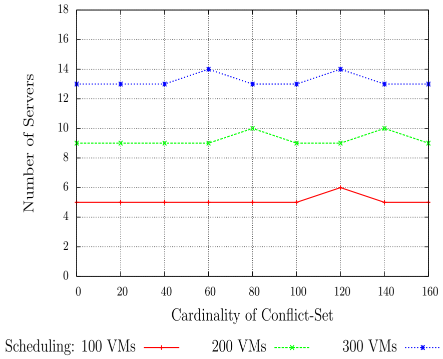


Figure 7: Required Number of `hosts` for Varying Number of Elements in Conflict-Set

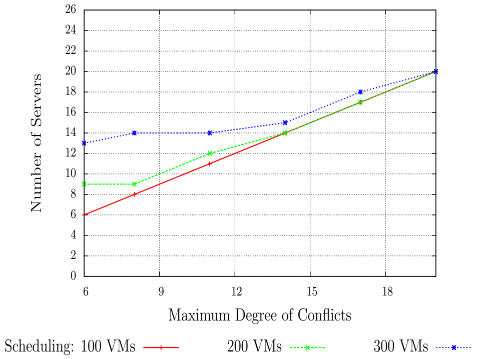


Figure 8: Required Number of `hosts` for Max Degree of Conflicts

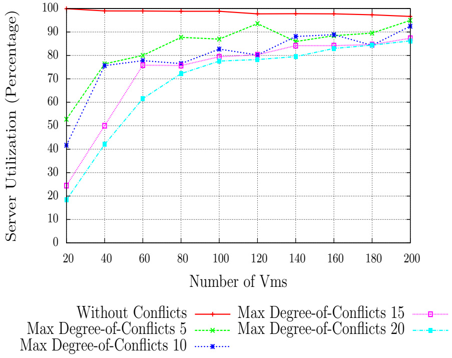


Figure 9: *Host Utilization Overhead*

The Controller node provides main services, e.g. database, message queues, etc., while the compute nodes only contain components such as hypervisor and nova-compute that are required for running `vms`. We deployed the prototype in the nova controller node. Our python-based implementation of conflict specification allows tenant admins to specify attribute conflict values and the ability to store conflict values in nova database (MySQL) (part I in figure 2). Our python based conflict free segments calculation process (steps 1 and 2 in figure 2) has 153 lines of code. Finally, our implementation of conflict-free `host` to `vm` scheduling (steps 3 and 4 in figure 2) has 170 lines of code that maps a `vm` to a conflict-free segment based on conflicting-values and assigned attribute values of the `vm` which are retrieved from the nova database. For the conflict-free segment, designated `hosts` are identified and weighed based on default Nova weighing factors and the `vm` is scheduled to the suitable `host`.

Experiment 1 - *Upper Bound of Algorithm 1.* This experiment analyzes the runtime of Algorithm 1. Since the complexity is in NP, here, we identify the maximum size of scope and conflict set for which required runtime of the algorithm remains feasible. First, we conduct the experiment with a small size of scope of an attribute and respective conflict set. We vary scope size from 10 to 40, and for each scope size, we vary the size of conflict set from 20 to 40. For each scope and a particular size of the conflict set, we randomly create elements in conflict set and execute the algorithm. Figure 4 shows the results where, for a small scope and conflict set, runtime is very low, e.g., 0.011s for a scope and conflict set size of 18 and 30 respectively. However, for bigger scope and conflict set sizes, it increases drastically, e.g, for scope size 30 and conflict set size 35 it becomes approximately 4s. We also conduct the same experiment for large scope and conflict sets where we vary the size from 40 to 100 and 60 to 100 respectively. Figure 5 shows the results where the runtime is very high as expected. For instance, for a scope and conflict set size of 50 and 70 respectively, the execution time is more than 7mins. Note that, a high runtime may be acceptable, since conflict-free partitions are created before starting the scheduling of `vms` and hence it does not impact the scheduler's performance drastically. This experiment gives an estimation of delay the *CSP* might face before scheduling the `vms` if it wants to create conflict-free partitions for a given scope and conflict set size.

Experiment 2 - *Scheduling Latency.* In the second experiment, we analyzed the timing overhead of our conflict-free `host`-to-`vm` scheduler once that conflict-free partitions are calculated by algorithm 1. In figure 6, we study how the amount of time the scheduler takes to schedule a single `vm` varies with increasing number of `vms` that have already been scheduled. A value of 500 in the x-axis, for example, indicates that 499 `vms` have already been scheduled and the corresponding value in the y-axis (0.19s) indicates the time to schedule one new `vm`. The attribute values of the pre-scheduled `vms` were randomly assigned. The scheduler takes a fairly fixed amount of time to schedule a single `vm` regardless of the number of conflict-free pre-scheduled `vms`.

