**CONSTRAINTS FOR ATTRIBUTE BASED ACCESS CONTROL WITH APPLICATION IN CLOUD IAAS**

APPROVED BY SUPERVISING COMMITTEE:

Ravi Sandhu, Ph.D., Co-Chair

Ram Krishnan, Ph.D., Co-Chair

Weining Zhang, Ph.D.

Gregory White, Ph.D.

Shouhuai Xu, Ph.D.

Accepted:

Dean, Graduate School

## DEDICATION

*This dissertation is dedicated to my wife, Ms. Nilufar Ferdous, for her innumerable supports during the most demanding period of our life and, my parents who inspired me each step of the way with their unconditional love.*

# CONSTRAINTS FOR ATTRIBUTE BASED ACCESS CONTROL WITH APPLICATION IN CLOUD IAAS

by

KHALID ZAMAN BIJON, M. Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2015

## ACKNOWLEDGEMENTS

I would like to express my sincerest appreciation and profoundest gratitude to my advisors Dr. Ravi Sandhu and Dr. Ram Krishnan for their guidance and support throughout my doctoral studies. This dissertation would not be accomplished without their inspiring ideas, critical comments, and constant encouragement. I also thank Farhan Patwa, Associate Director of the ICS@UTSA, for his help and comments on the OpenStack implementation of the part of this dissertation.

I would like to extend my gratitude to other members of my committee, Dr. Rajendra Boppana, Dr. Greg White, and Dr. Shouhuai Xu, for their insightful comments and the time they devoted to reading the dissertation.

Finally, I would like to thank many colleagues and friends at the University of Texas at San Antonio for their help and support during all these years, which made my stay very enjoyable.

This work has been graciously supported by the National Science Foundation grant CNS-1111925 and CNS-1423481 and by the State of Texas Emerging Technology Fund.

*This Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.*

*It is acceptable for this Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Doctoral Dissertation attest to the accuracy of this statement.*

May 2015

**CONSTRAINTS FOR ATTRIBUTE BASED ACCESS CONTROL WITH APPLICATION IN CLOUD IAAS**

Khalid Zaman Bijon, Ph.D.
The University of Texas at San Antonio, 2015

Supervising Professors: Ravi Sandhu, Ph.D. and Ram Krishnan, Ph.D.

Recently, attribute based access control (ABAC) has received considerable attention from the security community for its policy flexibility and dynamic decision making capabilities. The general idea of ABAC is to determine the authorization decisions of an access request based on various *attributes* of the entities involved in the access (e.g., users, subjects, objects, context, etc.). Hence, in an ABAC system, proper assignment of attribute values to different entities is necessary to protect against unauthorized access. There has been considerable prior research for ABAC in various aspects such as formal models, enforcement models, policy composition languages and so on. However, mechanisms for ensuring proper attribute value assignments to entities have not been well studied.

In this dissertation, we propose a mechanism to specify and enforce constraints in ABAC that partially ensures proper assignment of attribute values to entities. We do so by specifying constraints on attribute values of a particular entity, so as to preserve various kind of conflicting relations between these values. We develop a declarative language called attribute-based constraint specification language (ABCL) for such constraints specification. During assignment of attribute values to entities, the mechanism enforces these specified constraints by prohibiting assignments that would violate one or more constraints. We validate expressiveness of ABCL by configuring several well-known constraint policies that include separation of duty and cardinality policies of the role based access control system. We also demonstrate the practical usefulness of ABCL by configuring various security policies for banking organizations. We discuss enforcement algorithms for ABCL and analyze their complexity.

We further devise a similar constraints specification mechanism in the concrete domain of cloud

infrastructure-as-a-service (IaaS). In cloud IaaS, both physical resources and virtual resources need to be mapped to each other in order to build a particular computing environment. Any misconfigurations in these mappings may result in potential security and performance losses. Unlike for attribute value assignment in ABAC, here, we generate constraints for ensuring proper mappings among cloud resources. Different properties of IaaS resources can be captured as attribute values where these values can have several conflicting relations that restrict how these resources can be mapped to each other. We identify customized versions of ABCL to specify such conflicting relations in cloud IaaS. In particular, we specify constraints for the following two mappings in cloud IaaS. (i) In cloud IaaS, a major problem for enterprise-scale tenants concerns orchestrating their virtual resources in a secure manner where they restrict any unwanted mapping between two virtual resources. We develop a constraints specification mechanism in order to restrict possible misconfiguration for such mappings, and demonstrate its implementation in the open source OpenStack cloud platform. We verify the expressiveness of the mechanism by configuring the mappings for 3-tier business applications and hadoop clusters setup. Also, we develop a constraint mining process in order to construct constraints automatically for the tenants according to their virtual resources mapping requirements. (ii) Another major concern arises from the tenants' lack of control on mapping of their virtual machines to physical servers operated by a cloud service provider. This limitation leads to many security and performance issues. We develop a virtual machine scheduler where the enterprises gain some controls by specifying constraints for this mapping. Our scheduler also optimizes the number of physical servers while satisfying the specified constraints. We analyze various performance and usability issues of the scheduler in OpenStack. Together, these two constraint mechanisms enable cloud tenants to maintain a level of control over their virtual assets in the cloud that is somewhat comparable to the level of control that was possible to maintain via their own premises.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# Chapter 1: INTRODUCTION

In general, constraints are an important and powerful mechanism for laying out higher-level security policies. For instance, in an organization, constraints can specify higher-level policies to put restriction on the behavior of its employees such as separation of duty constraints in role based access control whereby a particular employee cannot take both 'programmer' and 'tester' roles for the same project. Such a constraint eventually prevents the employee from simultaneously working on both developing and testing code for same project. In this dissertation, we develop constraints specification in attribute based access control (ABAC) and cloud infrastructure-as-a-service (IaaS).

Generally, ABAC regulates permissions of users or subjects to access system resources dynamically based on associated authorization rules with a particular permission. A user is able to exercise a permission on an object if the attributes of the user's subject and the object have a configuration satisfying the authorization rule specified for that permission. Hence, proper assignment of attribute values (or simply attribute assignment) to these entities is crucially important in an ABAC system for preventing unintended accesses. In this dissertation, we focus on constraints specification as an high-level policy specification on attributes assignment to entities in an ABAC based system as a mechanism to determine which entity should get which attribute values. By entities, we refer to users, subjects and objects which are common in access control systems. A user is an abstraction of a human being. A subject is an instantiation of a user and can refer to a particular session much like in role based access control (RBAC) and an object is a resource in the system. While ABAC is policy neutral, it is also complex to manage since its access management not only depends on authorization rules but also assigned attributes to the entities. Imposing constraints on attribute value assignments can mitigate this complexity by imposing centrally designed and configured constraints on the decentralized process by which specific attribute values are assigned to individual entities.

Moreover, when an organization migrate to an external cloud IaaS system, such higher level policies become the only means to capture required security requirements. For instance, before

moving to cloud, an organization (via its security architects for IT operations) specifies configuration policies for arranging its assets including establish networks among particular set of servers, backups and connecting a specific storage volume to a server, etc. However, when an organization moves to IaaS cloud, these resources become virtual and remote such as virtual machines (VM) and virtual networks (NET), so configuration management policies need to be similarly specified to arrange the organization's entire virtual resources. Since a cloud service provider (CSP) only allows it clients to specify higher level policies, a proper constraints specification process can provide an organization to specify their required security policies in a cloud IaaS system.

## 1.1 Motivation

We discuss the motivations for constraints specification in ABAC and cloud IaaS system.

### 1.1.1 Attribute Based Access Control

Over the last few years, attribute based access control (ABAC) has been emerging as a flexible form of access control due to its policy-neutral nature (that is, an ability to express different kinds of access control policies including DAC, MAC and RBAC) and dynamic decision making capabilities. Compared to these other access control models, ABAC is more complex to manage. Authorization of an access request in ABAC requires certain assigned attributes to the entities involved in the access. Hence, proper assignment of the attributes to entities needs to be ensured for protecting against unauthorized access. To this end, a constraints specification and enforcement mechanism can configure required attributes assignment policies for an organization. However, constraints specification in ABAC is more complex than that in other access control models such as RBAC since there are multiple attributes (unlike a single *role* attribute in RBAC) and attributes can take different structures (e.g., atomic or single-valued attributes such as *security-clearance* and *bank-balance* and set-valued attributes such as *role* and *group*). Constraints may exist amongst different values of a set-valued attribute (e.g. mutual exclusion on group memberships) and also on values across different attributes. For instance, suppose that an organization requires that only

its vice-presidents can get both a top-secret clearance and membership in their board-members email group. The ABAC system should have mechanisms to specify such constraints. In this case, there are three attributes for each user namely *role*, *clearance* and *group*. If the *role* attribute of a user is not 'vice-president', then the user's *clearance* and *group* attributes cannot take the value of 'top-secret' and 'board-member-emails' respectively. Note that these constraints are not concerned about users' access to objects directly. Instead, they focus on high-level requirements that a security architect would specify, which will indirectly translate into enabling or disabling certain accesses.

In general, the more expressive power a model has, the harder it is (if at all possible) to carry out many types of security analysis. It has already been shown that the safety problem of an ABAC system with infinite value domain of attributes is undecidable [131]. Nevertheless, ABAC is the leading mechanism that overcomes the limitations of discretionary access control (DAC) [107], mandatory access control (MAC) [106] and role-based access control (RBAC) [78]. NIST recognizes that ABAC allows an unprecedented amount of flexibility and security that makes it a suitable choice for large and federated enterprizes relative to existing access control mechanisms [37]. Given that ABAC is known to be hard to analyze, constraints specification on attribute values is a powerful means to ensure that essential high-level access control requirements are met in a system that utilizes ABAC.

### 1.1.2 Cloud Infrastructure-as-a-Service

When an organization moves to cloud IaaS, two major issues emerge. (i) In a CSP, a customer, also referred to as a tenant, is forced to re-think their access control and security policies in terms of configuration management facilities offered by the CSP. (ii) Multi-tenancy, availability and reliability are some of the major concerns for customers in IaaS cloud. For example, if a customer is concerned about co-location of their virtual machine in the same physical server with, say, some other competing tenants, it would be desirable to state this requirement. However, currently, there is no simple way for a customer to realize this policy. Our motivation is to investigate and

develop robust, flexible, and intuitive virtual resource configuration management mechanisms for infrastructure as a service (IaaS) cloud — one important piece of the overall cloud security puzzle.

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 1.1 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 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. Generally, the PR-to-PR mappings are completely managed by the CSPs with very little information to tenants.

There are at least two other type of mappings in cloud IaaS where it would be desirable for tenants to have some level of control by means of constraints. A 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 shown in figure 1.1 as VR-to-VR mappings. 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 to a sensitive internal network or a low-assurance image may be applied to a VM that is expected to be security-hardened. All major CSPs including Amazon [3] acknowledge that managing such configurations is beyond their responsibilities, rather they should be managed by individual tenants. However, current CSPs fail to offer suitable tools in this regard. This dissertation presents the design, implementation, and evaluation of attribute-based CVRM (constraint-driven virtual resource management) as an approach to mitigate such concerns in cloud IaaS.

