Virtual Resource Orchestration Constraints in Cloud Infrastructure as a Service

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

In an infrastructure as a service (IaaS) cloud, virtualized IT resources such as compute, storage and network are offered on demand by a cloud service provider (CSP) to its tenants (customers). A major problem for enterprise-scale tenants that typically obtain significant amount of resources from a CSP concerns orchestrating those resources in a secure manner. For instance, unlike configuring physical hardware, virtual resources in IaaS are configured using software, and hence prone to misconfigurations that can lead to critical security violations. Examples of such resource orchestration operations include creating virtual machines with appropriate operating system and software images depending on their purpose, creating networks, connecting virtual machines to networks, attaching a storage volume to a particular virtual machine, etc. In this paper, we propose attribute-based constraints specification and enforcement as a means to mitigate this issue. High-level constraints specified using attributes of virtual resources prevent resource orchestration operations that can lead to critical misconfigurations. Our model allows tenants to customize the attributes of their resources and specify fine-grained constraints. We further propose a constraint mining approach to automatically generate constraints once the tenants specify the attributes for virtual resources. We present our model, enforcement challenges, and its demonstration in OpenStack, the *de facto* open-source cloud IaaS software.

Categories and Subject Descriptors

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

General Terms

Security

CODASPY'15, March 2–4, 2015, San Antonio, Texas, USA.
Copyright © 2015 ACM 978-1-4503-3191-3/15/03 ...$15.00.
http://dx.doi.org/10.1145/2699026.2699112 .

Keywords

Cloud IaaS, Virtual-Resource Orchestration, Constraints, Configuration Policy, Security Policy Mining

1. INTRODUCTION

Enterprises are increasingly driven by economics and flexibility to use computing resources provided by cloud IaaS [23]. In cloud IaaS the physical resources in a datacenter are logically arranged by the cloud service provider (CSP), while computing resources are virtualized and hosted on those logical collections of physical resources. This is illustrated in figure 1 where, for example, a rack is a collection of a specific set of physical servers and network hosts. Other resources such as physical storages 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. In cloud IaaS, enterprises or tenants obtain a number of separate pieces of virtual computing resources (or simply virtual resources or resources), e.g. virtual machines (VMs), virtual networks (NETs), etc., from the CSP, where a physical resource is shared by multiple virtual resources for maximizing utilization and reducing cost. IaaS providers allow multi-tenancy which multiplexes virtual resources of multiple enterprises upon same hardware. This includes co-location of VMs from different tenants on a single (physical) host, sharing of VM images in a repository, etc. This is illustrated as virtual resource to physical resource mapping in figure 1. This raises many security and performance considerations for a tenant's workload in the cloud. For instance, a tenant's VMs can be attacked by co-located malicious VMs of an adversary tenant. Such issues that arise due to multi-tenancy of virtual resources on physical resources have been extensively investigated in the past (see for example [19, 22, 28, 29, 32, 33]).

Another major issue arises from the fact that, for a given tenant with large-scale, heterogeneous virtual resources in IaaS, orchestrating those resources in a secure manner is cumbersome. Virtual to virtual resource mapping relations are also shown in figure 1. Here, orchestration refers to resource management issues such as creating networks, designing network layouts, applying appropriate images to VMs, etc. Since, in IaaS resource orchestration operations are performed in software (unlike in the case of physical resources where, for instance, servers are physically connected via Ethernet cables), they are highly prone to misconfigurations that can lead to security issues or increased exposure. For instance, a web-facing VM can be accidentally connected

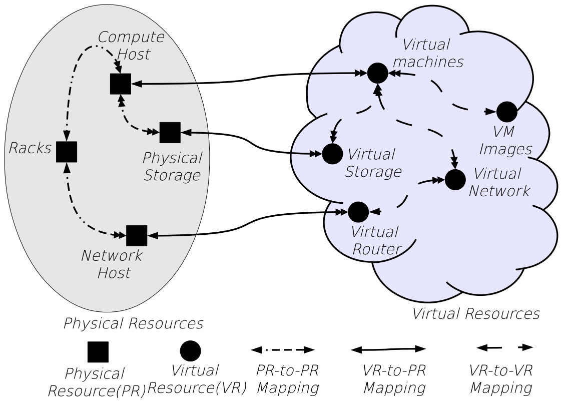


Figure 1: Cloud Resources Mapping Relation

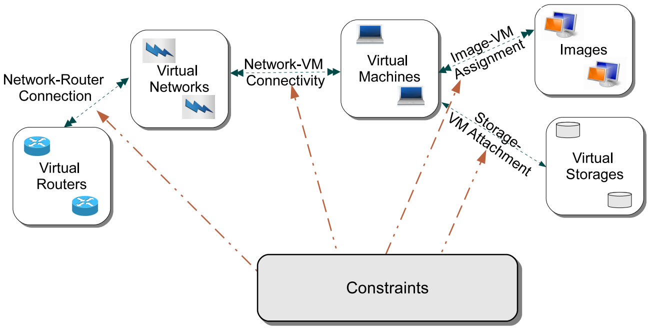


Figure 2: Constraints on Virtual Resources Arrangement Configurations

to a sensitive internal network or a low-assurance image may be applied to a VM that is expected to be security-hardened.

In this paper, we present the design, implementation, and evaluation of attribute-based CVRM (constraint-driven virtual resource management) as an approach to mitigate such concerns in cloud IaaS. Constraints in access control theory are effective ways of risk mitigation. A classic example is separation of duty in RBAC [24] where certain roles such as accounts manager and purchase manager conflict with each other and hence should never be assigned to the same user. Our developed CVRM enables tenants to express several essential properties of cloud resources as their attributes and specify constraints on resource mappings (VR-to-VR in figure 1) based on those attributes. We expect such constraints to be specified by a tenant administrative user or the CSP administrator. We provide a number of examples illustrating the utility of this technique in practical situations, such as configuring a Hadoop Cluster and a 3-tier business application in cloud IaaS. The CSP can then algorithmically enforce such constraints specified by all of its tenants when a virtual resource is mapped with another. We have implemented a prototype of CVRM in the widely-deployed OpenStack, the *de facto* open-source IaaS cloud software. We also propose an algorithm for mining the constraints in CVRM where the tenants specify the necessary attributes according to their business specifications and the algorithm automatically constructs the required constraints.

2. MOTIVATION

Migrating line-of-business applications to IaaS can be disastrous rather than beneficial for the tenants if their virtual resources are not properly configured. A misconfigured system not only hinders expected performance but also poses several security threats to a tenant. These threats include (i) malicious image insertion and inadvertent leakage of sensitive information through snapshot, (ii) sensitive information passing from a virtual machine to malicious virtual networks, and (iii) flow of information from a highly sensitive virtual network to a malicious or less sensitive one. However, present commercial cloud IaaS providers, including Amazon and Rackspace, offer at best rudimentary capabilities for such configuration management. For instance, AWS-IAM [1] offers a tenant to specify policies that can restrict resource-level permissions for certain users where the permissions include snapshot a VM, create a virtual storage with specific capacity, etc. On the other hand, Rackspace provides a fixed mechanism for isolation management where cloud resources and administrative users of a tenant, also referred as admin-users, can be grouped into different projects where admin-users can only configure the resources in their assigned projects. These fixed approaches lack the generality to capture diverse enterprise-specific requirements for configuring virtual resources. Moreover, in these user-driven configuration management setups, completely relying on the admin-users increases the risk of possible misconfiguration since admin-users may inadvertently create incorrect configurations. It also elevates potential for insider threats.

Motivated by this scenario, we aim to develop CVRM that offers a tenant to specify various constraints for configuring the required arrangements of virtual resources. We address the fact that security concerns due to misconfiguration will vary across line-of-business applications of the tenants. For instance, 3-tier business application will be concerned about protecting unauthorized disclosure of data, while hadoop cluster configurations will seek to ensure integrity and availability of the resources. CVRM is designed to address tenant-specific constraints where the constraints are enforced on user-operations that affect the configurations of virtual resources. Constraints specified by a tenant can be enforced on operations performed by the tenant's admin-users during regular operations or by CSP's admin-users in case of exceptions and troubleshooting. We believe that, in addition to any access control mechanism implemented in this system, CVRM provides resource management capability that prevents misconfiguration caused by admin-users.

Figure 2 shows conceptual view of constraint driven virtual resource management. Constraints can be specified for a specific mapping relation (or simply relation) between two virtual resources. We describe these mapping relations and possible misconfiguration issues. We also provide examples for 3-tier application and hadoop cluster configurations. Note that, 3-tier aims to isolate computational requirements of an organization by three different tiers—presentation (PS), application (APP), and database (DB), for better security and scalability. Hadoop is a master-slave architecture for faster analysis of big-data where security issues include integrity and availability of the resources.

- **IMG-VM Compatibility Relations:** As shown in figure 2, a virtual machine image, also referred as IMG, can be used by multiple VMs and also from a VM multiple snapshots can be imaged. This process provides quick replication of a VM into large numbers of VMs, and also quick migration of a VM to another server. However, incorrect usage of IMG can critically affect the security and performance in the system. For instance, in 3-tier, VMs running the application of each tier require separate IMGs since VMs in different tiers perform different operations. Thus, an IMG created for DB-tier is not to be used for VMs of PS-tier, since PS-tier VMs are web-facing and the IMG may expose critical information about DB-tier. Similarly, in hadoop, each type of VMs such as nameNode and taskTracker have different functionalities, whereby improper use of IMG can hamper performance and availability.