Experiment 3 - *Required Number of Hosts.* Our third experiment concerns the impact of satisfying conflicts on the resource requirements. In our case, the conflict set of a given attribute can be varied in two significant ways to evaluate the number of physical hosts that are necessary. In figure 7, we vary the number of elements in the conflict set while fixing the maximum degree of conflict to a constant value. The highest number of values that conflict with each other in the conflict set is referred to as the maximum degree of conflict for that conflict set. In figure 7, we fix the maximum degree to 2. In figure 8, we vary the maximum degree of conflicts with a fixed attribute scope. Given the server memory capacity to be 3 GB, the `vm` capacity is varied between 512 MB and 1024 MB. The experiment confirms our intuition that that the maximum degree of conflict dominates the server requirement to schedule `vms`. Note that minor spikes and drops (for example between 100 and 140 on the x-axis for scheduling 100 `vms`) are due to the randomness of the workload we automatically generate and some variability in Devstack. However, overall, our observation holds true.

Experiment 4 - *Host Utilization.* Finally, this experiment concerns the impact of conflict-free scheduling on the overall utilization level of all the physical servers. Since we know from experiment 2 that resource requirements are predominantly impacted by maximum degree, in figure 9, in the x-axis we vary maximum degree while scheduling a varied number of `vms`. The y-axis specifies the aggregate percentage of utilization of all the servers after scheduling the `vms` in a conflict-free manner. For example, given N number of servers, 80% utilization means that 20% of N servers in total is not utilized. We can see, server utilization dramatically increases with the number of `vms` that are scheduled. This is because since the max degree dictates server requirements, for smaller number of `vms`, a minimum of max degree number of servers remain heavily under-utilized. Once the `vms` scale toward real-world numbers, the utilization is above 80% even with a very high degree of conflict.

5. INCREMENTAL CONFLICTS

So far, our conflict-free scheduling approach has assumed that conflicts can be pre-specified and remains unchanged. However, in practice, conflicts may change, and may be specified incrementally as new tenants join the cloud. We now explore this fundamentally hard problem—if two `vms` that did not conflict at a certain time happen to be co-located in a server, but later develop a conflict due to an update of conflict specification, it is necessary to migrate one of those `vms` from that server, to remain conflict free.

5.1 Types of Conflict Change

In general, a conflict-set changes if a new conflict is added or an existing conflict is removed. Given a ConSet_{att} and a $\mathsf{PARTITION}_{att}$ of an $att \in \mathsf{ATTR}_{\mathsf{VM}}$, ConSet_{att} can change to a new conflict set ConSet'_{att} (a new partition $\mathsf{PARTITION}'_{att}$ can be calculated accordingly) in three different ways.

- Δ_1—this type of change involves operations that only remove an element from ConSet_{att} where $|\mathsf{PARTITION}'_{att}| < |\mathsf{PARTITION}_{att}|$. Evidently, it does not add new conflicts, hence, the scheduled `vms` need not migrate.
- Δ_2—this type of change involves operations that add an element to ConSet_{att}. However, $\mathsf{PARTITION}_{att}$ remains unchanged. If addition of a new conflict results in no change in conflict-free partition, scheduled `vms` need not be migrated.
- Δ_3—this type of change adds an element to ConSet_{att} where $\mathsf{PARTITION}'_{att} \neq \mathsf{PARTITION}_{att}$. Evidently, certain `vms` need to be migrated if they need to remain conflict-free.

Consider an attribute $att \in \mathsf{ATTR}_{\mathsf{VM}}$ and SCOPE_{att} = {a1,a2,a3,a4,a5,a6}, where the initial conflict-set ConSet_{att} = {{a1,a2},{a1,a4}, {a2,a4}, {a1,a5}, {a2,a6}, {a4,a6}} and the corresponding partition set which is calculated using algorithm 1 is $\mathsf{PARTITION}_{att}$={{a1,a3,a6},{a2,a5},{a4}}.

Consider a change of type Δ_1 that removes {a2,a4} from ConSet_{att} where resultant conflict set ConSet^1_{att}={{ a1,a2}, {a1,a4},{a1,a5},{a2,a6},{a4,a6}} and $\mathsf{PARTITION}^1_{att}$ ={{a1,a3,a6}, {a2,a4,a5}}. Here, $\#\mathsf{PARTITION}^1_{att} < \#\mathsf{PARTITION}_{att}$ and it does affect already scheduled `vms`.