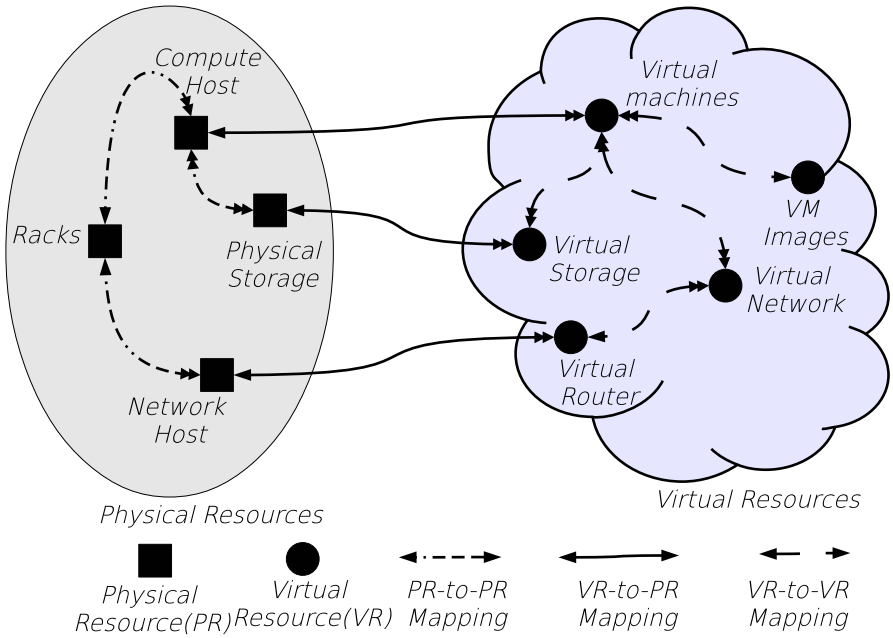


**Figure 1.1**: Cloud Resources Mapping Relation

Furthermore, 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 1.1. This raises many security and performance considerations for a tenant's workload in the cloud. For instance, a hypervisor on a host may be inadvertently misconfigured by a cloud administrator enabling leakage of data between virtual machines on that host that should otherwise be isolated. More problematically, 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 [98, 120, 132, 133] 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) [98].

Our objective is to address this goal where the tenants and the cloud system provider are able to schedule virtual resources on physical resources consistent with high-level and fine-grained constraints. In this respect, even the leading IaaS service providers currently offer minimal support to their tenants. In particular, tenants have very little influence on 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 fault tolerance but does not concern co-location of a tenant's resources with those of others in a given physical server or a rack. This dissertation explores a highly dynamic and fine-grained technique for scheduling virtual resources based on high-level constraints specified by tenants. It is worthwhile to note that multi-tenancy concerns can arise even in private cloud scenarios involving a single large-scale enterprise due to various reasons such as the need to separate the resources of various departments within that enterprise for assurance or compliance reasons. Similar to entities in ABAC, different properties of the resources of cloud IaaS can be captured as attributes where the attributes can have several conflicting relations that restrict how those resources to be mapped with each other. This dissertation develop constraints specification mechanisms for cloud IaaS that capture various conflicting relations among attributes of the resources in cloud and restrict improper mappings accordingly.

## 1.2 Thesis

The central thesis of this dissertation is as follows:

*Attributes can capture various high-level properties of entities and objects in a system and these attributes preserve specific conflicting relations with each other. By exploiting this fact, a suitably devised attribute based constraints specification mechanism can provide effective and expressive capabilities in laying out higher-level security policies for a traditional organization*

*that exercises attribute based access control as well as for the mapping configuration management of virtual resources in cloud infrastructure-as-a-service.*

## 1.3 Summary of Contributions

The major contributions of this research are stated as follows:

- We develop an attribute based constraint specification language (ABCL) for specifying constraints on attributes assignment. ABCL provides a mechanism to represent different kinds of conflicting relations amongst attributes in a system in the form of relation-sets. Relation-sets contain different attribute values and ABCL *expressions* specify constraints on attributes assignment based on these values. There is considerable literature, such as [53, 64, 73, 83, 101, 109, 121], on the utility of attributes in managing various aspects of security in a system. Our work is the first investigation on how attributes themselves could be managed based on their intrinsic relationships. We show that ABCL can express many types of constraints including those that can be expressed using the role-based constraint language RCL-2000 [12] and those supported by the NIST standard RBAC model [57]. We demonstrate the usefulness of ABCL in different usage scenarios such as in banking and cloud computing application domains. We also discuss ABCL enforcement and its performance.

- We develop the CVRM (constraint driven virtual resource management) framework that enables tenants to express several essential properties of cloud resources as their attributes and specify constraints on resource mappings (VR-to-VR in figure 1.1) based on those attributes. We provide a customized language of ABCL for constraints specification which is suitable for this purpose. We expect such constraints to be specified by a tenant administrative user. We provide a number of examples illustrating the utility of this technique in practical situations, such as configuring 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 to another. We demonstrate a comprehensive enforcement process of the CVRM

and analyzes security issues of the enforcement process. We provide a detailed implementation of it in the widely-deployed OpenStack, the open-source IaaS cloud software. The implementation process includes new API declaration, database schema design and identifies different types of attributes in a cloud IaaS system including attributes specific to a tenant and system attributes across multiple tenants. Finally, we develop a system automated constraint construction process in which the tenants specify the necessary attributes according to their business specifications and the required constraints are automatically constructed. This construction process includes a novel policy mining algorithm which is designed specifically for the cloud IaaS system and a process to identify policies which are semantically meaningful with respect to the virtual resources configuration requirements of a tenant in cloud IaaS.

- Finally, we present a design of an attribute-based framework for specifying co-location constraints of virtual resources scheduled on given physical resources. Given that co-location constraints can drastically affect physical resource utilization, we propose a host optimization process while enforcing constraints. 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 [19]. We establish that, in general, the algorithms for host optimization while enforcing such constraints are NP-Complete. 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. We develop a prototype of the conflict-free virtual machine scheduling framework in OpenStack [6] and rigorously evaluate the framework on various aspects, e.g., resource requirements, resource utilization, etc. We analyze issues that arise due to the incremental changes of conflicts over time.

## 1.4 Organization of the Dissertation

Chapter 2 gives a brief background on constraints specification and reviews related work on attribute based systems and cloud IaaS. Chapter 3 presents constraints specification in ABAC. In chapter 4, we provides the foundation of attribute-based constraints specification in cloud IaaS where we provide an example of ABCL configuring security policies in cloud IaaS. This chapter also develops a simple attribute based isolation management system in cloud IaaS where we show that attributes can represent various properties of users and resources in cloud which can be utilized for specifying policies for managing isolation. Then, in chapter 5, we develop a constraint-driven virtual resource orchestration which provides a customized version of ABCL which is more specific for IaaS and an enforcement of the constraints with experimental analysis. This chapter also includes an automated construction process of the constraints. In Chapter 6, we present and analyze our developed constraint aware virtual resource scheduling mechanism. Chapter 7 summarizes the completed research and discusses future work.

# Chapter 2: BACKGROUND AND LITERATURE REVIEW

In this chapter, we provide background on constraints specification in access control systems and in cloud IaaS. We also provide brief overview of existing literature on traditional access control systems, attribute based systems and cloud IaaS that are related to this dissertation.

## 2.1 Overview of Traditional and Existing Access Control Models

Access control has always played a vital role in the security of a computing system. The earliest access control approaches consist of discretionary access control (DAC) [107] and mandatory access control (MAC) [106]. DAC enforces control over resources within a system as per resources' owners discretion. Essentially, the owner is responsible for specifying in which manner a particular resource is accessible to specific system users. MAC enforces the control over resources in a partial-ordered lattice of labels and clearances assigned to users and resources. The access is specified through read and writes rules according to the relations between these labels and clearances. Over time, systems have adapted to new demands and evolved. As a result, access control mechanisms have also been required to follow suit. Role-based access control [104, 107] (RBAC) has been a popular authorization solution in enterprise software and systems. The use of role constructs facilitates permissions and users management.

Apart from the above mainstream approaches on access control, other variations of access control mechanisms exist within the literature. Notably, the following access control mechanisms are related to the research in this dissertation.

Several approaches have been proposed for combining risk issues in different access control systems. In those approaches risks are quantified and gives more dynamic capabilities. Bijon et al [30] provide a generic framework for risk-aware role based access control where they propose a guideline to incorporate risks around various components of a role based access control system such as user role assignment, role activation, etc. A quantified risk-aware RBAC sessions and role activation/deactivation framework have been proposed in [29].

Provenance-based access control (PBAC) is capable of capturing, storing, and providing such information, as provenance data, to make access control decisions. Specially Nguyen et al [91,94], show that Provenance data essentially forms a direct-acyclic graph and provides a linkage structure of history information of any data object of interest. This characteristic enables and facilitates a traversal capability on provenance data from which appropriate access control decision can be made. For instance, Nguyen et al [92] show that PBAC can be used to effectively enforce the dynamic separation of duty.

Relationship-based Access Control (ReBAC) for online social graphs can also be built on path patterns of relation edges. Specifically, Cheng et al [45] specify policies that utilizes regular expression-based path patterns of relationship types between graph entities such as users and resources for finer-grained and more expressive access control on online social networks. This generic ReBAC model is later extended to capture various user-to-user, user-to-resource, or resource-to-resource relationships [44, 46].

Besides above described conventional access controls, there exists different models for secure organizational information sharing. For instance, two different approaches have been proposed for the organizations that exercise LBAC to securely collaborate with outside specialist/consultants for certain reasons [27, 28]. Also, an administrative model has been proposed in [103] for the group-centric secure information sharing for community cyber security.

## 2.2 Attribute Based Systems

### 2.2.1 Attribute Based Access Control

There is a sizable literature on ABAC in general. Damiani et al [53] described a framework for ABAC in open environments. Wang et al [121] proposed a framework that models an ABAC system using logic programming with set constraints of a computable set theory. The Flexible access control system [73] can specify various ABAC policies and provide a language that permits the specification of general constraints on authorizations. Yuan et al [129] described ABAC in the

aspects of authorization architecture for web services. Lang et al [83] provided informal configuration of DAC, MAC, and RBAC through ABAC in the context of grid computing. These authors seek to develop an access control system either for open systems such as web, Internet, etc., or to overcome the limitations of conventional access control models by utilizing attributes. Park et al [95] categorized attributes according to their mutability during execution of operations and developed a mechanism in which attributes of entities can be updated as a side-effect of an access. More recently, Jin et al [78] proposed an attribute based access control model in which they provide an authorization policy specification language and formal framework using which DAC, MAC and RBAC policies can be expressed. These works focus on ABAC in general and not so much on constraints specification on attributes assignment in ABAC.

### 2.2.2 Attribute Based Encryption

This body of literature concerns cryptographic enforcement mechanisms for attribute based access control systems. Sahani et al [101] introduced the concept of Attribute Based Encryption (ABE) in which an encrypted ciphertext is associated with a set of attributes, and the private key of a user reflects an access policy over attributes. The user can decrypt if the ciphertext's attributes satisfy the key's policy. Goyel et al [64] improved expressibility of ABE which supports any monotonic access formula and Ostrovsky [93] enhanced it by including non-monotonic formulas. Several other works examine different ABE schemes.