- **NET-VM Connectivity Relations:** A group of VMs, connect to a network NET, so as to internally communicate. However, a wrong VM connected to a NET may hamper communication in NET and availability of information. For instance, in 3-tier application, APP-tier VMs can be connected to each other for faster communications, however, accidental assignment of VMs from other tiers can hamper the flow. Similarly, the taskTracker VMs performing reduce jobs should be connected with each other and no other VMs should connect to this network.

- **RT-NET Connection Relations:** Using a virtual router (RT), VMs of two selected NETs can communicate. In 3-tier application, VMs of APP-tier and PS-tier should communicate, however, PS-tier should not directly communicate with DB-tier. Also, connection to the external internet is only authorized for tier-1 VMs. Similarly, in hadoop, a NET for nameNode VMs should only connect to the NET of jobTracker VMs.

- **STR-VM Attachment Relations:** A persistent virtual storage (STR), is like a hard-disk drive which can be attached and detached to multiple VMs, but one at a time, until it is destroyed. Note that, a STR attached to a VM stores data from the VM. Later, if the STR is detached and re-attached to another VM without deleting its data, the new VM will get access to the data of the previous VM.

3. DESIGN OF CVRM

Intuitively, an attribute captures a property of an entity in the system, expressed as a name:value pair. For instance, clearance can be a user attribute and values of clearance could be ‘top-secret’, ‘secret’, etc. In the context of cloud IaaS, various useful properties of the virtual resources, such as VMs and NETs, can be captured by associating attributes to them. For instance, attributes can represent a VM’s different properties including owner tenant, operational purpose, workloads sophistication, and connected networks. In CVRM, given that the properties of the virtual resources are represented by their attributes, a constraint is enforced while mapping (i.e., connecting) two virtual resources by comparing the specific attributes of the virtual resources.

In this section, we formally define CVRM that includes representation of the basic elements, relations among virtual resources and the constraint specification language. Then, we instantiate CVRM for 3-tier architecture in cloud IaaS.

3.1 Formal Specification

The basic elements of CVRM include representation of existing tenants and virtual resources in a cloud IaaS system where each virtual resource belongs to a particular tenant. A virtual resource is also mapped to a particular class of virtual resource such as VM, NET, IMG, RT, and STR. Formally, we have the following.

- VR is the set of all existing virtual resources in CSP.
- CLS is the set of all classes of virtual resources that are supported by the CSP.
- $rCls$: VR → CLS, is a function that maps each virtual resource to its class.
- TENANTS is the finite set of existing tenants in CSP.
- $tenant$: VR → TENANTS, is a function that maps each virtual resource to the tenant that owns it.
- $\mathsf{VR}_{\mathtt{tnt}}$ is the set of virtual resources that are owned by the tenant tnt. Formally, $\mathsf{VR}_{\mathtt{tnt}}=\{\mathrm{v}\in \mathsf{VR} \mid tenant(\mathrm{v})=\mathtt{tnt}\}$.

Here, $\mathsf{VR}_{\mathtt{tnt}}$ contains the virtual resources of a tenant tnt and these virtual resources are partitioned into different sets based on their class. We define such sets of the virtual resources of each tenant as follows,

- $\mathsf{VR}_{\mathtt{tnt},\mathsf{c}}$ is the set of virtual resources of class c that are owned by the tenant tnt. Formally, $\mathsf{VR}_{\mathtt{tnt},\mathsf{c}}=\{\mathrm{v}\in\mathsf{VR}_{\mathtt{tnt}} \mid rCls(\mathrm{v})=\mathsf{c}\}$.

In a tenant, a particular class of virtual resources can have specific type of mapping relations to virtual resources of another class. For instance, virtual resources of class VM can have connection-mapping and attachment-mapping relations with virtual resources of class NET and STR respectively. We can define the relations between elements of every two classes of virtual resources in a tenant as follows,

- $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ is the relation between virtual resources of class c_i and c_j in a tenant tnt. Formally, $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j} \subseteq \mathsf{VR}_{\mathtt{tnt},\mathsf{c}_i}\times\mathsf{VR}_{\mathtt{tnt},\mathsf{c}_j}$.

However, CVRM restricts the following type of relations,

1. Relations between virtual resources of same class cannot be specified (i.e., $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_i}$ cannot be specified).
2. For two classes $\mathsf{c}_i\neq\mathsf{c}_j$ we can define $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ or $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_j,\mathsf{c}_i}$ but not both.

CVRM provides two operations called $\mathcal{A}dd$ and $\mathcal{RM}$ respectively to add and remove tuples to a relation, where each operation is a function that takes as inputs the relation and two virtual resources of appropriate classes. Each operation also evaluates a specific constraint with respect to these two virtual resources as discussed below. Formally they are defined as follows (the notation for defining these operations is similar to the notation of schema used in NIST RBAC [25]),

$$\begin{array}{l} \mathcal{A}dd(\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j},\mathrm{vr}_1,\mathrm{vr}_2) \lhd \\ \quad \mathrm{vr}_1\in\mathsf{VR}_{\mathtt{tnt},\mathsf{c}_i} \wedge \mathrm{vr}_2\in\mathsf{VR}_{\mathtt{tnt},\mathsf{c}_j} \wedge consEval(\delta^{\mathcal{A}dd}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j},\mathrm{vr}_1,\mathrm{vr}_2) \\ \qquad\quad \mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}{}' = \mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j} \cup \{<\mathrm{vr}_1,\mathrm{vr}_2>\}\rhd \end{array}$$

$$\begin{aligned}&\mathcal{RM}(\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j},\mathrm{vr}_1,\mathrm{vr}_2) \lhd\\ &\quad \mathrm{vr}_1{\in}\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i} \wedge \mathrm{vr}_2{\in}\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j} \wedge \mathit{consEval}(\delta^{\mathcal{RM}}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j},\mathrm{vr}_1,\mathrm{vr}_2)\\ &\qquad\quad \mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}{}' = \mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j} - \{<\mathrm{vr}_1,\mathrm{vr}_2>\}\rhd\end{aligned}$$

Here, $\delta^{\mathcal{A}dd}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ and $\delta^{\mathcal{RM}}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ are constraints that are respectively specified for adding and removing a tuple to the relation $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$. A successful execution of an operation is allowed if the constraint is satisfied for the particular virtual resources vr_1 and vr_2. Both $\mathcal{A}dd$ and $\mathcal{RM}$ call the constraint evaluation function *consEval* with vr_1, vr_2 as inputs along with the relevant constraint. Evaluation of the constraint is a simple evaluation of a logical formula to true or false.

Basically, a constraint compares different properties assigned to the virtual resources vr_1 and vr_2 which are evaluated by *consEval* to make a decision. In CVRM, there are attributes of each class of virtual resource that characterize different properties of the resources 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. We assume values of attributes to be atomic,[1] therefore, for a particular element of that resource, the name of the attribute function maps to one value from the set. For convenience attribute functions are simply referred to as attributes. Formally, we have the following.

- $\mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ is the set of attribute functions of a virtual resource class c_i in tenant `tnt`. Here, for a function $att{\in}\mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, the domain of the function is the virtual resources $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}$ and the codomain is the values of *att* written as SCOPE_{att} which is a set of atomic values. Formally, $att : \mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}{\rightarrow}\ \mathsf{SCOPE}_{att}$ where $att{\in}\mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$.

Now, for each $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, at most two constraints can be specified for the operations $\mathcal{A}dd$ and $\mathcal{RM}$ respectively. Each constraint is used to verify if assigned values of specific attributes of two virtual resources vr_1 and vr_2 of class c_i and c_j respectively satisfy certain conditions. CVRM uses the grammar below to specify constraints,

```
<Quantifier>:= ∀(vr1,vr2) ∈ 𝓡_{<Cls>,<Cls>} . <Stmt>
<Stmt>:= <Stmt> <connector><Stmt> | (<rule>)
<rule>:= <Token> → <Token>
<Token>:= (<Token> <connector> <Token>)|(<Term>)
<Term>:= <Attribute>(<resource>) <comperator> <Scope>
<Attribute>::= <letter> | <digit> | <Attribute>
<Scope>::= <letter> | <digit> <Scope>
<connector>::= ∧ | ∨
<comparator>::= = | ≠
<Cls>::= c_1 | c_2 | . . . | c_n
<resoruce> ::= vr1 | vr2
<digit>::= 0|1|2| . . . |8|9
<letter>::= a|b|. . . |y|z|A|B|. . . |Y|Z
```

The constraint specification grammar syntax is given in Backus Normal Form (BNF). Each constraint statement contains single or multiple small expressions in the form of implication, A→B, joined by logical connectors. The small expression is also referred to as constraint-rule or just rule. Both A and B in a rule A→B contain one or more predicates connected by logical connectors, where a predicate contains an attribute function of a specific class of virtual resource and the function returns the assigned value to the attribute of a specific instance of that class and, then, the predicate compares the value with a particular value of the attribute. Basically, a rule, A→B, verifies that if assigned attribute values of a virtual resource vr1 meets the conditions specified in A then assigned attribute values of vr2 should satisfy the condition in B in order to insert vr1 and vr2 into a relation.