Consider a change of type Δ_2 that adds {a2,a3} to ConSet_{att} where new conflict set ConSet^2_{att}={{a1,a2}, {a1,a4}, {a2,a4}, {a2,a3}, {a1,a5}, {a2,a6}, {a4,a6}} and $\mathsf{PARTITION}^2_{att}$= {{a1,a3,a6}, {a2,a5},{a4}} which is equal to the previous partition set $\mathsf{PARTITION}_{att}$.

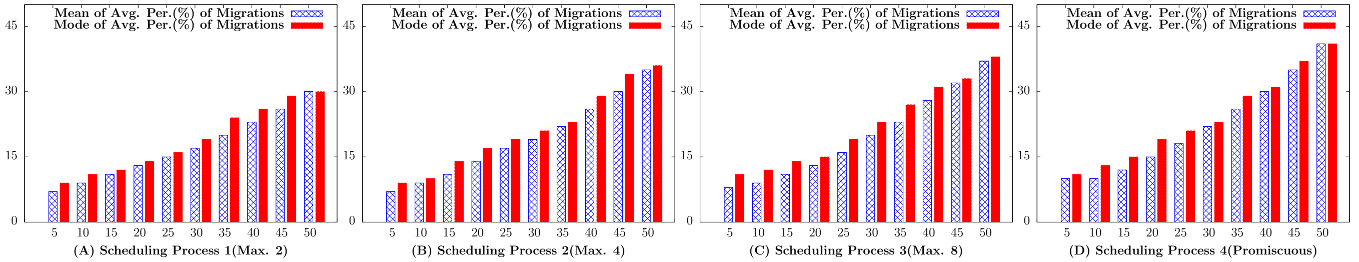
Consider a change of type Δ_3 that adds {a1,a6} to ConSet_{att} where ConSet^3_{att}= {{a1,a2}, {a1, a4}, {a2,a4}, {a1,a5},{a2,a5},{a2,a6},{a4,a6},{a1,a6}} and $\mathsf{PARTITION}^3_{att}$ = {{a1,a3},{a2},{a4},{a6}}. This clearly affects the previously scheduled VMs because, from $\mathsf{PARTITION}_{att}$, `vms` with attribute value a4 are co-located with `vms` with attribute values a1 or a3. Now, those `vms` with a4 need to migrate since they cannot co-locate with a1 or a3.

5.2 Cost Analysis

In this section, we analyze the cost of continuing to satisfy the conflicts as they change, when the change is of type Δ_3. We calculate the cost based on the number of migrations that are necessary when conflicts change. Based on experimentation, we gain insights on the strategies for minimizing the cost while handling this type of change.

We define an incremental plan, or simply plan, as a sequence of operations that adds a number of conflicts to the current conflict-set resulting in a Δ_3-type change (i.e., requires migration). Our strategy for minimizing cost is as follows. Consider an element $\{a1, a2, a3, a4\}$ of a conflict-free partition set $\mathsf{PARTITION}_{att}$ of attribute att. Since attribute values $a1$ through $a3$ are conflict-free, the scheduler is free to co-locate `vms` that have those attribute values in a given server. We refer to this as *promiscuous* conflict-free scheduling because it maximizes the mixing of `vms` in a given server so long as they do not conflict. In contrast, a *conservative* approach minimizes the co-location of `vms` even though their attribute values do not conflict. For instance, `vms` with values $a1$ or $a2$ may be co-located in one server, and those with values $a3$ or $a4$ may be co-located in another. In this case, if values $a3$ and $a1$ were to develop a conflict in the future, the migration cost can be minimal (zero in this scenario). Promiscuous scheduling can have better resource utilization but higher cost for managing conflict changes. Conservative scheduling can minimize cost when conflict changes more frequently, at the expense of lower resource utilization.