## 2.3 Constraints Specification

Several authors have focussed on issues in constraints specification in access control systems primarily in RBAC. Constraints in RBAC are often characterized as static separation of duty (SSOD) and dynamic separation of duty (DSOD). These two issues date back to the late 1980's [48], [102]. A number of subsequent papers identify numerous forms of SSOD and DSOD policies [56, 111]

and their formal specification them formally in RBAC systems [61, 71]. The RCL-2000 language for specifying RBAC constraint policies in a comprehensive way was proposed by Ahn et al [12]. Object constraint language (OCL) [123] is also another well-known language for specifying constraints. This language is basically designed to specify constraints in Unified Modeling Language (UML). UML is a general-purpose modeling language in which we can specify, visualize, and document the components of software systems. OCL can specify various type of constraints including SSOD and DSOD constraints of RBAC which are formally modeled in [113]. More recently, Jin et al [78] proposed an attribute based access control model in which they provide an authorization policy specification language that could also specify constraints on attributes assignment. However, their constraints specification focuses on what values the attributes of subjects and objects may take given that users are currently assigned with particular attribute values. This is much like constraints on what roles can be activated in a user's session in RBAC given that a user is pre-assigned to a set of roles. Thus, prior work does not address ABAC constraints comprehensively. In this dissertation, we have shown that ABCL can specify various types of constraints for configuring several of these RBAC constraints, including those expressible by RCL-2000 [12].

## 2.4 Policy Specification in cloud Infrastructure-as-a-Service

### 2.4.1 Related to Virtual Resources Mapping Configuration Management

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 [2] now can construct fine-grained policies to control users' access to Subnets, Virtual Private Clouds, Security Groups and also type of virtual machines they can create. Also, the open source cloud platform OpenStack [6] has developed service called Keystone to manage users privilege to access cloud resources using a variation 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 [110] shows that arbitrary sharing of network, in cloud, may cause denial of service attack and performance interferences. Wei et al [124] shows that uncontrolled snapshots and uses of images cause security risk for both creator and user of images. Sivathanu et al [112] 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 [54, 69, 76]. Hashizume et al [69] 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 [15, 17, 110]. CloudNaas [17] provides better management of application-specific address spaces, middlebox traversal, bandwidth reservation, etc. Shieh at al [110] gives a bandwidth allocation scheme that allows infrastructure providers to define bandwidth sharing in cloud network with multiple tenants. Sivathanu et al [112] identifies four different factors that affects storage I/O performance and provides guidelines and their experimental analysis to minimize I/O overhead. Present literature also contains several processes on users authorization and access control models for cloud IaaS that includes different RBAC models for cloud IaaS [31, 39, 126].

### 2.4.2 Related to Virtual Resource Scheduling

Generally scheduling problems are NP-Complete. However, various heuristic and approximate approaches have been well studied by the research community. 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 analy-

sis [34, 82] which proposes heuristics, such as first-come-first-served, and shortest operation first, to minimize average delay. Another scheduling problem is to minimize number of bins, while scheduling a number of finite items in them. This problem is called bin packing. There are single and multi capacity bin packing problems based on multiple requirements for scheduling [84]. Multi-capacity bin packing is also applied in resource scheduling in grid computing [114, 115]. 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 dissertation. Several bin-packing with conflict algorithms [74, 75] have been proposed where it is assumed that items can be conflicting in random manner. However, we investigate 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 items have conflicts.

Different performance and security issues exist in cloud IaaS for unorganized multiplexing of resources, which are summarized in [54, 69]. Recently, articles have been published exposing the vulnerability of state-of-art co-residency system in public cloud IaaS system [132]. 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 dissertation. Developing proper VM placement algorithms recently drew attention from the research community. Bobroff et al [32] propose an algorithm that proactively adapts to demand changes and migrates virtual machines between physical hosts. Yang et al [128] also propose a load-balancing approach in VM scheduling process. Calcavecchia et al [38] develop a process to select candidate physical host for a VM by analyzing past behaviors of a physical host and deploy the request, while Gupta et al [67] propose a process for scheduling HPC related VMs together. Li et al [85] propose VM-placement that maximizes a physical hosts cpu and bandwidth utilization. Also, Mastroianni et al [87] propose a probabilistic approach for VM scheduling for maximizing CPU and RAM utilization of the physical host. 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.

## 2.5 Overview of OpenStack Architecture

In recent years, the popularity and wide use of the open-source OpenStack project [7] has made it a mature platform on par with mainstream proprietary cloud management platforms such as Amazon Web Services, Google Compute Engine, and Microsoft Azure Infrastructure Services, to name a few. This section provides an overview of the OpenStack architecture and several of its components. Cloud computing consists of three primary service models: Software-as-a-Service, Platform- as-a-Service, and Infrastructure-as-a-Service (IaaS). Each of the service model type provides different types of resources that can be shared and used by consumers. The OpenStack platform provides IaaS, which mainly deals with virtual resources that include virtual networks, virtual machine images and instances. Other OpenStack components also provide other services that relate to the mentioned virtual resources, i.e. monitoring resources usage and graphical user interface. As depicted on OpenStack website, the logical architecture includes the following components.

- Nova: provides an API for controlling cloud computing resources and managing the consumers of those resources.
- Glance: provides an API for management of virtual machine images.
- Swift: provides an API for object storage of virtual resources.
- Heat: provides an API for cloud applications orchestration.
- Cinder: provides an API for block storage of virtual resources.
- Neutron: provides an API for defining network connectivity in the cloud.
- Keystone: provides an API for maintenance of users' information and identity for authentication and authorization purposes.

- Ceilometer: provides an API for monitoring and collecting information on the movements and usages of virtual resources.

In this dissertation, the focus is on enabling constraint driven virtual resource management process in OpenStack, specifically, we implement the prototype for virtual machine management process that is developed on Nova service component.

# Chapter 3: THE ABCL MODEL

*The materials in this chapter are published in the following venues [20, 22]:*

1. Khalid Bijon, Ram Krishman, and Ravi Sandhu. Constraints specification in attribute based access control. ASE Science Journal, 2(3), 2013.

2. Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Towards an attribute based constraints specification language. In IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT), Washington, DC, September 8-14, 2013

This chapter describes our developed attribute based constraint specification language (ABCL) for specifying constraints on attributes assignment to entities and their enforcement in an attribute based access control system.

## 3.1 Motivation and Scope

Attributes can represent identities, security clearances and classifications, roles, as well as location, time, strength of authentication, etc. As such ABAC supplements and subsumes rather than supplants currently dominant access control models including DAC, MAC and RBAC. Figure 1 [78] shows a typical ABAC model structure that contains users (*U*), subjects (*S*), objects (*O*) and different permissions (P). There are also user attributes (*UA*), subject attributes (*SA*) and object attributes (*OA*) associated with users, subjects and objects respectively. A subject is the representation of a user's particular interaction with the system. Each permission is associated with an attribute-based authorization policy that determines whether a subject should get that permission on an object. An authorization policy compares the necessary subject and object attributes to make an authorization decision. Hence, proper attributes assignment to the entities is crucially important in ABAC.

Recently, an ABAC model called $\text{ABAC}_\alpha$ [78] proposed a policy specification language that could specify policies for authorizing a permission as well as constraints on attributes assignment. The constraints of $\text{ABAC}_\alpha$ are shown in the top row of figure 3.1 (horizontal solid lines with a

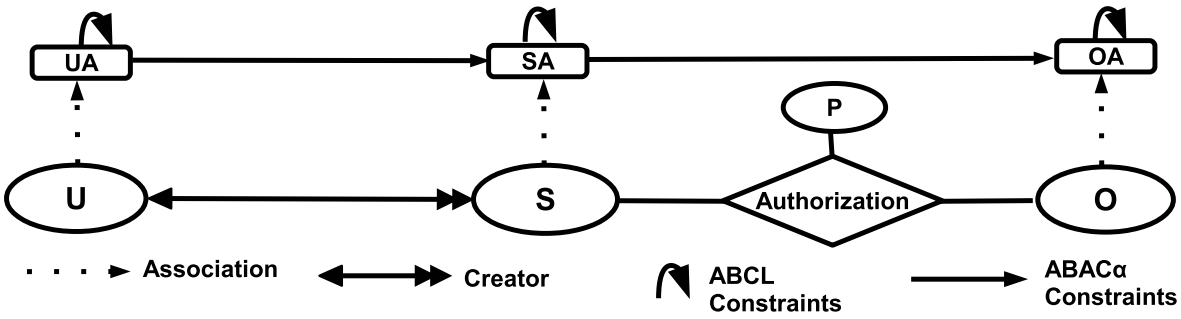


**Figure 3.1**: *ABAC model with ABAC$_\alpha$ and ABCL Constraints (adapted from ABAC$_\alpha$ [78])*

single arrow-head). These constraints apply to values a subject attribute may take when the subject is created, based on its owning user's attributes, or an object attribute may get when the object is created or operated-on by a subject. ABAC$_\alpha$ constraints apply only when specific events such as a user modifying a subject's attributes occur. In other words they are event specific. They relate the user attributes to the subject or the subject to the object depending on the event in question. ABCL constraints, on the other hand, are event independent and are to be uniformly enforced no matter what event is causing an attribute value to change. They are specified as restrictions on a single set-valued attribute or restrictions on values of different attributes of the same entity. ABCL constraints are depicted in the top row of figure 3.1 as arcs with a single arrow-head.

The central concept in ABCL is conflicting relations on attribute values which can be used to express notions such as mutual exclusion, preconditions and cardinality, amongst attribute values. For instance, suppose a banking organization utilizes a set-valued user (customer) attribute called *benefit* whose allowed values are {'bf$_1$', 'bf$_2$', ..., 'bf$_6$'}. Say that the bank wants to specify the following constraints: (a) a client cannot get both *benefits* 'bf$_1$' and 'bf$_2$', (b) a client cannot get more than 2 *benefits* from the subset {'bf$_1$', 'bf$_3$', 'bf$_4$'}, and (c) in order to get 'bf$_6$' the client first needs to get 'bf$_3$'. Here, the first policy represents a mutual exclusion conflict between 'bf$_1$' and 'bf$_2$', the second one is a cardinality constraint on mutual exclusion and the last one is an example of a precondition constraint. A number of other conflicts among attributes may also exist.

Figure 3.2 gives a hierarchical classification of the attribute conflict-relationships based on two parameters: the number of entities and the number of attributes of concern in a conflict relation.