Note that, the grammar is also weakly typed since in each predicate <Attribute> and <Scope> are replaced by arbitrary names. To this end, we develop a simple static type-checking system that ensures valid constraint expression. For each predicate of a constraint, it checks if the value, specified after the <comparator> sign of the predicate, belongs to the scope of the attribute name specified before the <comparator>. It is formally defined as follows.

Predicate format:
attribute(<Resource>) <comparator> attribute-value

Type-Checking Rule:
If attribute-value ∈ $\mathsf{SCOPE}_{\mathrm{attribute}}$ *Then*
return *true*
Else
return *error*

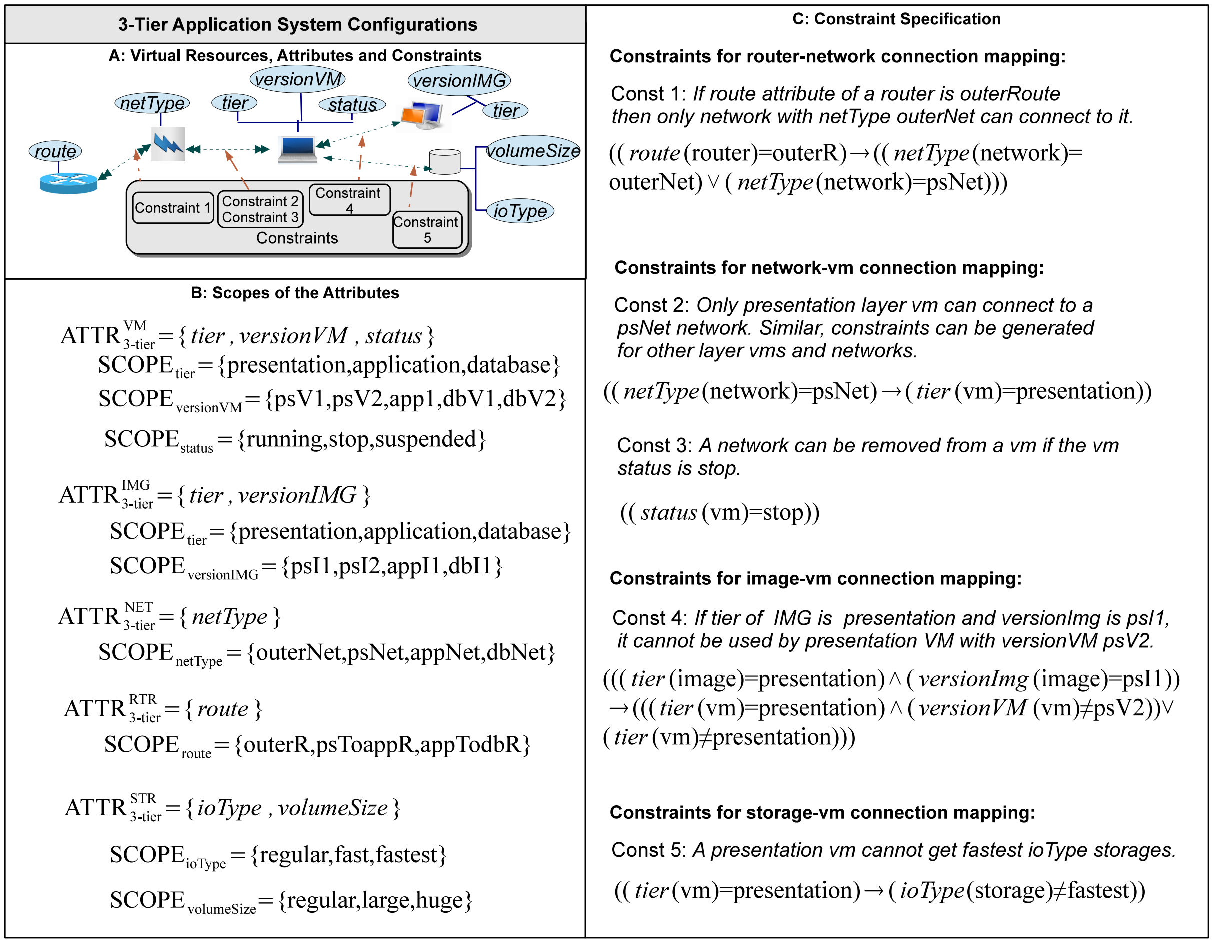
3.2 Instantiation

In this section, we instantiate CVRM for an example of 3-tier business application setup. Another example of hadoop cluster setup is given in the appendix. We focus on the tenant called `3-tier`. The classes of virtual resources supported by the CSP are VM, NET, RT, STR and IMG and `3-tier` supports relations from VM-to-NET, NET-to-RT, VM-to-IMG and VM-to-STR which are written as $\mathcal{R}_{\texttt{3-tier},\mathsf{VM},\mathsf{NET}}$, $\mathcal{R}_{\texttt{3-tier},\mathsf{NET},\mathsf{RT}}$, $\mathcal{R}_{\texttt{3-tier},\mathsf{VM},\mathsf{STR}}$ and $\mathcal{R}_{\texttt{3-tier},\mathsf{VM},\mathsf{IMG}}$ respectively.

Attributes are defined for the instances of each class of virtual resources that characterize different properties necessary to capture the requirements to run `3-tier` application in cloud. Figure 3-A identifies the attributes of the virtual resources of tenant `3-tier`. It also shows the mapping relation among virtual resources (represented by arrow-headed lines). Figure 3-B gives the scopes of these attributes. For instance, in A, a VM attribute *tier* represent the tier-operations a VM performs and B shows the scope of *tier* which is presentation, application, database. For each vm, *tier* assigns a value from the scope to the vm. An example attribute assignment for a vm that performs as a database server is: *tier*(VM)= database. Other two attributes of VM called *versionVM* and *status* represent the version of a VM in specific tier and the activity status respectively. Similarly, IMG also has attributes called *tier* and *versionIMG* that represnts the tier and version respectively for which an IMG is created. For `3-tier`, we also create a NET attribute called *netType* that specifies the layer for which a NET is created for the communication. For instance, a NET with *netType* value psNet should only carry presentation layer data. Figure 3-A and 3-B also defines attributes and their scopes for RT and STR respectively.

Generally, in CVRM, tenants can specify attributes for their virtual resources to capture specific organizational re-

[1] As given in section 2, in 3-tier application, an example of such constraints is to restrict communication between VMs of APP-tier and DB-tier. Here, if the attribute is called *tier* and the possible values are presentation, application and database, a `vm` can only get one of the three values. However, there might be constraints that require set-valued attributes where the virtual resources get multiple values. CVRM is not currently designed to express such constraints, however, can be easily extended to set-valued attributes.



B: Scopes of the Attributes

$\mathrm{ATTR}^{\mathrm{VM}}_{\text{3-tier}} = \{\, tier\,, versionVM\,, status\,\}$

$\mathrm{SCOPE}_{\mathrm{tier}} = \{\text{presentation,application,database}\}$

$\mathrm{SCOPE}_{\mathrm{versionVM}} = \{\text{psV1,psV2,app1,dbV1,dbV2}\}$

$\mathrm{SCOPE}_{\mathrm{status}} = \{\text{running,stop,suspended}\}$

$\mathrm{ATTR}^{\mathrm{IMG}}_{\text{3-tier}} = \{\, tier\,, versionIMG\,\}$

$\mathrm{SCOPE}_{\mathrm{tier}} = \{\text{presentation,application,database}\}$

$\mathrm{SCOPE}_{\mathrm{versionIMG}} = \{\text{psI1,psI2,appI1,dbI1}\}$

$\mathrm{ATTR}^{\mathrm{NET}}_{\text{3-tier}} = \{\, netType\,\}$

$\mathrm{SCOPE}_{\mathrm{netType}} = \{\text{outerNet,psNet,appNet,dbNet}\}$

$\mathrm{ATTR}^{\mathrm{RTR}}_{\text{3-tier}} = \{\, route\,\}$

$\mathrm{SCOPE}_{\mathrm{route}} = \{\text{outerR,psToappR,appTodbR}\}$

$\mathrm{ATTR}^{\mathrm{STR}}_{\text{3-tier}} = \{\, ioType\,, volumeSize\,\}$

$\mathrm{SCOPE}_{\mathrm{ioType}} = \{\text{regular,fast,fastest}\}$

$\mathrm{SCOPE}_{\mathrm{volumeSize}} = \{\text{regular,large,huge}\}$

C: Constraint Specification

Constraints for router-network connection mapping:

Const 1: *If route attribute of a router is outerRoute then only network with netType outerNet can connect to it.*

$((\,route\,(\text{router})=\text{outerR}) \rightarrow ((\,netType\,(\text{network})=\text{outerNet}) \vee (\,netType\,(\text{network})=\text{psNet})))$

Constraints for network-vm connection mapping:

Const 2: *Only presentation layer vm can connect to a psNet network. Similar, constraints can be generated for other layer vms and networks.*

$((\,netType\,(\text{network})=\text{psNet}) \rightarrow (\,tier\,(\text{vm})=\text{presentation}))$

Const 3: *A network can be removed from a vm if the vm status is stop.*

$((\,status\,(\text{vm})=\text{stop}))$

Constraints for image-vm connection mapping:

Const 4: *If tier of IMG is presentation and versionImg is psI1, it cannot be used by presentation VM with versionVM psV2.*

$(((\,tier\,(\text{image})=\text{presentation}) \wedge (\,versionImg\,(\text{image})=\text{psI1})) \rightarrow (((\,tier\,(\text{vm})=\text{presentation}) \wedge (\,versionVM\,(\text{vm})\neq\text{psV2})) \vee (\,tier\,(\text{vm})\neq\text{presentation})))$

Constraints for storage-vm connection mapping:

Const 5: *A presentation vm cannot get fastest ioType storages.*

$((\,tier\,(\text{vm})=\text{presentation}) \rightarrow (\,ioType\,(\text{storage})\neq\text{fastest}))$

Figure 3: Constraints Specification for 3-Tier Application System

quirements. Also, resources can have certain general properties irrespective of organizational diversities of the tenants. CVRM categorizes attributes in two types: one that captures the general properties across all the tenants (referred as inter-tenant attributes) and the other that captures tenant-specific properties (referred to as intra-tenant attributes).