We conduct an experiment to evaluate the impact of conflict change on the number of migrations for different levels of conservative scheduling. The steps of the experiment are: $\mathsf{(step\text{-}1)}$ We consider a single `vm` attribute called att where we vary the size of SCOPE_{att} from 10 to 35 with an increment of 5. $\mathsf{(step\text{-}2)}$ For each SCOPE_{att}, initially, we randomly populate ConSet_{att} with 5 to 50 elements and calculate $\mathsf{PARTITION}_{att}$. We repeatedly perform this step for 50 times for every $\mathsf{step\ 1}$. $\mathsf{(step\text{-}3)}$ For each $\mathsf{step\text{-}2}$, we schedule X number of `vms` where we vary X from 500 to 5000. We also schedule them using a promiscuous approach and four conservative approaches where VMs of same `host` can not have



(A) Scheduling Process 1(Max. 2) (B) Scheduling Process 2(Max. 4) (C) Scheduling Process 3(Max. 8) (D) Scheduling Process 4(Promiscuous)

Figure 10: Cost Analysis: X-axis(% of the Total Conflicts for Given Scopes), Y-axis(% of Total VMs that Require Migrations)

more than 1, 2, 4, and 8 different values from a conflict-free partition respectively. Also, each vm is randomly assigned a value to its *att*. We repeat each scheduling process for 30 times. We also randomly assign vm memory capacity to 512 and 1024 MB, and host capacity to 3GHz. (step-4) Finally, we measure migrations for 5 different plans where the plans gradually add random 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%, 45% and 50% of the total number of conflicts to ConSet$_{att}$ respectively. For each plan, step-4 is repeated for 50 times and we count the migrations. Note that these numbers (the number of times a particular step is repeated) provide sufficient variations, and are primarily dictated by amount of time it takes to perform these steps.

Figure 10 shows the result of our analysis. Parts (A), (B) and (C) are results of different degrees of conservative scheduling. For example, in part (A), if attribute values $\{a1, a2, \ldots, a8\}$ are conflict free, we at most schedule vms with one of two possible conflict-free values in any given server (e.g. vms with a1 or a2 are co-located, and those with a3 and a4 are co-located in a different server, etc.). Similarly, in part (B), we co-locate vms with either of a1, a2, a3 or a4 in one server and those with a5, a6, a7 or a8 in a different server. Part (D) is the result of promiscuous scheduling.

We found that the percentage of the migrating vms does not necessarily increase with the increasing number of vms, rather, it depends on the percentage of total number of conflicts that are newly added. For instance, in figure 10(A), for varying number of vms from 500 to 5000, mean value of the average percentage of vms that need to migrate is 29% when number of newly added conflicts is 35%. Also, the mode is 27%. We found that the average difference between the mean and mode values from all cases is no more than 0.5%. The percentage of migrations remain constant with respect to the size of the attribute scope and it does not depend on the initial conflicts for which the vms are scheduled. Finally, we found that it is always better to schedule vms with conservative scheduling with minimum degree. For instance, there is no migration using scheduling process #1 where a host can only contain vms with same attribute value. Also, we notice that addition of a large % of conflicts at a time costs less than combined cost of multiple additions of comparatively small % of conflicts. For instance, in Figure 10(C), 50% conflicts cost 79% migrations, where 10 different 5% conflicts cost 10×9%=90% migrations.

5.3 Reachability Heuristics

Besides analyzing the cost of a plan that leads to a particular conflict set, it is also important to find the steps of a plan where each step adds a particular conflict. For instance, identifying steps of a plan helps to design operations for maintaining conflicts and their authorization process, although, we consider the designing of such front-end operational model as future work. Here, we define this problem as plan reachability problem where for a given attribute, its scope, and an initial conflict-set, what are the steps with a particular cost that will reach target plan with specific values in conflict-set? This problem can be viewed as finding a path from an initial state to a goal state in a weighted state-transition directed graph where each edge of the graph is the cost for adding one conflict to the conflict-set. Here, a simple algorithm can construct the state-transition graph and uses a weighted shortest path algorithm to find a plan in $O(nlogn)$ time [14]. However, it is infeasible due to a very large number of states where, for a size of scope N, the number of conflicts is $\binom{N}{2}$ and possible states are $2^{\binom{N}{2}}$. Instead, it is possible to use a search algorithm to construct regions as needed. Proper heuristics can intelligently search for steps and some well-known heuristics such as k-lookahead based heuristics may be applied in this domain [14].

6. SECURITY ISSUES AND LIMITATIONS

In terms of applicability, an attribute of a vm can be applied to represent properties of a single tenant or multiple tenants. We refer such attributes as intra-tenant and inter-tenant respectively. In figure 2, *tenant* and *sensitivity* are inter-tenant and intra-tenant attributes respectively since values of *tenant* can represent different tenant in the system, while, *sensitivity* can be very particular to a tenant. We analyze the following security concerns for specifying conflicts of the inter-tenant attributes in a multi-tenant cloud.