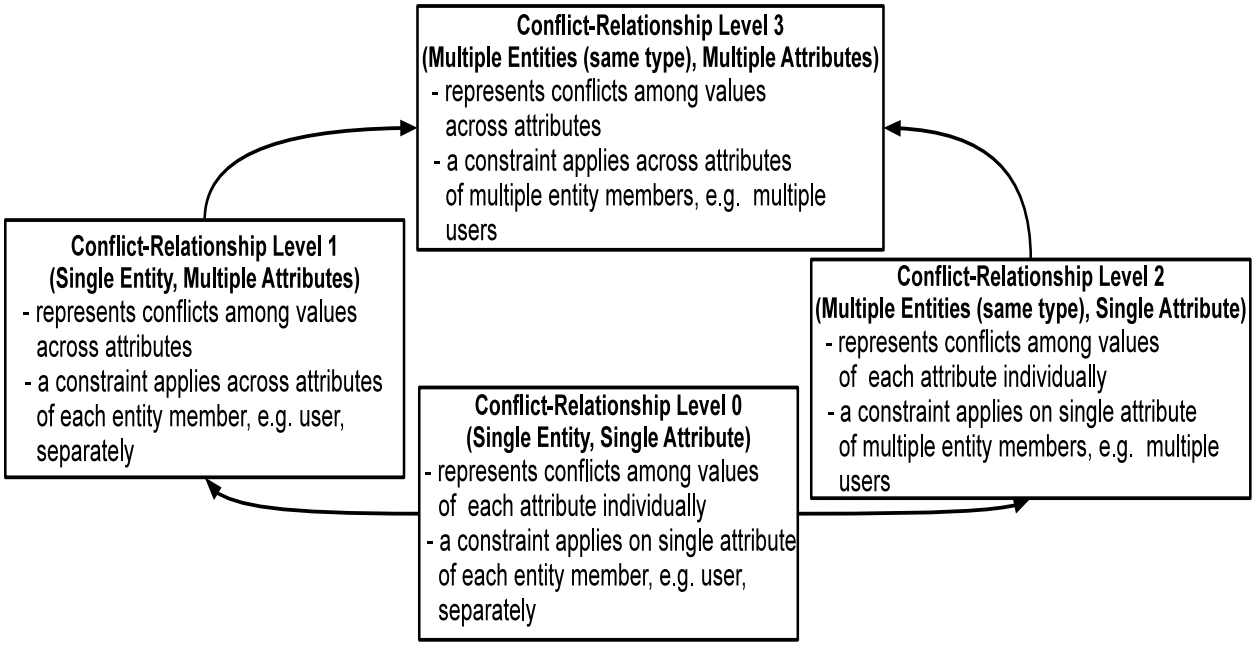


**Figure 3.2**: Attributes Relationship Hierarchy

For example for the user entity, each constraint in level 0 is concerned with conflicts among values of a single user attribute and it applies to each user independently. Level 1 allows constraints across values of different attributes of a single user. Level 2 constraints specify conflicting values of each attribute individually but across multiple users, while level 3 constraints can be across different attributes across multiple users. For instance, in the previous banking example, a constraint that disallows granting both *benefits* '$bf_1$' and '$bf_2$' to a client simultaneously is in level 0. Section 3.4.2 shows examples of several other constraints which fall in different levels of the relationship hierarchy. Further discussion of this hierarchical model and corresponding ABCL based functional requirements is given in section 3.3.1.

In the following sections, we present ABCL formalization and discuss them for user attributes in an ABAC model. However, ABCL is capable of expressing attributes assignment constraints of other entities as well, e.g. subject and objects. For simplicity, our examples focus exclusively on user attributes in the rest of this chapter.

## 3.2 Attribute Based Constraint Specification Language (ABCL)

We now formally present the elements of ABCL. ABCL consists of four basic components: the attributes of different entities in an ABAC model, a few basic sets and functions to capture different relationships amongst attributes, a few declared conflict sets and a language for specifying constraints using basic sets and functions and the declared conflict sets.

### 3.2.1 Basic Components of the ABCL Model

For the purpose of this research, we use the basic framework of the $ABAC_\alpha$ model [78] as a representative ABAC model for ABCL. However, note that ABCL is not tailored for $ABAC_\alpha$ and can be similarly applied to other ABAC models.

A brief overview of $ABAC_\alpha$ is provided in table 3.1. Like most access control models, $ABAC_\alpha$ consists of familiar basic entities: users (*U*), subjects (*S*) and objects (*O*). Each of these entities is associated with a respective set of attribute functions or simply *attributes* (*UA*, *SA* and *OA* respectively). Two types of attributes are considered in $ABAC_\alpha$, viz., set-valued and atomic-valued. For example, *role* is a set-valued attribute since a user may take multiple *roles* in an organization. However, *security-clearance* is an atomic-valued attribute since a user takes only a single value for security clearance such as 'top-secret' or 'secret'. As shown in table 3.1, an *attribute* is a function from the respective entity to a set of values that it can take (the **Range** of the *attribute*). The **Range** could be set or atomic-valued depending on the type of the attribute. A special attribute called **SubCreator** is used to keep track of the user that created a particular subject. Note that a user can create any number of subjects. The permissions that a subject can exercise on an object depends on the attribute values of the subject and object, and the attribute-based authorization rule specified for that permission in the system. Since ABCL is only concerned about constraints on what values the attributes can take and not on authorization rules for subject operations on objects or subject creation and other operations, the overview of $ABAC_\alpha$ provided in table 3.1 suffices for our purpose. For specifying ABCL constraints, we specify additional derived functions for convenience. For each attribute, we also define $\textbf{assignedEntities}_{U,att}$ (table 3.1) that identifies the set

**Table 3.1**: Basic sets and functions of ABAC

*U*, *S* and *O* represent finite sets of existing users, subjects and objects.

*UA*, *SA* and *OA* represent finite sets of user, subject and object attribute functions.

For each *att* in *UA* ∪ *SA* ∪ *OA*, **Range**(*att*) represents the attribute's range, a finite set of atomic values.

**SubCreator**: $S \rightarrow U$. For each subject it gives the creator.

**attType**: $UA \cup SA \cup OA \rightarrow$ {set, atomic}.
Given an attribute name, this function will return its type as either set or atomic.

Each attribute function maps elements in *U*, *S* and *O* to atomic or set values.

$$\forall ua \in UA.\ ua: U \rightarrow \begin{cases} \textbf{Range}(ua) \text{ if } \textbf{attType}(ua)\text{=atomic} \\ 2^{\textbf{Range}(ua)} \text{ if } \textbf{attType}(ua)\text{=set} \end{cases}$$

$$\forall sa \in SA.\ sa: S \rightarrow \begin{cases} \textbf{Range}(sa) \text{ if } \textbf{attType}(sa) = \text{atomic} \\ 2^{\textbf{Range}(sa)} \text{ if } \textbf{attType}(sa)\text{=set} \end{cases}$$

$$\forall oa \in OA.\ oa: O \rightarrow \begin{cases} \textbf{Range}(oa) \text{ if } \textbf{attType}(oa)\text{=atomic} \\ 2^{\textbf{Range}(oa)} \text{ if } \textbf{attType}(oa)\text{=set} \end{cases}$$

For convenience, we also use the following derived functions, for each $att \in UA$:

$\textbf{assignedEntities}_{U,\,att}$: **Range** (*att*) $\rightarrow 2^U$ where

$\textbf{assignedEntities}_{U,att}(attval) = \{u \in U \mid attval = att(u)\}$ if **attType**(*att*) = atomic

$\textbf{assignedEntities}_{U,att}(attval) = \{u \in U \mid attval \in att(u)\}$ if **attType**(*att*) = set

of users that are assigned a particular value of that attribute. Similar derived functions can also be defined for subjects and objects.

### 3.2.2 Syntax of ABCL

The syntax of ABCL is defined by the grammar in table 3.2 in Backus Normal Form (BNF). The grammar specifies declaration syntax for two type of relation-sets ***Attribute_Set*** and ***Cross_Attribute_Set***. The definition and structure of these relation-sets are given in the following section 3.2.3. The grammar also specifies syntax for constraint expressions.

ABCL includes two nondeterministic functions, *oneelement* and *allother*, adapted from [12, 42]. The function *oneelement*(*X*) nondeterministically selects one element $x_i$ from set *X*. In a constraint expression it is written as **OE**(*X*). Multiple occurrences of **OE**(*X*) in a single ABCL

**Table 3.2**: Syntax of Language

**Declaration of the *Attribute_Set* and *Cross_Attribute_Set*:**

<attribute_set_declaration> ::= <atribute_set_type> <set_identifier>

<attribute_set_type> ::= $\boldsymbol{Attribute\_Set}_{U,<\text{attname}>}$ | $\boldsymbol{Attribute\_Set}_{S,<\text{attname}>}$ | $\boldsymbol{Attribute\_Set}_{O,<\text{attname}>}$

<cross_attribute_set_type> ::= $\boldsymbol{Cross\_Attribute\_Set}_{U,<Aattset>,<Rattset>}$ |
$\boldsymbol{Cross\_Attribute\_Set}_{S,<Aattset>,<Rattset>}$ | $\boldsymbol{Cross\_Attribute\_Set}_{O,<Aattset>,<Rattset>}$

<*Aattset*> ::= {<attname>, <attname>$^*$}

<*Rattset*> ::= {<attname>, <attname>$^*$}

<set_identifier> ::= <letter> | <set_identifier><letter> | <set_identifier><digit>

<digit> ::= *0|1|2|3|4|5|6|7|8|9*

<letter> ::= *a|b|c|....|x|y|z|A|B|C|...|X|Y|Z*

**Constraint Expressions:**

<statement> ::= <statement> <connective> <statement> | <expression>

<expression> ::= <token> <atomiccompare> <token> | <token> <atomiccompare> <size>
| <token> <atomiccompare>|<set>| | <token> <atomiccompare> <set> | <token>

<token> ::= <token> <setoperator> <term> | <term> | |<term>|

<term> ::= <function> (<term>) | <attributefun> (<term>) | **OE** (<relationsets>).<item>
| **OE** (<term>) | **OE** (<set>) | **AO** (<term>) | **AO** (<set>) | <attval>

<connective> ::= $\wedge$ | $\Rightarrow$

<setoperator> ::= $\in$ | $\cup$ | $\cap$ | $\notin$

<atomicoperator> ::= + | < | > | $\leq$ | $\geq$ | $\neq$ | =

<set> ::= *U*| *S*| *O*

<relationsets> ::= <set_identifier>

<attname> ::= $ua_1$ | $ua_2$ | ... | $ua_x$ | $sa_1$ | $sa_2$ | ... | $sa_y$ | $oa_1$ | ... | $oa_z$

<attval> ::= ‘$\text{ua}_1\text{val}_1$’ | ‘$\text{ua}_1\text{val}_2$’ | ... | ‘$\text{ua}_x\text{val}_r$’ | ‘$\text{sa}_1\text{val}_1$’ | ‘$\text{sa}_1\text{val}_2$’ | ... | ‘$\text{sa}_y\text{val}_s$’ |
‘$\text{oa}_1\text{val}_1$’ | ... | ‘$\text{oa}_z\text{val}_t$’

<size> ::= $\phi$ | 1 | ... | N

<item> ::= limit| attval| attfun(<attname>).limit| attfun(<attname>).attval

<attributefun> ::= $ua_1$ | $ua_2$ | ... | $ua_x$ | $sa_1$ | $sa_2$ | ... | $sa_y$ | $oa_1$ | ... | $oa_z$

<function> ::= **SubCreator** | $\textbf{assignedEntities}_{U,<\text{attname}>}$ | $\textbf{assignedEntities}_{S,<\text{attname}>}$ |
$\textbf{assignedEntities}_{O,<\text{attname}>}$

expression select the same element $\text{x}_i$ from *X*. The function *allother*(*X*) returns a subset of elements from *X* by removing one element specified by the matching **OE**(*X*). We usually write *allother* as **AO**. These two functions are related by context. For any set *S*, {**OE**(*S*)}∪**AO**(*S*)=*S*. An example of **OE** in an ABCL expression is as follows.

**Requirement:** No user can get more than three benefits.

**ABCL Expression:** $|benefit(\textbf{OE}(U))| \leq 3$

**OE**(*U*) means a nondeterministically chosen single user from *U* and *benefit*(**OE**(*U*)) returns all benefits that are assigned to that user. This expression specifies that a single user cannot have more

**Table 3.3**: Declared ABCL Conflict Sets

**1. Sets that represent conflicts among values of a single attribute**
For each $att \in UA$ and **attType**($att$)=set there are zero or more conflict sets
***Attribute_Set***$_{U,att}$ = {avset$_1$, avset$_2$, ..., avset$_t$}, where
avset$_i$=(attval, limit) in which attval$\in 2^{\textbf{Range}(att)}$ and $1 \leq$limit$\leq$|attval|.