For a 3-tier application, the tenant specifies VM attribute called *tier*, as shown in figure 3-A, for their operational purpose. Here, *tier* is intra-tenant attribute since this attribute does not capture anything in other applications such as hadoop. However, *volumeSize* attribute of STR represents the size of the volume and this attribute is required by virtual storage regardless of operational objectives of different tenants, and is thereby an inter-tenant attribute

In this setup, proper administration of the attributes is necessary where administration process should include creation and deletion of the attributes and their scopes as well as assigning correct attribute values to the virtual resources. Creation and deletion of inter-tenant attributes and their scopes should be managed by the CSP's admin-users, while, the attribute value assignment to virtual resources are performed by CSP's or tenant's admin-users or by the IaaS system as appropriate. Attribute administration is beyond the scope of this paper. However, there is literature on attribute administration [18] that might apply in this context.

Followed by attribute specification of the resources, the tenant `3-tier` specifies at most two constraints for the relations of every two classes. Some example constraints with high level descriptions are shown in figure 3-C. For instance, constraint $\delta^{\mathrm{Add}}_{\mathtt{3\text{-}tier},\mathsf{VM},\mathsf{NET}}$ applies to the $\mathcal{A}dd$ operation where it checks if a vm is connecting to an appropriate virtual network by comparing their attributes. Another constraint called $\delta^{\mathrm{RM}}_{\mathtt{3\text{-}tier},\mathsf{VM},\mathsf{NET}}$ applies to $\mathcal{RM}$ operation of same relation where it checks if the a vm is in stop state to disconnect it from a virtual network. Figure 3-C also shows example constraints for other relations.

4. CVRM ENFORCEMENT

We describe a CVRM enforcement in OpenStack cloud platform. Then, we analyze some security issues of CVRM.

4.1 Enforcement in OpenStack

Figure 4 shows conceptual picture of CVRM enforcement process in IaaS. This process includes a constraint specifier and constraint enforcer components. Constraint specifier specifies necessary attributes and their scopes for the virtual resources in IaaS. It also specifies the constraints for the operations that add/remove configuration-relations between two virtual resources. When users execute the operations, respective constraints are enforced. As shown in the figure, after getting each request from users, the constraint evalua-

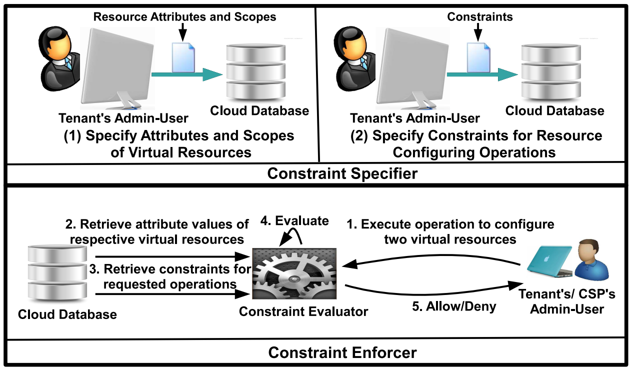

Resource Attributes and Scopes
Tenant's Admin-User
Cloud Database
(1) Specify Attributes and Scopes of Virtual Resources
Constraints
Tenant's Admin-User
Cloud Database
(2) Specify Constraints for Resource Configuring Operations
Constraint Specifier
2. Retrieve attribute values of respective virtual resources
4. Evaluate
1. Execute operation to configure two virtual resources
3. Retrieve constraints for requested operations
Cloud Database
Constraint Evaluator
5. Allow/Deny
Tenant's/ CSP's Admin-User
Constraint Enforcer

Figure 4: Components of CVRM Enforcement Process in a Service of OpenStack

tor retrieves the attributes of the virtual resources and the respective constraints from cloud database and evaluate the constraints to make decision.

4.1.1 OpenStack Overview

OpenStack comprises various service components that provide functionalities for managing different virtual resources. For instance, it has compute service called nova that offers operations for the management of VMs where the operations include create, delete, start and stop virtual machines. nova also has operations for arranging other virtual resources to VMs, e.g., connect VMs with NETs, attach STRs to VMs, etc. In OpenStack, each resource is a member of a specific project. A user is authorized to exercise a service operation to virtual resources of a project if she is a member of the project and has the role called *project-admin*. There is also a notion in OpenStack called domain where a domain consists multiple projects. A user who is a member of a domain and assigned to the role called *domain-admin* is responsible to create/delete projects in that domain as well as add/remove users to specific projects. We can consider such users of a domain and its projects as the super and regular admin-users of a tenant respectively. There is also a fixed domain called admin whose members are the CSP's admin-users. Members of the admin domain are responsible to create other domains and also add/remove users to them. Generally, in OpenStack, if a user requests a virtual resource configuration-operation, the authorization service which is called keystone provides a token that contains user authorization information including the projects where the user is a member. The operation is allowed if the project of respective virtual-resources are same as the requesting user.

Figure 5 shows execution steps of an user-operation (*volume-attach*) in OpenStack that attaches a STR to a VM. When a user in a project tries to execute the operation, the OpenStack client program retrieves the token for the user from keystone. Then, it forwards the token along with respective VM and STR names to nova since nova manages *volume-attach*. nova verifies validity of the token and collects the tenant information of VM and STR from database and it approves if the given user, VM and STR are in same tenant.

4.1.2 Constraint Specifier

Our designed constraint specifier component can be included in each service in OpenStack. The specifier extends respective service operations by adding functionalities for the creation and management of the attributes and their scopes for respective virtual resources. In a tenant (domain), managing such functionalities are only authorized for the users having *domain-admin* role in the domain. Specifier also provides operations for constraint specification. Each constraint is mapped to an operation-name to which it applies. Operations that specify constraints are also authorized only to users having *domain-admin* role. Attributes, their scopes and constraints are stored in database-tables of respective service. Entries in a database-table across tenants (domains) are isolated by specific domain ids and admin-users of a domain cannot access other domains' information.

Figure 6 shows a nova operation of the constraint-specifier component that specifies VM attributes. Database of nova contains tables for storing attributes and constraints. When a user tries to create an attribute, the token of the user is verified to check if the user has *domain-admin* role in order to make a decision. The component also contains similar operations that specify constraints.

4.1.3 Constraint Enforcer

Similar to constraint specifier, an enforcer component is included in every service in OpenStack. When a user executes a service operation that affects a relation between two virtual resources, enforcer verifies the respective constraint which is already specified by constraint specifier. This process retrieves attributes of the virtual resources and the constraint expression from service database. It implements an evaluator to evaluate the constraint for making a decision. Note that, in OpenStack, these operations are authorized only for project admin-users.

Figure 7 shows extended view of figure 5 for the execution of *volume-attach*. Besides, comparing the project information of the VM, storage and user, the enforcer component now retrieves the attributes for VM and STR and constraint for *volume-attach* and evaluates the constraints by considering the VM and STR attributes.

4.2 Security Analysis

We present different security issues for enforcing CVRM in practice.

4.2.1 Constraint Specifications Process

- **Constraints, Attribute and Scope:** CVRM aims to restrict privileges of admin-users in order to mitigate misconfiguration issues of a tenant. Therefore, constraints specification and modification process should be restricted to the majority of admin-users and only selected admin-users of a tenant should be authorized to specify constraints, attributes of the virtual resources and their scope. In OpenStack, there are three types of admin-users: CSP-admin, domain-admin and project-admin. In our developed constraint enforcement in OpenStack we only authorize domain-admins to manage the constraints, attributes and their scopes where the specified constraints are applied to all three type of admin-users. A more formal isolation management scheme is given in [8] that can also be applied here.