• *Privacy of a Tenant.* As seen in section 2.1, a conflict is specified between a pair of values of an attribute. However, for an inter-tenant attribute, the values of the attribute can belong to different tenants. For instance, in public cloud, values of the *tenant* attribute of figure 2 represent each tenant in the system and each tenant should not know values of *tenant* attribute except their own value for privacy of other tenant in this system. Specifying conflicts of such attributes can be very tricky where a tenant should be able to specify the conflicts with other tenants without, basically, knowing them. The *CSP* could take the initiative to develop a privacy preserving conflict specification process for inter-tenant attribute. A simple approach could be the classification of attribute-values based on some class, as shown in section 3.1 for conflict-of-interest classes, and a tenant can only mention the class of their attribute values where conflicts will be generated automatically with other values of the same class.

• *Disrupt Multi-tenancy:* In public cloud, multiplexing is to share a physical host among the vms of multiple tenants. However, if a tenant can specify conflicts with all other tenants in the system, then its vms cannot co-locate with any other tenant. This process disrupts the multi-tenancy in the system and, basically, creates a private cloud for the tenant. The *CSP* should restrict such specifications of conflicts.

We discuss following limitations on expressive-power of the generated conflicts by our mechanism.

• *Homogeneous and Non-hierarchical.* Generated conflicts in a conflict-set are treated equally and they do not have any hierarchical relationships. In figure 2, three different conflicts are specified in $\mathsf{ConSet}_{sensitivity}$ of attribute *sensitivity*. Here, each conflict has the same semantics, which is a binary relation between two values of *sensitivity*. Also, generated conflicts of the values of two different attributes are independent and bear equal meaning. In figure 2, the values of $\mathsf{ConSet}_{sensitivity}$ and $\mathsf{ConSet}_{\mathsf{tenant}}$ do not have any connection and have equal significance.

• *Conflicts between the Virtual Resources only.* Our scheduling mechanism does not consider any host property, such as location or trust-level of a host, for the scheduling decisions. Rather, it only focuses on generating attribute and their conflicts only for the vms and schedule them accordingly. Also, it does not consider any relationship between hosts and vms for the scheduling. Such type of relations between hosts and vms are specified in [25]. A potential future extension is to consider conflicts between hosts and vms for the scheduling decisions while optimizing the number of hosts.

7. RELATED WORK

Generally scheduling problems are NP-Complete. However, these problems are well-studied by the research community where they proposed various heuristic and approximate approaches for addressing different issues. For instance, the goal of resource-constrained multi-project scheduling problem is to minimize average delay per project. A number of efforts have been made in this scheduling problem including the priority rule based analysis [13, 27] where they propose heuristics, such as first-come-first-served, and shorted operation first, to minimize average delay. Another scheduling problem is to minimize number of bins, while scheduling a number of finite items in it. This problem is called bin packing. There are one and multi capacity bin packing based on multiple requirements for scheduling and approaches have been proposed in [28]. Multi-capacity bin packing is also applied in resource scheduling in grid computing [33, 34]. One variation of bin packing problem is called bin-packing with conflicts that packs items in a minimum number of bins while avoiding joint assignments of items that are in conflict. This problem is analogous to the problem we address in this paper. Several bin-packing with conflict algorithms [23, 24] are proposed where it is assumed that items can be conflicting in random manner. However, we investigated the nature of various conflicts for scheduling items (vms) where the items do not have direct conflict with each other, rather the attributes of the item have conflicts.

Different performance and security issues exist in cloud IaaS for unorganized multiplexing of resources and several of which are summarized in [17, 21, 35]. Recently, articles have been published exposing the vulnerability of state-of-art co-residency system in public cloud IaaS system [39, 40]. However, the virtual resources schedulers designed by the commercial IaaS clouds such as Amazon and IBM mainly aim to address performance management or load balancing related issues rather than security conflicts that we address in this article. Developing proper vm placement algorithms recently drew attention from the research community. Boboff et al [12] propose an algorithm that proactively adapts to demand changes and migrates virtual machines between physical hosts. Yang et al [38] also propose a load-balancing approach in vm scheduling process. Calcavecchia et al [15] develop a process to select candidate host for a vm by analyzing past behaviors of a host and deploy the request, and Gupta et al [19] propose a process for scheduling HPC related vms together. Li et al [29] propose vm-placement that maximizes a hosts cpu and bandwidth utilization. Also, Mastroianni et al [30] propose a probabilistic approach for vm scheduling for maximizing CPU and RAM utilization of the hosts. The main focus of these efforts is scheduling vms either for the purpose of high-performance computing or load balancing. Our approach is to capture different properties of vms by means of assigned attributes, and scheduling them while respecting conflicts expressed over those attributes.