**2. Sets that represent conflicts across values of multiple attributes**
For each $Aattset \subseteq UA$ and $Rattset \subseteq UA$ there are zero or more conflict sets
***Cross_Attribute_Set***$_{U,Aattset,Rattset}$={attfun$_1$, ..., attfun$_u$}, where
for each $att \in Aattset \cup Rattset$
attfun$_i$($att$)=(attval, limit) in which
attval$\in 2^{\textbf{Range}(att)}$ if **attType**($att$)=set or
attval$\in$ **Range**($att$) if **attType**($att$)=atomic
and $0 \leq$limit$\leq$|attval|.

than three benefits. Later, we will see how **AO** is used in an ABCL expression.

### 3.2.3 Declared Conflict Sets of ABCL

Conflicts among attribute values can occur in several ways. ABCL recognizes two types of conflict: values that conflict with other values of the same attribute (referred to as single-attribute conflict) and values having conflict with values of other attributes (cross-attribute conflict). Note that single-attribute conflict is applicable only for set-valued attributes (e.g. mutual-exclusive roles) while cross-attribute conflict applies to both atomic and set-valued attributes. In order to specify these two types of conflict, ABCL specifies two type of sets that specify potentially conflicting values for single and cross-attribute conflicts respectively.

Items 1 and 2 in table 3.3 define these two sets for single-attribute and cross-attribute conflicts. As shown in item 1, each ***Attribute_Set*** contains a set of values of an attribute that may have a particular type of conflict (mutual exclusion, precondition, etc.). A separate ***Attribute_Set*** for each type of conflict could also be specified. Each element of an ***Attribute_Set*** is an ordered pair (attval, limit) where attval contains the values that have some form of conflict and limit specifies the cardinality, that is the number of values in attval for which the conflict applies. The semantic of limit for a particular ***Attribute_Set*** can be either at-least, exactly or at-most depending on the

constraint expression that uses the ***Attribute_Set***. For instance, the following are the declarations and initializations of two different ***Attribute_Set*** representing conflicting-relations between the values of *benefit* attribute of the banking example in section 3.1. The declaration syntax is shown in table 3.2.

***Attribute_Set***$_{U,benefit}$ *UMEBenefit*

*UMEBenefit*={avset$_1$, avset$_2$} where

avset$_1$=({'bf$_1$','bf$_2$'}, 1) and

avset$_2$=({'bf$_1$','bf$_3$','bf$_4$'}, 2)

***Attribute_Set***$_{U,benefit}$ *PreconditionBenefit*

*PreconditionBenefit*={avset$_1$} where

avset$_1$=({'bf$_3$', 'bf$_4$'}, 1)

Although these two sets structurally look identical, their purposes are different. *UMEBenefit* specifies the values of the *benefit* attribute that are mutually exclusive, hence, they cannot be assigned to a user simultaneously. As shown above, avset$_1$ in *UMEBenefit* indicates that the values 'bf$_1$' and 'bf$_2$' has limit value 1, therefore, at-most one value from them can be assigned to a user. Similarly, avset$_2$ indicates that 'bf$_1$', 'bf$_3$' and 'bf$_4$' has limit value 2, therefore, *benefit* of a user can get at-most two values from them. Note that, an ***Attribute_Set*** itself cannot specify the semantic meaning of the limit, rather, generated ABCL expression that uses this relation-set needs to specify it. The following ABCL expression is generated for this purpose,

**ABCL Expression:** $|benefit(\mathbf{OE}(U)) \cap \mathbf{OE}(UMEBenefit).\text{attset}| \leq \mathbf{OE}(UMEBenefit).\text{limit}$

Here, the expression restricts that each user can get at-most specified mutual exclusive values of *benefit* which are specified in each element of *UMEBenefit*. Similarly, in *PreconditionBenefit* for avset$_1$ the number of elements from attval,{'bf$_3$', 'bf$_4$'}, should be at-least the specified limit which is 1. Now, the following ABCL expression specifies for this purpose,

**ABCL Expression:** $|benefit(\mathbf{OE}(U)) \cap \mathbf{OE}(PreconditionBenefit).\text{attset}| \geq \mathbf{OE}(PreconditionBenefit).\text{limit}$

As mentioned earlier, there could also be conflicts amongst values across different attributes of a user. Let us say in the banking example of section 3.1, there is another user attribute called *felony* and its range is {'$fl_1$', '$fl_2$', '$fl_3$'}. The bank seeks to restrict a user to *benefit* '$bf_1$' if she has ever committed felony '$fl_1$' or '$fl_2$'. This is a mutual exclusive conflict relation among the values of *benefit* and *felony*. These relations are represented as another type of relation-set called ***Cross_Attribute_Set*** which is formally defined in table 3.3 item 2. Each ***Cross_Attribute_Set*** is declared for two arbitrary sets of user attributes which are determined at declaration time. These two sets of attributes are represented as *Aattset* and *Rattset* and combination of certain values of the attributes in *Aattset* as a group has specific type of conflicts with certain values of each attribute in *Rattset*. In other words, values of the attributes of *Aattset* together restrict the values of each attribute in *Rattset*. Each element of a ***Cross_Attribute_Set*** is a function called attfun that returns the values of the attributes of *Aattset* and *Rattset* as an ordered pair (attval, limit) where attval represents the values and limit is the cardinality. ***Cross_Attribute_Set*** declaration and initialization for the banking example are as follows (the syntax is shown in table 3.2).

***Cross_Attribute_Set***$_{U,Aattset,Rattset}$ *UMECFB*

Here, *Aattset*= {*felony*} and *Rattset*= {*benefit*}

*UMECFB*={$\text{attfun}_1$} where

$\text{attfun}_1$(*felony*)=({'$fl_1$', '$fl_2$'},1) and $\text{attfun}_1$(*benefit*)=({'$bf_1$'},0)

Using the set above, one can state that if at least one value from {'$fl_1$','$fl_2$'} is assigned to *felony* of a user, '$bf_1$' should not be assigned to *benefit* of that user. Similar to ***Attribute_Set***, the meaning of limit is not specified by the ***Cross_Attribute_Set*** but rather the constraint expression that uses it. The following expresses this constraint.

**ABCL Expression:**

$$|\mathbf{OE}(UMECFB)(felony).\text{attset} \cap felony(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECFB)(felony).\text{limit} \Rightarrow$$

$$|\mathbf{OE}(UMECFB)(benefit).\text{attset} \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMECFB)(benefit).\text{limit}$$

## 3.3 ABCL Enforcement

ABCL constraints are enforced during each attribute assignment to a user. As shown in algorithm 3.1, we design a simple procedure for the set-valued attribute assignment to users. The inputs of this algorithm are a user (*u*), a set-valued attribute (*att*) and a subset of values (attval) of $\mathsf{SCOPE}_{att}$ that can be assigned to *att* of u. In line-4, the procedure temporarily replace the already assigned values to *att* of the user *u*. Then, for each already generated constraint expressions (both single attribute and cross attributes), it calls a function called *Evaluate*, in line-6, to check whether the newly assignment satisfies them all. We assume, **ConsExprSet** contains all ABCL constraints for users. Each invocation of **ConsExprSet** returns a *true/false*. A *true* indicates that the requested attribute values can be assigned. If any of the constraints is not satisfied, the user will retain its old attribute assignment. Similar procedure can also be developed for the atomic-valued attribute.

In ABCL, constraints are generated only for the assignment of values to attributes. We do not consider constraint enforcement for the delete requests of already assigned values of attributes. However, such constraints can also be generated using ABCL and enforced accordingly. Also, the specified constraints may restrict an attribute assignment request for different reasons based on the conflict-relations of the requested values of an attribute with the values of same/other attributes. In this dissertation, we do not develop a system that automatically notifies the reason for denying a request. However, such a system can be built on top of our proposed enforcement mechanism which can identify the reason based on the constraint expression that evaluates to a *false*. Then, proper actions (assign/delete the values of attributes) can be taken so that the requested attribute value can be assigned. In the following, we list some of the reasons and the actions that can be taken.

- If the constraint that restricts the assignment request is generated for pre-condition, then assignment of the prerequisite values of attributes is required to resolve the issue. Note

**Algorithm 3.1** User Set-Valued Attribute Assignment

```
/*
   For an atomic-valued attribute, line-2 should be replaced by the following line
   if u∈U and att∈UA and attval∈Range(att) then
*/
1:  procedure AssignAttributetoUser(u, att, attval)
2:    if u∈U and att∈UA and attval⊆Range(att) then
3:      old_value ← att(u)
4:      att(u) ← attval
5:      for all cnst ∈ ConsExprSet do
6:        if Evaluate(cnst)=false then
7:          att(u) ← old_value
8:          Return false
9:        end if
10:     end for
11:     Return true
12: end procedure
```

that, if the constraint capture the conflict-relations among the values of single attribute, then prerequisite value is for same attribute. Therefore, the set of values of the attribute which is previously requested needs to be modified properly to resolve this. In cross-attribute conflict, the prerequisite values should be assigned to other attributes.

- In case of the constraints capturing mutual-exclusive relations, it is required to delete/remove already assigned mutual-exclusive values of the attributes. If the constraint captures single-attribute conflict, some values from the requested values of the attribute need to be removed in order to resolve the conflict. In cross-attribute conflict, the already assigned values of other attributes should be deleted.

Similar to RCL-2000 [12], ABCL constraints are represented in the form of restricted first order predicate logic (RFOPL) expressions for enforcement. RFOPL is a restricted version of FOPL that contains only universal quantifiers ($\forall$) where in each expression $\forall$ comes first followed by the predicates. The following is an example of an ABCL constraint and corresponding RFOPL expression:

**ABCL Expression:** $id(\mathbf{OE}(U)) \neq id(\mathbf{OE}(\mathbf{AO}(U))$

**RFOPL Expression:** $\forall u1 \in U, \forall u2 \in U\text{-}\{u1\}$: $id(u1) \neq id(u2)$

Here, **OE**($U$) and **OE**(**AO**($U$)) is converted to $\forall u1 \in U$ and $\forall u2 \in U\text{-}\{u1\}$ in RFOPL expression. The general structure of a converted RFOPL expression from ABCL is as follows:

**1)** The expression has a (possibly empty) sequence of universal quantifiers as a left prefix, and these are the only quantifiers.

**2)** The quantifier part will be followed by a predicate separated by a colon (:) (i.e., universal quantifier part : predicate).

**3)** The predicate has no free variables or constant symbols. All variables are declared in the quantifier part (e.g., $\forall u \in U$, $\forall cben \in MutualExlcusiveBenefit$, $\forall r \in role(u)$).

**4)** Predicate follows all rules in the syntax of ABCL except the term syntax in table 3.2. The syntax for term in RFOPL is as follows in which an element is a variable in quantifier part:

<term> ::= <function> (element) | <attributefun> (element)

| element.<item> | element | (<set>-{element}) | <attval>

A loop is created for each quantifier to traverse respective elements and the parser parses predicates of the expression. The following section discusses the ABCL enforcement complexity.

### 3.3.1 Constraints Hierarchy and Enforcement Complexity

We discuss the enforcement complexity of the ABCL constraints for each level in attribute conflict-relationship hierarchy (figure 3.3).

**Level 0** (Single User, Single Attribute): In this level, the system neither contains cross attribute relations nor constraints evaluating those relations. The system needs a set of users ($U$), ***Attribute_Set*** and functionality to evaluate properties of each user separately (**OE**). Here, a constraint enforcement complexity is $\mathcal{O}(N \times M \times P)$ where N is the number of users, M is the number of elements in respective ***Attribute_Set*** and P is number of predicates in the expression.