- **Attribute Value Assignments:** An admin-user who can create virtual-resources should also assign values to their attributes. In CVRM, the project-admin users

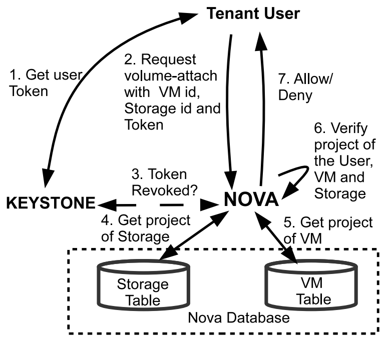

Tenant User
1. Get user Token
2. Request volume-attach with VM id, Storage id and Token
7. Allow/ Deny
6. Verify project of the User, VM and Storage
3. Token Revoked?
KEYSTONE
NOVA
4. Get project of Storage
5. Get project of VM
Storage Table
VM Table
Nova Database

Figure 5: Operation *volume-attach* in Nova

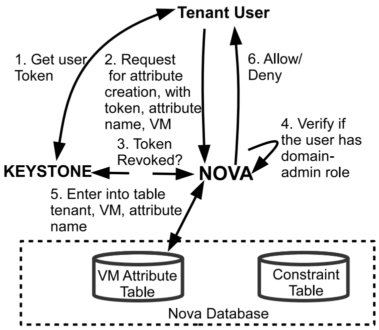

Tenant User
1. Get user Token
2. Request for attribute creation, with token, attribute name, VM
6. Allow/ Deny
3. Token Revoked?
4. Verify if the user has domain-admin role
KEYSTONE
NOVA
5. Enter into table tenant, VM, attribute name
VM Attribute Table
Constraint Table
Nova Database

Figure 6: Constraint Specifier in Nova

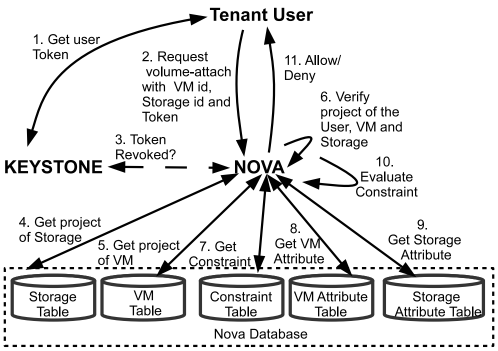


Figure 7: Constraint Enforcement for *volume-attach*

can assign values to the attributes. However, one needs to make sure that the admin-users can only assign appropriate values. For instance, a project-admin can create VMs and assign only her project-id to those VMs (not ids of other projects). In this paper we do not focus on such access control system, however, existing mechanisms such as [8] might be useful.

- **Generalized Enforcement Engine with Data Isolation:** For scalability, one generalized enforcement engine should be designed for the evaluations of the constraints of all the tenants. In our developed enforcement engine in OpenStack, constraints are stored in the database separated according to the domain-id of a tenant and only respective admin-users can have access to their constraints.

4.2.2 Issues on Constraint Structure

- **Contradictory Constraint:** The sub-expressions of a constraint can be of two types. One restricts the relation of virtual resources of two different classes when values of their attributes are mutual exclusive. Another one forces relation when the values of the attributes are congruent. A constraint containing both type of sub-expressions for same combination of values of the attributes generates contradictory decisions for a relation. We call these constraints as contradictory constraints and they need to be avoided.

- **Deadlock Constraints:** In a constraint, a value, let's say, val$_x$ of an attribute att_p of the virtual resources of specific class c_i can have mutual exclusion relation with all the values of an attribute att_q of the virtual resources of class c_j. Then, the virtual resources of class c_i with assigned value val$_x$ cannot be arranged with any resources of c_j. These constraints are deadlock constraints and tenants need to handle them properly.

- **Redundant CVRM Expressions:** In CVRM, an expression is redundant if it specified multiple times. Redundant expression unnecessarily increases the runtime complexity since it requires evaluation of the same expression more than once. One such example of a redundant expression is the multiple occurrences of same sub-expression in a constraint expression.

4.3 Prototype Implementation

We describe the implemented prototype of CVRM enforcement process. We leverage the DevStack cloud framework [2], a quick and stand-alone installation of OpenStack, for the implementation and analysis. We choose DevStack as it provides all components of the open source cloud platform OpenStack. We installed DevStack in a physical server that has 4 cores and 3GB RAM. We implemented the CVRM components for nova. The implemented component can specify attributes and their scopes for VMs and NETs which are stored in the database. It also has a process to specify and evaluate constraints for VM-NET connection. Our python-based implementation of constraint specifier, that includes API design to enable users to declare attributes and constraints which require tables creation into DevStack database (MySQL), has 190 lines of sqlalchemy code. The constraint enforcer that includes the constraint parser has 257 lines. The parser returns true or false value based on a constraint expression by considering assigned attribute values for a VM and NET which need to be connected.

5. CONSTRAINT MINING IN CVRM

In this section, we consider approaches for mining CVRM constraints from already specified relations among the instances of two classes of virtual resources. Basically, this process generates a collection of restricted rules, also refereed as min_rule, where a min_rule is an implication, a→b, in which both a and b are single predicates. Note that, the actual rule in the form of A→B as defined in section 3.1 allows both A and B to be collections of predicates connected by ∧ and/or ∨. Now for a given $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, $\delta^{\mathcal{A}dd}_{\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}}$ and $\delta^{\mathcal{RM}}_{\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}}$, a min_rule can be generated by following grammar,

<Quantifier>:= $\forall$(vr1,vr2) $\in \mathcal{R}_{<\text{Cls}>,<\text{Cls}>}$. <Stmt>
<Stmt>:= <Stmt> <connector><Stmt> | (<min_rule>)
<min_rule>:= <Token> → <Token>
<Token>:= <Attribute>(<resource>)<comperator><Scope>
<Attribute>::= <letter> | <digit> | <Attribute>
<Scope>::= <letter> | <digit> <Scope>
<connector>::= $\wedge$ | $\vee$
<comperator>::= = | $\neq$
<Cls>::= c_1 | c_2 | . . . | c_n
<resoruce> ::= vr1 | vr2
<digit>::= 0|1|2| . . . |8|9
<letter>::= a|b|. . . |y|z|A|B|. . . |Y|Z

Each min_rule is restricted to specify a comparison between only two attributes of virtual resource classes c_i and c_j. Now let us say $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ is a given set of tuples that specifies the relation between instances of the two classes

c_i and c_j. The min_rule mining problem is to construct all possible min_rules. For given $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, $\mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, $\mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, $\mathrm{att}_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, $\mathrm{att}_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, SCOPE_{att_p} and SCOPE_{att_q}, min_rules can be of four following formats, where each val has to belong to the appropriate attribute SCOPE_{att}.

- $\mathrm{a}\rightarrow\mathrm{b}$ where $\mathrm{a}\equiv(\mathrm{att}_p(\mathrm{vr1})=\mathrm{val}_x) \wedge \mathrm{b}\equiv(\mathrm{att}_q(\mathrm{vr2})=\mathrm{val}_y)$.
- $\mathrm{a}\rightarrow\overline{\mathrm{b}}$ where $\mathrm{a}\equiv(\mathrm{att}_p(\mathrm{vr1})=\mathrm{val}_x) \wedge \overline{\mathrm{b}}\equiv(\mathrm{att}_q(\mathrm{vr2})\neq\mathrm{val}_y)$.
- $\overline{\mathrm{a}}\rightarrow\mathrm{b}$ where $\overline{\mathrm{a}}\equiv(\mathrm{att}_p(\mathrm{vr1})\neq\mathrm{val}_x) \wedge \mathrm{b}\equiv(\mathrm{att}_q(\mathrm{vr2})=\mathrm{val}_y)$.
- $\overline{\mathrm{a}}\rightarrow\overline{\mathrm{b}}$ where $\overline{\mathrm{a}}\equiv(\mathrm{att}_p(\mathrm{vr1})\neq\mathrm{val}_x) \wedge \overline{\mathrm{b}}\equiv(\mathrm{att}_q(\mathrm{vr2})\neq\mathrm{val}_y)$.

For simplicity, we provide a mining algorithm for the format $\mathrm{a}\rightarrow\overline{\mathrm{b}}$ which is also referred as mutual exclusive min_rule. Similar algorithms can be generated for other formats. We choose mutual exclusive min_rule format because we develop mining algorithm on top of a constraint mining algorithm for role based access control [20] where they also mine mutual exclusive roles, so it is feasible to compare mutual exclusive min_rule to mutual exclusive roles.

5.1 Overview: Mining Constraints in RBAC

Mining association rules has become a fundamental problem in data mining, and it has been studied extensively. Many algorithms such as FP-growth, Apriori, and Eclat [4] have been developed to solve this problem in databases containing transactions. Recently, a constraint mining algorithm, called anti-Apriori, is proposed for role-based access control (RBAC) [20] which is developed on top of the Apriori algorithm [4]. In RBAC, U and R contains set of users and roles in the system. A function *user_roles* maps each user to a set of roles that are assigned to the user. Now the mutual exclusive constraint for RBAC is defined as follows.

A mutual exclusive RBAC constraint between roles $\in$ R is an implication of the form $\mathrm{R1}\rightarrow\overline{\mathrm{R2}}$ where $\mathrm{R1} \subset \mathsf{R}$ and $\mathrm{R2} \subset \mathsf{R}$ and $\mathrm{R1} \cap \mathrm{R2}=\emptyset$ and $user\_roles(\mathrm{u}) \subseteq \mathrm{R1} \rightarrow user\_roles(\mathrm{u}) \cap \mathrm{R2} = \emptyset$ for each user $\mathrm{u} \in \mathsf{U}$. Let D be a set of user-role assignments, the constraint $\mathrm{R1}\rightarrow\mathrm{R2}$ has confidence c if c% of users in U that are assigned a role in R1 do not have any role from R2, and it has support s if s% users are assigned a role in R1. The constraint $\mathrm{R1}\rightarrow\overline{\mathrm{R2}}$ holds for D if it has certain user-specified minimum support and confidence.