8. CONCLUSION

We presented a generalized attribute-based constraint specification framework for virtual resource to physical resource scheduling in IaaS clouds. The mechanism also optimizes the number of physical resources while satisfying the conflicts. A potential future work is to extend this mechanism to address the limitations discussed in section 6. Another future research is to develop a suitable front-end application program interface for specification and management of the conflicts. Our vision is to expose resource management capabilities to the tenants.

9. ACKNOWLEDGEMENT

This research is partially supported by NSF Grants (CNS-1111925 and CNS-1423481).

10. REFERENCES

[1] AWS availabiltiy-zones. *http://docs.aws.amazon.com/AWSEC2/latest/using-regions-availability-zones.html/*.

[2] Devstack. *https://wiki.openstack.org/wiki/DevStack*.

[3] Openstack. *http://docs.openstack.org/*.

[4] Amazon and CIA ink cloud deal. In *http://fcw.com/articles/2013/03/18/amazon-cia-cloud.aspx*, 2013.

[5] 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*, pages 9–20. ACM, 2014.

[6] E. A. Bender and H. S. Wilf. A theoretical analysis of backtracking in the graph coloring problem. *Journal of Algorithms*, 6(2):275–282, 1985.

[7] A. Berl et al. Energy-efficient cloud computing. *The computer journal*, 53(7):1045–1051, 2010.

[8] K. Bijon, R. Krishman, and R. Sandhu. Constraints specication in attribute based access control. *ASE Science Journal*, 2(3), 2013.

[9] K. Bijon, R. Krishnan, and R. Sandhu. Towards an attribute based constraints specfication language. In *Proc. of the International Conference on Privacy, Security, Risk and Trust*. IEEE, 2013.

[10] K. Bijon, R. Krishnan, and R. Sandhu. A formal model for isolation management in cloud infrastructure-as-a-service. In *Proceedings of the Network and System Security*, pages 41–53. Springer, 2014.

[11] K. Bijon, R. Krishnan, and R. Sandhu. Virtual resource orchestration constraints in cloud infrastructure as a service. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 183–194. ACM, 2015.

[12] N. Bobroff et al. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management*, pages 119–128. IEEE, 2007.

[13] T. R. Browning and A. A. Yassine. Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 2010.

[14] D. Bryce and S. Kambhampati. A tutorial on planning graph based reachability heuristics. *AI Magazine*, 2007.

[15] N. M. Calcavecchia et al. Vm placement strategies for cloud scenarios. In *IEEE Cloud*, 2012.

[16] E. G. Coffman Jr et al. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

[17] W. Dawoud, I. Takouna, and C. Meinel. Infrastructure as a service security: Challenges and solutions. In *IEEE INFOS*, pages 1–8, 2010.

[18] M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[19] A. Gupta et al. HPC-aware vm placement in infrastructure clouds. In *IEEE Intl. Conf. on Cloud Engineering*, volume 13, 2013.

[20] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45(1):19–23, 1993.

[21] K. Hashizume et al. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):1–13, 2013.

[22] S. Iyer. Top 5 challenges to cloud computing. *Cloud Computing Central, https://www.ibm.com/*, 2011.

[23] K. Jansen. An approximation scheme for bin packing with conflicts. In *Algorithm Theory—SWAT*. 1998.

[24] K. Jansen. An approximation scheme for bin packing with conflicts. *Combinatorial Optimization*, 3(4), 1999.

[25] R. Jhawar, V. Piuri, and P. Samarati. Supporting security requirements for resource management in cloud computing. *IEEE CSE*, 0:170–177, 2012.

[26] D. Karger et al. Approximate graph coloring by semidefinite programming. In *35th Annual Symp. on Foundations of Computer Science*, pages 2–13, 1994.

[27] I. S. Kurtulus and S. C. Narula. Multi-project scheduling: Analysis of project performance. *IIE Transactions*, 1985.

[28] W. Leinberger et al. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Proc. of ICPP*, 1999.

[29] K. Li et al. Elasticity-aware virtual machine placement for cloud datacenters. In *IEEE 2nd Int. Conf. on Cloud Networking*, pages 99–107, Nov 2013.

[30] C. Mastroianni et al. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Tran. on Cloud Computing*, 1(2):215–228, 2013.

[31] T. Ristenpart et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proc. of the ACM CCS*, 2009.

[32] J. Rivera. Gartner identifies the top 10 strategic technology trends for 2014. *http://www.gartner.com.*

[33] M. Stillwell et al. Resource allocation using virtual clusters. In *IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pages 260–267, May 2009.