**Level 1** (Single User, Multiple Attributes): Conflict-relations among values across multiple attributes are specified and applied to each user separately. Besides the functionalities of relationship level 0, ***Cross_Attribute_Sets*** are needed in this level. The enforcement complexity is

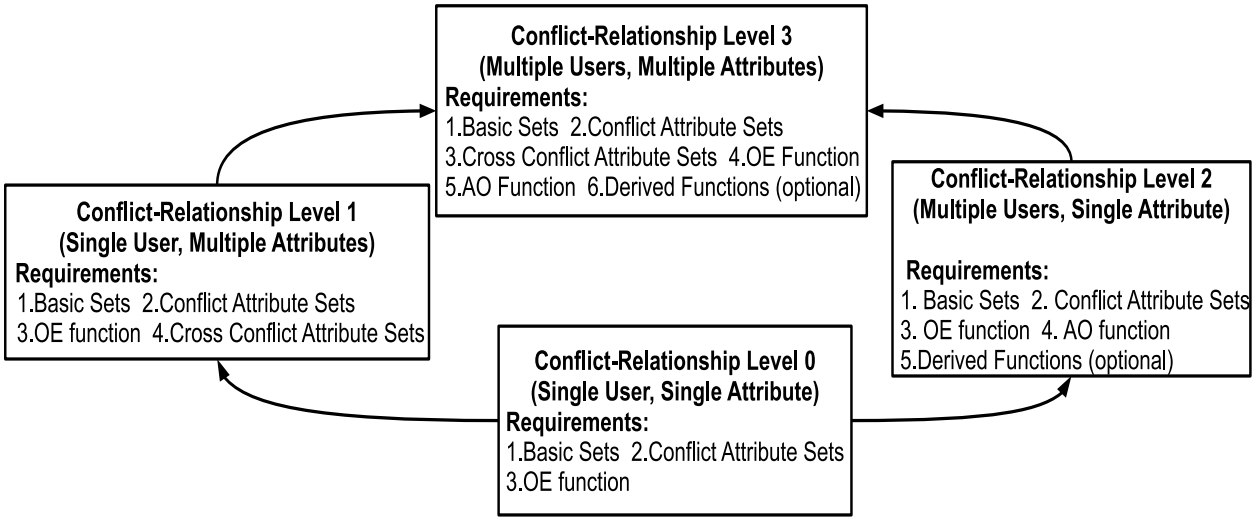


**Figure 3.3**: *Relationship Hierarchy with Required ABCL Functionality*

$\mathcal{O}(N\times(M+O)\times P)$ where N is the number of users, M the size of ***Attribute_Set***, O the size of ***Cross_Attribute_Set*** respectively and P is number of predicates.

**Level 2** (Multiple Users, Single Attribute): Constraints are specified based on conflict-relations among values of an attribute and applied to a set of users collectively. The function **AO** is required in a constraint expression besides **OE** for enforcing constraints across different users. The complexity here is $\mathcal{O}(N^2\times M\times P)$. Note that, constraints in this level enable dynamic separation of attribute values across subjects of a single user. For instance, a constraint might say that two subjects of a user cannot get 'president' role simultaneously.

**Level 3** (Multiple Users, Multiple Attributes): In this level, all type of constraints can be generated. The complexity is $\mathcal{O}(N^2\times(M+O)\times P)$ and a constraint can specify both single attribute and cross attribute conflicts and enforce within or across users.

## 3.4 ABCL Use Cases

We first show an ABCL instantiation for representing constraints in RBAC systems. Then, we present an extensive case study in which a large set of ABCL expressions is generated to capture various access control requirements of a banking organization.

**Table 3.4**: Attributes of User, Subject and Object in RBAC

| | Attribute Name | attType | Range |
|---|---|---|---|
| *UA* | *role* | set | {'$r_1$', '$r_2$', ..., '$r_n$'} |
| *SA* | *activerole* | set | {'$r_1$', '$r_2$', ..., '$r_n$'} |
| *OA* | *permittedrole* | set | {'$r_1$', '$r_2$', ..., '$r_n$'} |

**Table 3.5**: Constraints Specification for RBAC SSOD and DSOD

**1. Attribute_Sets Declaration:**

***Attribute_Set***$_{U,role}$ *ConflictRoles*

*ConflictRoles*={avset$_1$, avset$_2$, ...} where

avset$_i$ = (attval, limit) where attval$\in 2^{\textbf{Range}(role)}$ and

(limit=1 (for RCL-2000) or 1$\leq$limit$\leq$|attval| (for NIST-RBAC))

***Attribute_Set***$_{S,activerole}$ *ConflictActiveRoles*

*ConflictActiveRoles*={avset$_1$, avset$_2$, avset$_3$, ...} where

avset$_i$ = (attval, limit) where attval$\in 2^{\textbf{Range}(activerole)}$ and

(limit=1 (for RCL-2000) or 1$\leq$limit$\leq$|attval| (for NIST-RBAC))

**2. ABCL Expression for SSOD of RCL-2000 and NIST-RBAC**

**Requirement:** No user should be assigned to two roles which are in conflict with each other.

**Expression:** $|\textbf{OE}(\textit{ConflictRoles}).\text{attval} \cap \textit{role}(\textbf{OE}(U))| \leq \textbf{OE}(\textit{ConflictRoles}).\text{limit}$

**3. ABCL Expression for DSOD of RCL-2000 and NIST-RBAC**

**Requirement 1:** A Subject of a user cannot activate roles having conflict with each other.

**Expression:** $|\textbf{OE}(\textit{ConflictActiveRoles}).\text{attval} \cap \textit{activerole}(\textbf{OE}(S))| \leq \textbf{OE}(\textit{ConflictActiveRoles}).\text{limit}$

**Requirement 2:** Subjects of a user cannot activate roles having conflict with each other.

**Expression:** $\textbf{SubCreator}(\textbf{OE}(S))=\textbf{SubCreator}(\textbf{OE}(\textbf{AO}(S))) \Rightarrow |(\textit{activerole}(\textbf{OE}(S)) \cap \textbf{OE}(\textit{ConflictActiveRoles}).\text{attval}) \cup (\textit{activerole}(\textbf{OE}(\textbf{AO}(S))) \cap \textbf{OE}(\textit{ConflictActiveRoles}).\text{attval})| \leq \textbf{OE}(\textit{ConflictActiveRoles}).\text{limit}$

### 3.4.1 RBAC Constraints (RCL-2000 and NIST-RBAC SOD)

In RBAC, users create sessions in which they activate certain roles to perform particular tasks. The main constraints in RBAC concern static and dynamic separation of duty (termed SSOD and DSOD respectively). SSOD is applied on role assignment to users and DSOD is for role activation within or across sessions of a user. An ABAC model could be configured to enforce RBAC by defining only one attribute called *role* for users, subjects and objects as shown in table 3.4. Here, a

subject is synonymous to a session in RBAC. Hence, SSOD is applied during a user's *role* attribute assignment and DSOD for *activerole* assignment to subjects by their owners.

Table 3.5 shows the ABCL expressions for SSOD and DSOD constraints proposed in two well-known RBAC constraints models: role based constraint language (RCL-2000) [12] and constraints of NIST-RBAC [57]. RCL-2000 has a set called conflicted role (*CR*) in which each element of *CR* is a set of roles having conflict with each other. Here, SSOD and DSOD are maintained by allowing no more than one role assigned to users or activated in any user session respectively from each set element of *CR*. RCL-2000 also provides a constraints specification language for generating various constraints.

NIST-RBAC includes a cardinality metric with each set element that allows variable number of roles from each conflicted set instead of always allowing only one. In table 3.5, two instantiations of ***Attribute_Set***, *ConflictRoles* and *ConflictActiveRoles* are declared in order to represent conflicted values of *role* and *activerole* attributes. Each element of these sets is an ordered pair (attset, limit) where attset is the conflicted values and limit is the cardinality.

Items 2 and 3 of table 3.5 show ABCL constraint expressions for SSOD and DSOD respectively that capture both RCL-2000 and NIST-RBAC requirements. Similar to conflict role-set, RCL-2000 also has a set *CU* representing different set of conflicting users. ABCL can generalize the concept of conflicting users by introducing a user attribute *ucType* that represents different types of user conflict. Therefore, instead of identifying each conflicted user and creating a conflict set like *CU*, the values of *ucType* determine the conflict group a user belongs to and restrict user-role assignment accordingly.

### 3.4.2 Security policy specifications for Banking Organizations

We present ABCL constraints for several high-level security requirements in a banking organization. For simplicity, we only show constraints for user attribute management in this context. In a banking organization, let us consider a finite set of existing users (U) in which a user is a human being and could be of different types, e.g. client, junior employee.

**Table 3.6**: User Attributes (UA)

| Attribute | attType | Range |
|---|---|---|
| *id* | atomic | {'$id_1$','$id_2$', ..., '$id_x$'} |
| *uType* | atomic | {'client', 'junior', 'senior', 'leader'} |
| *orgType* | set | {'$org_1$', '$org_2$', ..., '$org_{20}$'} |
| *role* | set | {'customer', 'cashier', 'manager', 'president', 'vice-president'} |
| *benefit* | set | {'$bf_1$', '$bf_2$', '$bf_3$', ..., '$bf_{10}$'} |
| *felony* | set | {'$fl_1$', '$fl_2$', '$fl_3$', ..., $fl_8$'} |
| *loan* | set | {'car', 'house', 'education'} |
| *cCard* | set | {'$card_1$', '$card_2$', ..., '$card_{12}$'} |

Table 3.6 shows different user attributes, their types and ranges in this system. Each user is assigned an attribute *id* which is a unique identifier. Attribute *uType* represents the type of a user and *orgType* represents the organization a user belongs to. There is a *role* attribute representing various job descriptions of a user such as 'customer', 'cashier', etc. The banks might provide a number of benefits, e.g., bonus, cash back rate, etc, to customers represented by the *benefit* attribute. Attribute *felony* represents if the user has any felony record and *loan* and *cCard* represent granted loans and credit cards to a user respectively. Suppose that the banking authority wishes to specify the following security policy requirements for user attribute management. The ABCL formalism for these requirements are also given. We also show the conflict-relationship level of each of these constraints.