5.2 Mining min_rule in CVRM

In this section, we discuss the mining approaches for mutual exclusive min_rule. We first utilize the anti-Apriori algorithm [20] for mining min_rules. Then, we customize the anti-Apriori algorithm, which we call CVRM-Apriori, for min_rule mining in order to get better performance.

5.2.1 Reduction to RBAC constraint mining

In this approach, we identify inputs of a mutual exclusive min_rule mining algorithm and reduce them to the inputs of anti-Apriori. Then, we collect the outputs from anti-Apriori algorithm and construct min_rules.

Inputs of a min_rule mining algorithm: In CVRM, each mutual exclusive min_rule is restricted to specify a mutual exclusive relation between one value of an attribute of virtual resources of a particular class with another value of an attribute of virtual resources of another class. For given $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}$, $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j}$, and for each $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, the inputs for a mutual exclusive min_rule algorithm are $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}$, $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j}$, $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, att_p, att_q, SCOPE_{att_p} and SCOPE_{att_q}.

Inputs of anti-Apriori: The inputs of the anti-Apriori algorithm are U, R, the user-role assignment matrix M (M is a u×r dimension boolean matrix where u and r is the size of U and R and for each $\mathrm{u}_i \in \mathsf{U}$ and $\mathrm{r}_j \in \mathsf{R}$, $\mathrm{M}[\mathrm{u}_i][\mathrm{r}_j]=1$ if $\mathrm{r}_j \in user\_roles(\mathrm{u}_i)$ and 0 otherwise), matrix O where $\mathrm{O}=\overline{\mathrm{M}}$, minconf (minimum confidence) and minsup (minimum support). The inputs of the min_rule mining algorithm are reduced to the anti-Apriori algorithm as follows.

1. $\mathrm{U} = \mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i} \times \mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j}$ and $\mathrm{R} = \mathsf{SCOPE}_{att_p} \cup \mathsf{SCOPE}_{att_q}$. Without loss of generality, we assume the values in SCOPE_{att_p} and SCOPE_{att_q} are disjoint.

2. M is a $|\mathrm{U}| \times |\mathrm{R}|$ dimensional boolean matrix where, for each $\mathrm{u}\in\mathrm{U}$ and for each $\mathrm{r} \in \mathsf{R}$, $\mathrm{M}[\mathrm{u}][\mathrm{r}]=1$ where $(\mathrm{vr1},\mathrm{vr2})=\mathrm{u}$ and $\mathrm{att}_p(\mathrm{vr1})=\mathrm{r}$ or $\mathrm{att}_q(\mathrm{vr2})=\mathrm{r}$. Also, $\mathrm{O}=\overline{\mathrm{M}}$.

3. minconf and minsup are the values specified by the users.

Now, anti-Apriori generates constraints in following steps,

1. Scan M to find all combinations of $\mathrm{R}_i \subseteq \mathrm{R}$ in a set F where the support of R_i is greater than minsup.

2. Scan O to find all combinations of $\mathrm{R}_i \subseteq \mathrm{R}$ in a set $\overline{\mathrm{F}}$ where the support is greater than minsup.

3. For each $\mathrm{R}_i \in \mathrm{F}$ and for each $\overline{\mathrm{R}_j} \in \overline{\mathrm{F}}$, generate mutual exclusive rules in the format of $\mathrm{R}_i\rightarrow\overline{\mathrm{R}_j}$ if its confidence is greater than minconf and store $\mathrm{R}_i\rightarrow\overline{\mathrm{R}_j}$ in Rules.

min_rule Creation from Output of anti-Apriori algorithm: Output of anti-Apriori is a set of mutual exclusive rules from which the min_rules are constructed as follows, for each $\mathrm{R}_i\rightarrow\overline{\mathrm{R}_j} \in$ Rules, each $\mathrm{val}_x \in \mathrm{R}_i$ and $\mathrm{val}_y \in \overline{\mathrm{R}_j}$ construct a min_rule $att_p(\mathrm{v1})=\mathrm{val}_x \rightarrow att_q(\mathrm{v2})\neq\mathrm{val}_y$ where $\mathrm{val}_x \in \mathsf{SCOPE}_{att_p}$ and $\mathrm{val}_y \in \mathsf{SCOPE}_{att_q}$.

Although, this approach constructs min_rules, it lacks scalability for the following reasons.
(1) Size of the input parameter U is multiplicative with respective to the virtual resources of two different class since it is created by the cross product of each pair of virtual resources. It thereby makes the size of matrix M and O very large, increasing the run-time complexity.
(2) Algorithm anti-Apriori is designed to identify relations among all possible subset of roles, therefore, it needs multiple scans to database which is very costly. However, for mining the min_rules should require much simpler approach since it only needs to identify relations between every two values of two different attributes of the virtual resources.

5.2.2 Anti-Apriori for min_rule mining (CVRM-Apriori)

We customize the anti-Apriori algorithm for mining mutual exclusive min_rules. For a given $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}$, $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j}$, for each $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, a mutual exclusive min_rule between each $\mathrm{val}_x \in \mathsf{SCOPE}_{att_p}$ and $\mathrm{val}_y \in \mathsf{SCOPE}_{att_q}$ holds for an already specified $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ if it satisfies certain user-specified minsup and minconf. The support and confidence of a min_rule is calculated as follows,

- We define a function called $insideR^{att_p}_{tnt,c_i,c_j}$ that returns a set of elements in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that has a value val_x of an attribute $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$. Formally, $insideR^{att_p}_{tnt,c_i,c_j}(\mathrm{val}_x)$=$\{$(vr1,vr2) $\mid$ (vr1,vr2) $\in \mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j} \wedge att_p(\mathrm{vr1})=\mathrm{val}_x\}$.

- Another function called $outsideR^{att_q}_{tnt,c_i,c_j}$ returns the set of elements in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that does not have a value val_y of an attribute $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$. Formally, $outsideR^{att_q}_{tnt,c_i,c_j}(\mathrm{val}_y)$=$\{$(vr1,vr2) $\mid$ (vr1,vr2) $\in \mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j} \wedge att_q(\mathrm{vr2})\neq\mathrm{val}_y\}$.

- Now, a function called $support^{att_p}_{tnt,c_i,c_j}$ calculates support of a value val_x of an attribute $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$. Formally, $support^{att_p}_{tnt,c_i,c_j}(\mathrm{val}_x) = \frac{|insideR^{att_p}_{tnt,c_i,c_j}(\mathrm{val}_x)|}{|\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}|}$, that calculates the ratio of the number of tuples in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that contain val_x of the attribute $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ with all tuples in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$.

- Similarly, $support^{att_q}_{tnt,c_i,c_j}(\mathrm{val}_y) = \frac{|outsideR^{att_q}_{tnt,c_i,c_j}(\mathrm{val}_y)|}{|\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}|}$, is another function that calculates the ratio of the number of tuples in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that do not contain val_y of the attribute $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$ with all tuples in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$.

- Finally, a function called $confidence^{att_p,att_q}_{tnt,c_i,c_j}$ calculates the confidence which is the ratio of the number of elements in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that have a value val_x of an attribute $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, but, simultaneously, do not have a value val_y of an attribute $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$ with the total number of elements in $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$ that have a value val_x of an attribute $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$. Formally, $confidence^{att_p,att_q}_{tnt,c_i,c_j}(\mathrm{val}_x,\mathrm{val}_y) = \frac{|insideR^{att_p}_{tnt,c_i,c_j}(\mathrm{val}_x) \cap outsideR^{att_q}_{tnt,c_i,c_j}(\mathrm{val}_y)|}{|insideR^{att_p}_{tnt,c_i,c_j}(\mathrm{val}_x)|}$

Now, for a given $\mathcal{R}_{\texttt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, for each $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, user specified $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ and $min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$, algorithm 1 constructs the min_rules. In algorithm 1, procedure Identify_ Frequency identifies each attribute value $\mathrm{val}_x \in \mathsf{SCOPE}_{att_p}$ and each attribute value $\mathrm{val}_y \in \mathsf{SCOPE}_{att_q}$ whose supports satisfy $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ and returns them in sets F and $\overline{\mathrm{F}}$ respectively. Now, the Gen_min_rule procedure takes the sets F and $\overline{\mathrm{F}}$ and for each $\mathrm{val}_x \in \mathrm{F}$ and for each $\mathrm{val}_y \in \overline{\mathrm{F}}$ constructs the min_rules that satisfy the value of $min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$. This algorithm overcomes the scalability issues of anti-Apriori algorithm since it only identifies relations between two values instead of two subset of values of attributes, and F and $\overline{\mathrm{F}}$ are specified separately from the scopes of two different attributes.