[34] M. Stillwell et al. Dynamic fractional resource scheduling for HPC workloads. In *IEEE Int. Symp. on Parallel Distributed Processing*, pages 1–12, 2010.

[35] H. Takabi, J. B. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, 8(6):24–31, 2010.

[36] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift. Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In *ACM CCS*, 2012.

[37] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *JACM*, 30(4), 1983.

[38] C.-T. Yang et al. A dynamic resource allocation model for virtual machine management on cloud. In *Grid and Distributed Computing*. Springer, 2011.

[39] Y. Zhang et al. Homealone: Co-residency detection in the cloud via side-channel analysis. In *IEE S&P*, 2011.

[40] Y. Zhang et al. Cross-vm side channels and their use to extract private keys. In *ACM CCS*, 2012.

[41] Y. Zhang et al. Cross-tenant side-channel attacks in paas clouds. In *ACM CCS*, 2014.

APPENDIX

A. APPROXIMATE ALGORITHMS

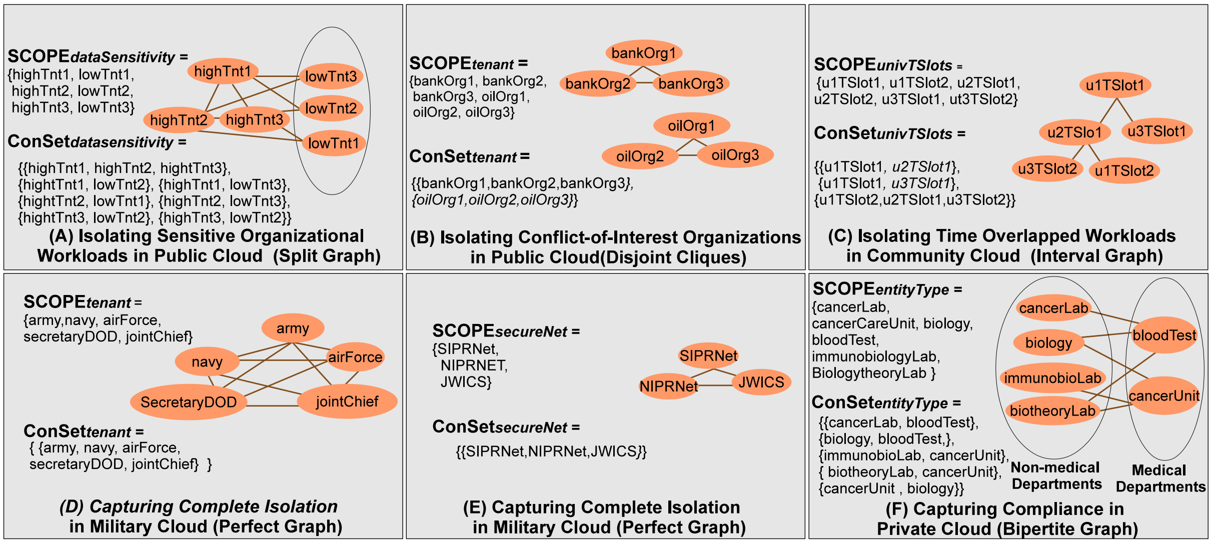
Present literature contains a number of approximate algorithms for the graph-coloring problem which also can be used for solving the MIN_PARTITION problem. For instance, an approximate graph-coloring algorithm given in [37]. Their Algorithm B takes as input a graph G(V, E) and a variable k, and returns *true* if G is k-colorable. Then algorithm C finds colors for the vertices in G using a binary search. It is shown that if the chromatic number (i.e., the minimum number of colors for coloring the graph) of a graph G(V, E) of n vertices is denoted $\mathcal{X}(\mathrm{G})$, the approximate colors generated by their algorithms is $2 \times \mathcal{X}(\mathrm{G}) \times \lceil \mathrm{n}^{1-1/(\mathcal{X}(\mathrm{G})-1)} \rceil$ (where n is the number of vertices) and the running time of the algorithm is $\mathcal{O}((|\mathrm{V}| + |\mathrm{E}|) \times \mathcal{X}(\mathrm{G}) \times log\mathcal{X}(\mathrm{G}))$. Therefore, for a given ConSet_{att} of an $att \in \mathsf{ATTR}_{\mathsf{VM}}$, if the minimum number of conflict-free partitions is p, this algorithm will generate $2 \times p \times \lceil {\mathsf{SCOPE}_{att}}^{1-1/(p-1)} \rceil$ number of conflict-free partitions. A few other approximate approaches include [20, 26].