**Req# 1:** A user can get at most 5 *benefits*. (Level 0)

**Req# 2:** A user cannot hold the 'president' and 'vice-president' *roles* simultaneously. (Level 0)

**Req# 3:** A user cannot get both *benefits* '$bf_1$' and '$bf_2$'. (Level 0)

**Req# 4:** A user can get at most 5 *loans* and *cCards*. (Level 1)

**Req# 5:** If a user has *felony* records '$fl_1$' and '$fl_2$', she cannot get more than one *benefit* from {bf1, bf2, bf3}. (Level 1)

**Req# 6:** If a user is a 'client', she cannot get certain *roles*, e.g. 'cashier', 'manager'. (Level 1)

**Req# 7:** No more than 12 users can get a 'car' *loan*. (Level 2)

**Req# 8:** *ids* of two users cannot get the same value. (Level 2)

**Req# 9:** If a user has *felony* '$fl_1$' and belongs to '$org_1$', no users from '$org_1$' can get *benefit* '$bf_1$'. (Level 3)

**Formal ABCL Specifications**

**Table 3.7**: ABCL Sets Declaration and Initialization:

**1. *Attribute_Set* Declaration and Initialization:**

***Attribute_Set***$_{U,benefit}$ *UMEBenefit*

*UMEBenefit*={avset$_1$, avset$_2$}

avset$_1$=({'$bf_1$', '$bf_2$'}, 1), avset$_2$=({'$bf_2$', '$bf_3$', '$bf_4$', '$bf_5$'}, 2)

***Attribute_Set***$_{U,role}$ *UMERole*

*UMERole*={avset$_1$}

avset$_1$=({'president', 'vice-president'}, 1)

**2. *Cross_Attribute_Set* Declaration and Initialization:**

***Cross_Attribute_Set***$_{U, \{uType\}, \{role\}}$ *UMECTR*

*UMECTR*={attfun$_1$}

attfun$_1$(*uType*)=({'client'},1)

attfun$_1$(*role*)=({'cashier','manager','president','vice-precident'}, 0)

***Cross_Attribute_Set***$_{U, \{felony\}, \{benefit\}}$ *UMECFB*

*UMECFB*={attfun$_1$, attfun$_2$}

attfun$_1$(*felony*)=({'$fl_1$','$fl_2$'},2)

attfun$_1$(*benefit*)=({'$bf_1$','$bf_2$','$bf_3$'},1)

attfun$_2$(*felony*)=({'$fl_1$'},1), attfun$_2$(*benefit*)=({'$bf_2$'}, 0)

***Cross_Attribute_Set***$_{U, \{felony, orgType\}, \{benefit\}}$ *UMECFOB*

*UMECFOB*={attfun$_1$}

attfun$_1$(*felony*)=({'$fl_1$'},1), attfun$_1$(*orgType*)=({'$org_1$'}, 1),

attfun$_1$(*benefit*)=({'$bf_1$'}, 0)

Table 3.7 shows declaration and initialization of the ABCL sets for representing necessary relations among attributes for specifying above given security policies. *UMEBenefit* contains mutual exclusive values of the *benefit* attribute and *UMERole* represents mutual exclusive roles. Similarly, mutual exclusive conflicts of *uType* with *role*, *felony* with *benefit*, and *felony* and *orgType* with *benefit* attributes are represented by the ***Cross_Attribute_Set***s *UMECTR*, *UMECFB*, and *UMECFOB* respectively. The following are the required ABCL expressions.

**Req# 1:** $|benefit(\mathbf{OE}(U))| \leq 5$.

**Req# 2:** $|\mathbf{OE}(UMERole).\text{attset} \cap role(\mathbf{OE}(U))| \leq \mathbf{OE}(UMERole).\text{limit}$

**Req# 3:** $|\mathbf{OE}(UMEBenefit).\text{attset} \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMEBenefit).\text{limit}$

**Req# 4:** $|cCard(\mathbf{OE}(U)) + loan(\mathbf{OE}(U))| \leq 5$

**Req# 5:** $|\mathbf{OE}(UMECFB)(felony).\text{attset} \cap felony(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECFB)(felony).\text{limit} \Rightarrow$
$|\mathbf{OE}(UMECFB)(benefit).\text{attset} \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMECFB)(benefit).\text{limit}$

**Req# 6:** $|\ \mathbf{OE}(UMECTR)(uType).\text{attset} \cap uType(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECTR)(uType).\text{limit} \Rightarrow$
$|\ \mathbf{OE}(UMECTR)(role).\text{attset} \cap benefit(\mathbf{OE}(U))| \leq \mathbf{OE}(UMECTR)(role).\text{limit}$

**Req# 7:** $|\mathbf{assignedEntities}_{U,\ loan}(\text{`car'})| \leq 12$

**Req# 8:** $id(\mathbf{OE}(U)) \neq id(\mathbf{OE}(\mathbf{AO}(\mathbf{OE}(U))))$

**Req# 9:** $|\mathbf{OE}(UMECFOB)(felony).\text{attset} \cap felony(\mathbf{OE}(U))| \geq \mathbf{OE}(UMECFOB)(felony).\text{limit}$
$\wedge\ |\mathbf{OE}(UMECFOB)(orgType).\text{attset} \cap orgType(\text{OE(U)})| \geq$
$\mathbf{OE}(UMECFOB)(orgType).\text{limit} \Rightarrow |\mathbf{OE}(UMECFOB)(benefit).\text{attset} \cap$
$(benefit(\mathbf{OE}(U)) \cup benefit(\mathbf{OE}(\mathbf{AO}(U))))| \leq \mathbf{OE}(UMECFOB)(benefit).\text{limit}$

## 3.5 Performance Evaluation

In this section, we present experiments aimed at evaluating the performance of our ABCL enforcement algorithm during user attribute assignment (discussed in section 3.3). The experiments were conducted on a machine having the following configuration: 2.40GHz with 2GB RAM running a Windows 7 enterprize OS and JDK 1.7. As shown in section 3.3, ABCL constraints are represented as RFOPL expressions for enforcement during attributes assignment to a user and each universal quantifier of an expression generates a loop for traversing respective elements and checks if the constraint holds for those elements. Here, an element could be a member of the user set ($U$) or a declared relation-set and the required time for a constraint enforcement during a user attribute assignment depends on the size of these sets.

**Simulation scenario:** We define three user attributes: *att1*, *att2*, and *att3* where *att1* and *att2* are atomic and *att3* is set valued. We enforce the following two constraints during an attribute assign-

ment to a user (shown in RFOPL format).

**1)** $\forall u1 \in U, \forall u2 \in U\text{-}\{u1\}, \forall ele \in MUatt1$: $att1(u1) \in ele.\text{attval} \wedge att1(u2) \in ele.\text{attval} \Rightarrow att2(u1) \neq att2(u2)$.

**2)** $\forall\, u \in U, \forall\, ele \in MUatt3$: $\mid att3(u) \cap ele.\text{attval} \mid \leq ele.\text{limit}$.

Here, *MUatt1* and *MUatt3* contains mutual exclusive (ME) values of *att1* and *att3*. Expression 1 says if *att1* of two users contains ME values then they can get the same *att2* values and expression 2 says a user cannot get ME *att3* values. In **experiment 1**, we compare the required time of our enforcement algorithm when the number of users increase. We vary the number from 50 to 500 users with an increase of 50 at each step and check the required time for an attribute assignment to a user. We separately enforce these two constraints and check the timing. Note that, the size of both *MUatt1* and *MUatt3* are fixed to 5 elements, hence, execution time varies for the first two quantifiers in constraint 1 and the first quantifier of constraint 2. Figure 3.4(A) shows the results where constraint 1 takes more time as it applies to multiple users (falls in level 1 of the conflict-relationship hierarchy) while constraint 2 applies to every user separately (falls in level 0). Enforcement time for constraint 1 is 0.3s for 50 users with comparison to 1.27s for 500 users. And, for constraint 2 it is 0.109s to 0.3937s. Therefore, this process is scalable for a large set of users. In **experiment 2**, we verify the timing when the number of constraints increase while the total number of users are fixed to 500. Here, all constraints are similar to the constraint 1 which belongs to the hierarchical level 1, thus, applies across users. Figure 3.4(B) shows the required enforcement time when number of constraints increases from 5 to 30 which is only a 1.84s increase. In **experiment 3**, we analyze when the elements of a relation-set increases. Here, we enforce constraint 1 with number of users fixed to 500. Therefore, the required time varies only for the 3rd quantifier which depends on the size of *MUatt1*. Figure 3.4(C) shows the enforcement time where number of elements in *MUatt1* increases from 5 to 30. This causes an increase of 0.91s which is negligible, hence, it proves that the ABCL enforcement algorithm is scalable.

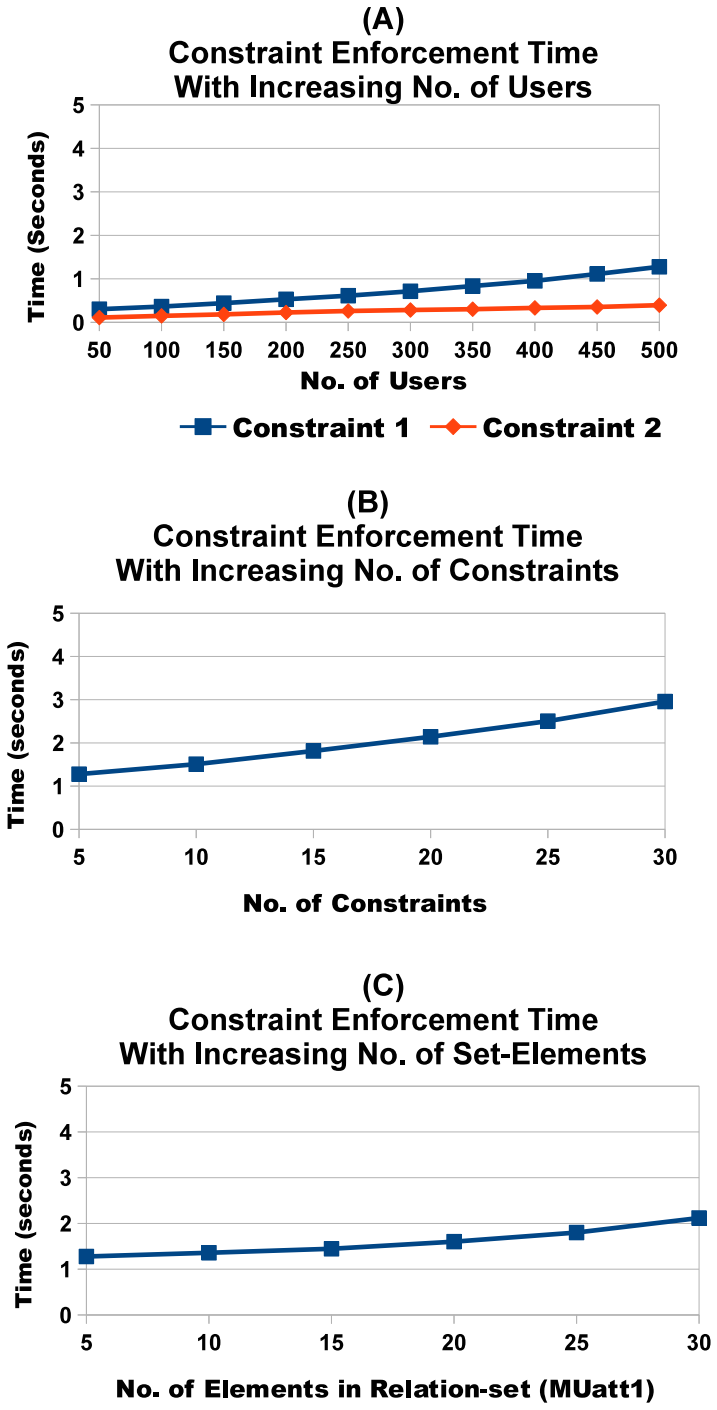


**Figure 3.4**: Evaluation Graphs of ABCL Constraints

# Chapter 4: FOUNDATION OF ATTRIBUTE-BASED CONSTRAINTS SPECIFICATION IN CLOUD IaaS

*The materials in this chapter are published in following venues [20,24]*:

1. Khalid Bijon, Ram Krishman, and Ravi Sandhu. Constraints specification in attribute based access control. ASE Science Journal, 2(3), 2013.