Algorithm 1 Apriori Algorithm for min_rule Mining

```
procedure IDENTIFY_FREQUENCY($\mathsf{SCOPE}_{att_p}$,$\mathsf{SCOPE}_{att_q}$, $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$)
    $\mathrm{F} = \{\}, \overline{\mathrm{F}}=\{\}$
    for all $val \in \mathsf{SCOPE}_{att_p}$ do
        if $support^{att_p}_{tnt,c_i,c_j}(val) \geq min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ then
            $Insert\ val\ into\ F$
        end if
    end for
    for all $val \in \mathsf{SCOPE}_{att_q}$ do
        if $support^{att_q}_{tnt,c_i,c_j}(val) \geq min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ then
            $Insert\ val\ into\ \overline{\mathrm{F}}$
        end if
    end for
    $Return\ F\ and\ \overline{\mathrm{F}}$
end procedure
procedure GEN_min_rule($F, \overline{\mathrm{F}}, min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$)
    for all $val_x \in F\ and\ val_y \in \overline{\mathrm{F}}$ do
        if $confidence^{att_p,att_q}_{tnt,c_i,c_j}(val_x, val_y) \geq$
            $min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$ then
            Create_min_rule($\mathsf{min\_rule}_i, val_x, val_y$)
        end if
    end for
end procedure
```

5.3 Implementation and Analysis

We compare the performance of anit-Apriori and CVRM-Apriori algorithms. We implemented and evaluated both the mining algorithms, which are defined in 5.2, to construct min_rules for the add operation for VM-NET connectivity relations. We define three attributes for VM and two attributes for NET. The value of each attribute of the virtual resources is specified in their 'meta' information. We randomly connect 10 NETs to VMs where each VM is assigned to at-least 3 NETs. Then, we collect logs of VM-NET connection from the nova database of DevStack and evaluate both algorithms.

Our first experiment verifies scalability of the algorithms when number of VMs increases. We gradually increase VMs from 50 to 500 with a fixed size of scope of each attribute to 10 from which we randomly assign a value for each attribute of VMs and NETs. Then, for each VM attribute and NET attribute pair we separately execute both algorithms and record time. We repeated this process 10 times for each algorithm. Figure 8 shows the average execution time of both algorithms where time of anti-Apriori is very high while CVRM-apriori gives much better performance. For instance, for 50 VMs the average time of anti-Apriori is 1.3s where it is 14.2s for 500 VMs. On the other hand, in CVRM-Apriori, it is 0.23s and 1.2s. The reason is that the size of U of anti-Apriori is multiplicative with increasing number of VMs where in CVRM-Apriori it is only additive.

In second experiment, we fixed the VMs to 100, however, increase the scope of each VM attribute from 10 to 20 and executed both algorithms. We also executed each of them 10 times and recorded the time. Figure 9 shows the evaluation results. Note that, like experiment one, anti-Apriori gives very poor performance with compare to CVRM-Aprior. For instance, from 10 to 20 values in scope the required time of anti-Apriori increases 1.3s where, in CVRM-Apriori, it remains almost constant. The reason behind this is that anti-Apriori calculates mutual exclusive relations for all the combination of the values of two attributes which unnecessarily increases time since min_rule only needs to capture separate relations between each two values of attributes.

In general, CVRM-Apriori behaves similar to the 2-frequent Apriori algorithm which requires exactly 2 scans over the database, hence, the required run-time complexity of CVRM-

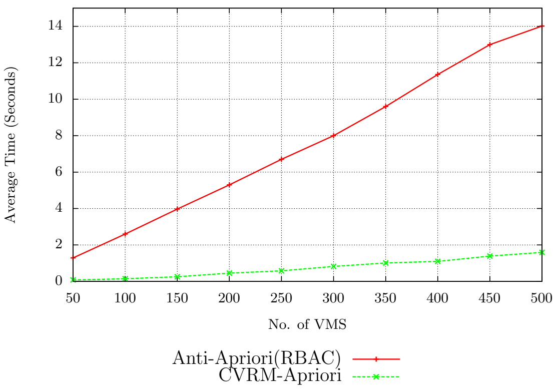


Figure 8: Mining Time with Increasing No. of VMs

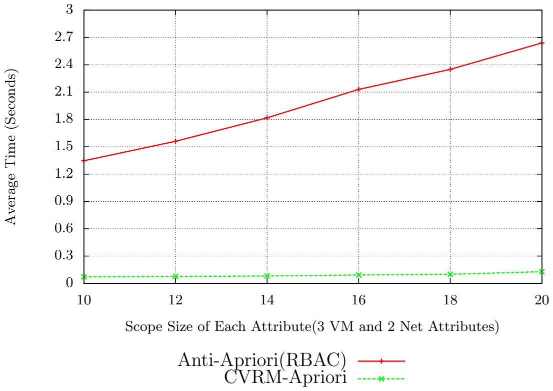


Figure 9: Mining Time with Increasing Scopes

Apriori is as good as FP-growth algorithm, which is an efficient Apriori algorithm with FP-tree data structure. Also, the accuracy of CVRM-Apriori is exactly same of the general Apriori algorithm since it does not discard any items from database for calculating the support and confidence.

6. RELATED WORK

Providing functionality to clients for resource-level permission management has started to receive more attention recently from cloud IaaS providers. However, this is primarily for managing user or group privileges to access their virtual resources. AWS Identity and Access Management (IAM) policies [1] now can construct fine-grained policies to control users' access to Subnets, VPCs, Security Groups and also type of virtual machines they can create. Also, the open source cloud platform OpenStack [3] has developed service called Keystone to manage users privilege to access cloud resources using some type of role-based access control. However, both platforms lack suitable mechanism so that clients can systematically specify policy to manage their virtual resources towards building a desired computing environment that addresses security, scale, hpc, etc. This increases various security threats for the running workloads from different tenants in cloud IaaS system. For instance, Shieh at al [26] shows that arbitrary sharing of network, in cloud, may cause denial of service attack and performance interferences. Wei et al [30] shows that uncontrolled snapshots and uses of images cause security risk for both creator and user of images. Sivathanu et al [27] presents an experimental analysis on I/O performance bottleneck when virtual storages are placed arbitrarily in physical storage and shared by random vms. Hence, different performance and security issues exist in cloud IaaS for unorganized multiplexing of resources and lack of controls, several of which are summarized in [14, 16, 17]. Hashizume et al [16] discuss and enumerates the security threats in cloud IaaS arising due to sharing physical machine, using images from public repository, sharing networks and storage, and also lack of proper resource control mechanism. Recently, for improving these scenarios, several efforts have been conducted by different groups of researchers. For instance, several improvements on shared network performance management have been proposed [5, 6, 26]. CloudNaas [6] provides better management of application-specific address spaces, middlebox traversal, bandwidth reservation, etc. Shieh at al [26] gives a bandwidth allocation scheme that allows infrastructure providers to define bandwidth sharing in cloud network with multiple tenants. Sivathanu et al [27] identifies four different factors that affects storage I/O performance and provides guidelines and their experimental analysis to minimize I/O overhead. Developing proper virtual machine placement algorithm also recently drew attention from research community [10, 11, 13, 15, 21, 31] for improving different aspects, e.g. high performance and load balancing. Present literature also contains several processes on users authorization and access control models for cloud IaaS that includes different RBAC models for cloud IaaS [9, 12].

Our contribution, in this paper, is unique and different than above described efforts. We aim to provide flexible mechanism to capture different requirements of the tenants to manage their virtual resources. Trusted virtual data center (TVDc) [7] is closely related to our work, where they assign virtual resources and users different colors where resources with similar colors can be combined to build a computing environment. Note that, color can be represented by an attribute, hence, CVRM is a generalization of TVDc in which resources are managed by multiple attributes.

7. CONCLUSIONS

We presented CVRM, the very first constraint specification process that enables tenants to specify several virtual resource management policies needed for production enterprise applications to run in IaaS clouds. CVRM can be specified as part of a cloud deployment, and are installed in the every cloud service provided by the IaaS providers.

We also identified that virtual-resource management policies can be discovered and constructed from log-file where it is similar to the well-known frequent-itemsets mining problem in database system. We demonstrated a constraint mining algorithm for CVRM where the algorithm leverages standard Apriori algorithm from the data mining literature. However, in this paper, we consider that the log-file is static and noise-free. Also, we do not analyze whether the mined constraints preserve semantic meaning with respective to the configuration requirements of the tenant. An obvious future work would be to identify various factors in IaaS that helps mining the rule with semantic meaning. Also, it will be interesting to determine noise in this system and develop a more dynamic mining algorithm that can eliminate such noises from the mining data.

8. ACKNOWLEDGEMENT

This research is partially supported by NSF Grants (CNS-1111925 and CNS-1423481). The authors thank Farhan Patwa and the anonymous reviewers for their helpful suggestions.

9. REFERENCES

[1] AWS identity and access management. *https://aws.amazon.com/iam/*.

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

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

[4] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of International Conference on Management of Data*, pages 207–216. ACM, 1993.

[5] H. Ballani, P. Costa, T. Karagiannis, and A. I. Rowstron. Towards predictable datacenter networks. In *Procedings of The ACM Special Interest Group on Data Communication*, pages 242–253, 2011.

[6] T. Benson et al. Cloudnaas: a cloud networking platform for enterprise applications. In *Proceedings of the 2nd Symposium on Cloud Computing*. ACM, 2011.

[7] S. Berger et al. Security for the cloud infrastructure: Trusted virtual data center implementation. *IBM Journal of Research and Development*, 53(4):6–1, 2009.

[8] K. Bijon, R. Krishnan, and R. Sandhu. A formal model for isolation management in cloud infrastructure-as-a-service. In *Proceedings of the 8th International Conference on Network and System Security (NSS)*. 2014.