B. RESTRICTED CONFLICT GRAPHS

This section explores restricted graphs having polynomial-time solutions and demonstrate their usage scenarios for private, public, and community cloud deployment scenarios.

B.1. Public Cloud

A public cloud provides compute services to multiple tenants. We present two scenarios where tenants may need isolation depending on the kind of data the `vm`'s process.



(A) Isolating Sensitive Organizational Workloads in Public Cloud (Split Graph)

(B) Isolating Conflict-of-Interest Organizations in Public Cloud(Disjoint Cliques)

(C) Isolating Time Overlapped Workloads in Community Cloud (Interval Graph)

***(D) Capturing Complete Isolation* in Military Cloud (Perfect Graph)**

(E) Capturing Complete Isolation in Military Cloud (Perfect Graph)

(F) Capturing Compliance in Private Cloud (Bipertite Graph)

Figure 11: Conflicts of different Systems and Corresponding Conflict Graphs

1. Sensitive Organizational Data: Suppose an e-commerce organization moves to a public cloud. An expectation could be that the vm's that run the general website may be co-located with other tenants while those that process sensitive data such as customer's credit card information or PII should not be co-located. This is infeasible in current public clouds since a tenant can only manually choose to avail services from clouds and carefully distribute the vm's across those clouds based on data sensitivity.

Such scenarios can be easily automated using our conflict specification framework. In this situation (figure 11-A), the cloud provider generates an attribute called *dataSensitivity* and for each tenant it includes two values, e.g., highTnt$_i$ and lowTnt$_i$ for tenant$_i$, to represent the high and low sensitivity of data that will be respectively processed by the vms. When a tenant creates a vm it assigns an appropriate value to the *dataSensitivity* attribute. Here, a vm with highTnt$_i$ would conflict with all the vms of other tenants, however, it does not conflict with vms of own tenant. Conflict-Set of this attribute is a split graph, hence, can be solved in polynomial-time [18].

2. Conflict-of-Interest: Please refer back to section 3.1 and figure 11-B for conflict-of-interest use cases.

B.2. Community Cloud

In a community cloud, the infrastructure is typically shared between enterprises with a common interest.One example of a community cloud is a scientific computing cloud infrastructure that is shared between, say, a set of universities. Figure 11-C illustrates an example where compute resources of participating universities must be isolated if the time-slot assigned to those universities happen to overlap. If there is no overlap in the time-slot, university 1, for example, can use the same physical host that was allocated to university 2 (though at a different time). Such a scenario forms an interval graph for which can be solved in polynomial-time [18].

B.3. Private Cloud

A private cloud has a single owner and thus does not share infrastructure with other tenants. The cloud infrastructure is typically hosted and operated in-house by the tenant or sometimes outsourced to a service provider. A great example is the private cloud operated by Amazon for the CIA [4].

1. Sensitivity in Military Cloud: Consider a large-scale cloud for the US Department of Defense (DoD). A fundamental principle in DoD's move to IaaS cloud from their current IT infrastructure could be that the different military organizations including army, navy and air-force, and their operations need to be isolated from each other consistent with the current operational status of each organization (currently, most of each organization's infrastructure is isolated from each other). To this end, a vm attribute *military-Org* can be created where $\mathsf{SCOPE}_{militaryOrg}$={army, navy, airForce, secretaryDoD, jointChief} and all values of *militaryOrg* would conflict with each other. The graph generated from this conflict-set is a complete graph as illustrated in figure 11-D which can be solved in polynomial-time [18].

Figure 11-E illustrates another DoD example resulting in a complete graph where vm's processing data belonging to different networks (such as SIPRNet, NIPRNet and JWICS) in the DoD need to be isolated from each other.

2. Compliance in Healthcare Cloud: For the compliance scenario, consider a *hybrid entity* in Health Insurance Portability and Accountability Act (*HIPAA*) that provides both healthcare and non-healthcare related services. An example of such entity is a university that includes a medical center that provides health-care services to the general public and also research labs in the university that conduct healthcare-related research internally. HIPAA rule mandates that such a hybrid entity should maintain a strict separation between those departments while handling protected health information (PHI). In order to comply strictly with HIPAA, virtual resources processing PHI need to be isolated. Such a scenario is illustrated in figure 11-D where bloodTest and cancerUnit are departments that provide healthcare and hence utilize compute services that process PHI. Those compute services need to be isolated from compute services of non-medical departments such as immunobiologyLab. This scenario forms a bipartite graph which has polynomial-time [18] coloring.