[9] S. Bleikertz et al. Secure cloud maintenance - protecting workloads against insider attacks. In *Proceedings of the ACM Symposium on Information, Computer and Communications Security*, 2012.

[10] N. Bobroff et al. Dynamic placement of virtual machines for managing SLA violations. In *Proceedings of the International Symposium on Integrated Network Management*. IEEE, 2007.

[11] N. M. Calcavecchia, O. Biran, E. Hadad, and Y. Moatti. Vm placement strategies for cloud scenarios. In *Procedings of The International Conference on Cloud Computing*. IEEE, 2012.

[12] J. M. A. Calero et al. Toward a multi-tenancy authorization system for cloud services. *IEEE Security & Privacy*, 8(6):48–55, 2010.

[13] L. Cherkasova et al. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[14] W. Dawoud, I. Takouna, and C. Meinel. Infrastructure as a service security: Challenges and solutions. In *Procedings of International Conference on Informatics and Systems*, pages 1–8, 2010.

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

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

[17] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse. Security in multi-tenancy cloud. In *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, pages 35–41. IEEE, 2010.

[18] X. Jin, R. Krishnan, and R. Sandhu. A role-based administration model for attributes. In *Proceedings of the International Workshop on Secure and Resilient Architectures and Systems*. ACM, 2012.

[19] E. Keller, J. Szefer, J. Rexford, and R. B. Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 350–361, 2010.

[20] X. Ma, R. Li, Z. Lu, and W. Wang. Mining constraints in role-based access control. *Mathematical and Computer Modelling*, 55(1):87–96, 2012.

[21] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *Procedings of The International Conference on Cloud Computing Technology and Science*. IEEE, 2011.

[22] T. Ristenpart et al. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of The ACM Conference on Computer and Communications Security*, 2009.

[23] J. Rivera. Gartner identifies the top 10 strategic technology trends for 2014. *http://www.gartner.com/newsroom/id/2603623*, 2013.

[24] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[25] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 2000, 2000.

[26] A. Shieh et al. Sharing the data center network. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, 2011.

[27] S. Sivathanu, L. Liu, M. Yiduo, and X. Pu. Storage management in virtualized cloud environment. In *Procedings of The International Conference on Cloud Computing*, pages 204–211. IEEE, 2010.

[28] J. Szefer et al. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of The ACM Conference on Computer and Communications Security*, pages 401–412. ACM, 2011.

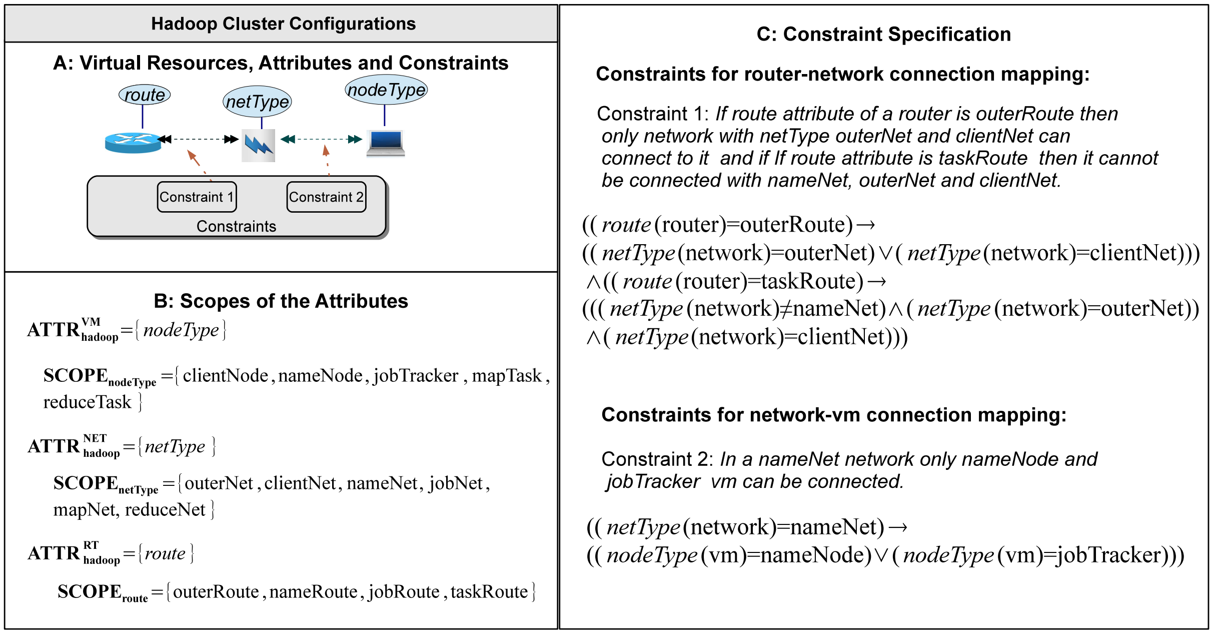
[29] V. Varadarajan et al. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *Proceedings of The ACM Conference on Computer and Communications Security*, pages 281–292, 2012.

[30] J. Wei et al. Managing security of virtual machine images in a cloud environment. In *Procedings of the ACM workshop on Cloud computing security*, 2009.

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

[32] F. Zhang et al. Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 203–216, 2011.

[33] Y. Zhang et al. Cross-vm side channels and their use to extract private keys. In *proceedings of the 19th ACM Conference on Computer and Communications Security*, 2012.


Hadoop Cluster Configurations
A: Virtual Resources, Attributes and Constraints
route
netType
nodeType
Constraint 1
Constraint 2
Constraints

B: Scopes of the Attributes

$\mathbf{ATTR}^{\mathbf{VM}}_{\mathbf{hadoop}} = \{ \mathit{nodeType} \}$

$\mathbf{SCOPE}_{\mathbf{nodeType}} = \{ \text{clientNode}, \text{nameNode}, \text{jobTracker}, \text{mapTask}, \text{reduceTask} \}$

$\mathbf{ATTR}^{\mathbf{NET}}_{\mathbf{hadoop}} = \{ \mathit{netType} \}$

$\mathbf{SCOPE}_{\mathbf{netType}} = \{ \text{outerNet}, \text{clientNet}, \text{nameNet}, \text{jobNet}, \text{mapNet}, \text{reduceNet} \}$

$\mathbf{ATTR}^{\mathbf{RT}}_{\mathbf{hadoop}} = \{ \mathit{route} \}$

$\mathbf{SCOPE}_{\mathbf{route}} = \{ \text{outerRoute}, \text{nameRoute}, \text{jobRoute}, \text{taskRoute} \}$

C: Constraint Specification

Constraints for router-network connection mapping:

Constraint 1: *If route attribute of a router is outerRoute then only network with netType outerNet and clientNet can connect to it and if If route attribute is taskRoute then it cannot be connected with nameNet, outerNet and clientNet.*

$$((\mathit{route}(\text{router}) = \text{outerRoute}) \rightarrow ((\mathit{netType}(\text{network}) = \text{outerNet}) \vee (\mathit{netType}(\text{network}) = \text{clientNet}))) \wedge ((\mathit{route}(\text{router}) = \text{taskRoute}) \rightarrow (((\mathit{netType}(\text{network}) \neq \text{nameNet}) \wedge (\mathit{netType}(\text{network}) = \text{outerNet})) \wedge (\mathit{netType}(\text{network}) = \text{clientNet})))$$

Constraints for network-vm connection mapping:

Constraint 2: *In a nameNet network only nameNode and jobTracker vm can be connected.*

$$((\mathit{netType}(\text{network}) = \text{nameNet}) \rightarrow ((\mathit{nodeType}(\text{vm}) = \text{nameNode}) \vee (\mathit{nodeType}(\text{vm}) = \text{jobTracker})))$$

Figure 10: Constraints Specification for Hadoop Cluster

Appendix: CVRM for Hadoop Cluster Setup

We discuss the CVRM instantiation for a simple hadoop cluster setup. The set TENANTS contains the tenant `hadoop`. The classes of the virtual resources supported by CSP are VM, NET, and RT and specified relations are between VM-to-NET and NET-to-RT. The relations are represented as $\mathcal{R}_{\texttt{hadoop},\mathsf{VM},\mathsf{NET}}$ and $\mathcal{R}_{\texttt{hadoop},\mathsf{NET},\mathsf{RT}}$.

In this simple hadoop setup, we only define one attribute for each virtual resources (shown in figure 10-A). Here, a VM attribute *nodeType* represent the type of operations a vm performs in `hadoop` cluster and figure 10-B shows the scope of *nodeType* that is clientNode, nameNode, jobTracker, mapTask, reduceTask. Similarly, two attributes *netType* and *route* are defined for NET and RT respectively. Note that, other attributes can also defined for more complex hadoop configuration management.

Followed by attribute specification of the resources, we show two constraints for adding elements in each of the relations (shown in figure 10-C). Here, for instance, constraint $\delta^{\text{Add}}_{\texttt{hadoop},\mathsf{NET},\mathsf{RT}}$ applies to the *Add* operation where it restricts all the NETs except outerNet and clientNet to connect a RT which has value outerRoute in *route* attribute. This constraint only allows clientNet to connect to outer internet.