2. Khalid Bijon, Ram Krishnan and Ravi Sandhu, A Formal Model for Isolation Management in Cloud Infrastructure-as-a-Service. In Proceedings 8th International Conference on Network and System Security (NSS 2014), Xi'an, China, October 15-17, 2014

We show that attributes can represent various properties of virtual resources in cloud IaaS. Given that attributes represent properties of virtual resources, we provide ABCL specifications for various high-level constraints that can mitigate various security issues in cloud IaaS. We also show that an existing isolation management of the users and resources in cloud datacenter [18] can be configured using an attribute based system.

## 4.1 Attributes Specification for Virtual Resources

Attributes, associated with an virtual resource, can represent various properties of the virtual resource. For instance, attributes can represent a virtual machine's owner tenant, sensitivity-level, cpu intensity-level of workloads, etc. In this section, we provide a design of attributes for virtual resources that capture various properties in order to address issues like multi-tenancy, isolation, scale, etc.. Figure 4.1 shows an example of attributes of different virtual resources where the classes of virtual resources include virtual machine (VM), virtual machine image (IMG), virtual network (NET) and virtual storage (STR). Note that, as described in chapter 3, there is only one set called *OA* that specifies the objects in ABAC, whereas, cloud IaaS has different classes of virtual resources which can be represented in different such sets. In the figure, these virtual resources are represented as square and attributes are ellipses or hexagons for atomic and set-valued attributes

respectively. Each attribute has a *Range* from which the values are assigned to that attribute for a specific virtual resource. Many of these attributes may represent some trivial properties such as *id* represents the identification of a virtual machine and *host* represents the physical host where a virtual machine runs. Attributes can also represent several important properties of the virtual resources. For instance, *sensitivity* and *cpu_intensive* represents a virtual machine's sensitivity level of the running workloads and cpu requirements respectively. For simplicity, the value of *sensitivity* could be 'high', 'medium', and 'low'. Also, VM can be divided among different hpc groups which is represented by attribute *hpcgroup*. Also, virtual resources, i.e. VM, STR, and IMG, can belong to a particular department or group of a tenant, which is represented as *group* attribute of these entities. Also, *imgSrc* specifies the source of the image of a virtual machine such as public, private, etc. *Status* shows that whether a virtual machine is in running, stop or migrating mode. Similarly, in figure 4.1, attributes of IMG includes *ownerTnt* and *allowedTnt* representing the tenant that owns the image and the tenants that can use that image respectively. There are also attributes called *repository* and *version* that represents the repository where the image is located and the version number of the image. Similarly, figure 4.1 shows attributes of STR and NET.

We identify that attributes of these resources have certain conflicting relations with each other which is similar to what we describe in chapter 3. In the rest of the dissertation, we identify specific conflicting relations and utilize them to solve various security related issues in the cloud.

## 4.2 Constraints Specification using ABCL

In this section, we show that ABCL can specify different constraints to solve various security related issues in cloud IaaS.

### 4.2.1 Security policy specification for IaaS Public Cloud

In a *public* cloud IaaS, virtual machines (VMs) are provided by a service provider to its clients where the physical servers are shared by multiple clients. In the following sections we also refer to clients as tenants. In this system, VM of a tenant can be compromised by at least four different

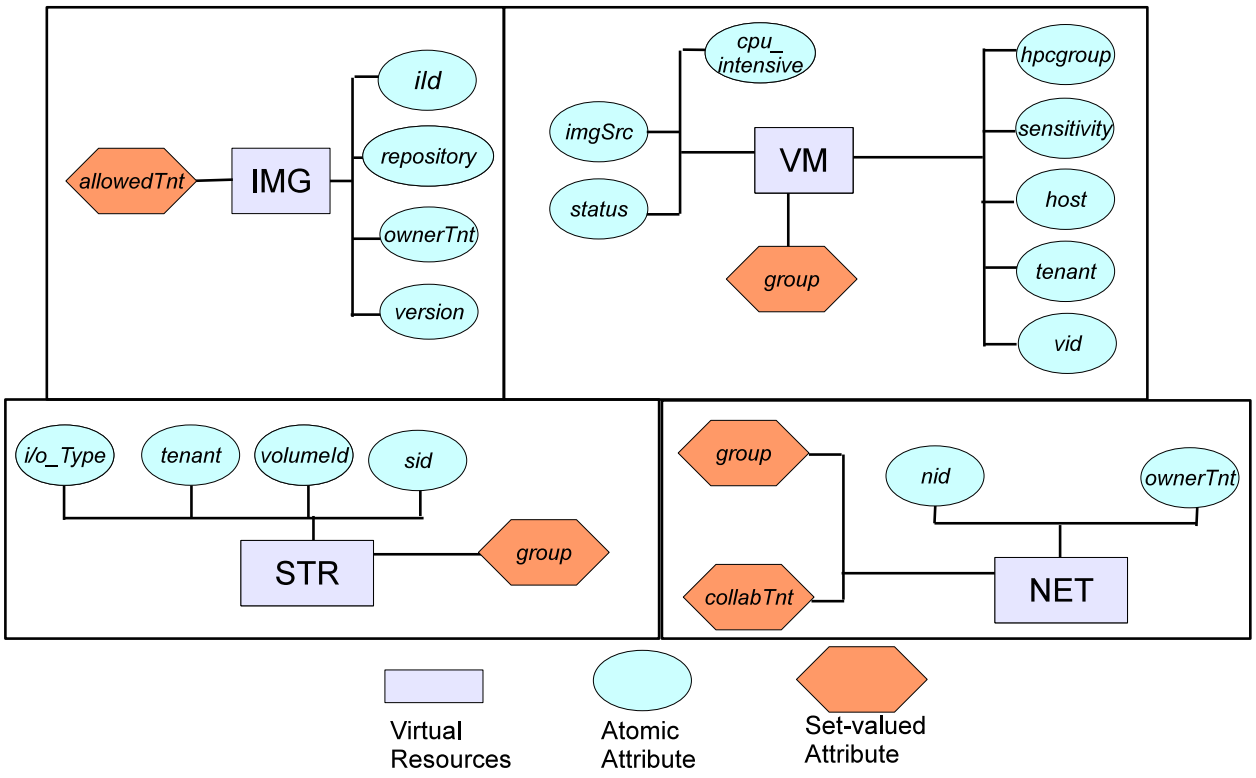


**Figure 4.1**: Attribute Design for Virtual Resources in Cloud IaaS

types of personnel: (1) malicious administrative users (admin) of the cloud provider, (2) malicious VMs of a competing client (tenant), (3) client's own admins, and (4) outsiders from the cloud system. Threats relevant to 3 and 4 are conventional security issues for which well-known protection mechanisms already exists, e.g. firewall, conventional access control policies. Since, threats 1 and 2 are more specific to IaaS public cloud environments, we aim to specify ABCL constraints for mitigating these threats.

In an IaaS public cloud, a provider's admins maliciously or unintentionally may abuse their privileges to compromise consumer's confidential data. Cloud service providers claim that they are aware of this issue and they have mitigation mechanisms [65] such as zero tolerance policy and isolating physical access to servers. However, zero tolerance policy is useful only after an attack has occurred. Also, several attacks including stealing of cleartext passwords, private keys, etc. by a malicious admin do not require any physical access [100]. Bleikertz et al [31] proposed a privilege management process for cloud admins by proposing three different administrative roles, i.e.

hardware-maintenance, remote-maintenance, and security-team, for requiring separation of duties. An admin with remote-maintenance role only has access and responsibility to maintain the servers that run client VMs . However, this mechanism cannot restrict certain administrative actions including restricting same admin from accessing VMs from competing tenants in multi-tenant public cloud. Competing tenants are organizations with conflict of interests, e.g. business competitors, conflicting departments of an organization. Thus, access to the VMs of the competing tenants by same admin might cause (un)intentional critical information flows from one VM to another. Again, a malicious VM of a competing tenant might also launch attack. Ristenpart et al [98] showed a side-channel attack is possible when VMs are co-located in the same server. Berger et al [18] mentioned other attacks, e.g. denial-of-service, could also be initiated by a malicious VM towards other VMs sharing same cloud resources, e.g. hosts, network. Hence, a cloud consumer should demand several security policies from the IaaS cloud providers, e.g. isolate physical location of their VMs from competing tenants VMs , restrict administrative privileges, etc. Below we enumerate several such issues, including those addressed by [18]. We categorize them as admin privilege management and VM resources management in IaaS cloud.

**A. Security issues related to the VM resources management**

**1)** A consumer tenant wants its high sensitive VMs not to be co-located on the same physical server where VMs of its competing tenants reside.

**2)** A tenant does not want its VMs to connect to a network (VLAN) to which VMs from competing tenants are connected.

**3)** A group of tenants collaborate together, thus, they want their collaboration-purposed VMs to reside in same server for enhancing several issues, e.g. performance, security, etc.

**4)** Collaborating tenants wants their VMs to connect to the same network so that they can securely share information.

**5)** Some VMs of a tenant might require to exchange highly critical data for some reasons. Thus, they need to reside on same physical sever to utilize internal process communication.

**6)** A tenant wants highly sensitive VMs to reside in different servers so that any kind of service

**Table 4.1**: Attributes

| | attType | Range | Description |
|---|---|---|---|
| **UA** | | | |
| *tenant* | set | ‘$t_1$’, ‘$t_2$’, ..., ‘$t_8$’ | Tenants an admin can access |
| *host* | set | ‘$node_1$’, ‘$node_2$’, ...., ‘$node_{20}$’ | Servers an admin has access |
| *adminGrp* | set | ‘hardware_maintenance’, ‘security’, ‘remote_maintenance’ | Different groups of an administrative users |
| *role* | set | ‘pCreator’, ‘vmMonitor’, ‘vmAdmin’ | Admin Roles |
| **SA** | | | |
| *acctnt* | set | ‘$t_1$’, ‘$t_2$’, ..., ‘$t_8$’ | Tenants a subject can access |
| *accserver* | set | ‘$node_1$’, ‘$node_2$’, ...., ‘$node_{20}$’ | Servers a subject has access |
| *activerole* | set | ‘pCreator’, ‘vmMonitor’, ‘vmAdmin’ | Admin roles |
| **OA (VM)** | | | |
| *otnt* | atomic | ‘$t_1$’, ‘$t_2$’, ..., ‘$t_8$’ | Tenant that owns the VM |
| *host* | atomic | ‘$node_1$’, ‘$node_2$’, ...., ‘$node_{20}$’ | Server where the VM resides |
| *purporsetype* | atomic | ‘$p_1$’, ‘$p_2$’, ‘$p_3$’, ‘$p_4$’, ‘$p_5$’ | Job of a VM |
| *sensitivity* | atomic | ‘high’, ‘low’ | Sensitivity level of the VM |
| *status* | atomic | ‘Active’, ‘Stop’, ‘Maintenance’, ‘Transferring’ | Current status of the VM |
| *network* | atomic | ‘$vlan_1$’, ‘$vlan_2$’, ..., ‘$vlan_{20}$’ | Network connection a VM can get |
| *permittedRole* | set | ‘pCreator’, ‘vmMonitor’, ‘vmAdmin’ | Roles that can access the VM |

interruption or security issues on that server may only cause partial disruptions.

**7)** A tenant allows its less sensitive VMs to reside on the same server where VMs of a competing tenant reside. However, during maintenance, these VMs need to be migrated to a server that does not contain any VMs from the competing tenants.

**B. Security issues on admin-user privileges management**

**1)** A tenant does not allow the same admin to access their sensitive VMs if she has access to the competing tenant VMs.

**2)** In general, an admin cannot maintain more than n tenants.

**3)** A tenant cannot be managed by more than one sessions (subjects) of an admin simultaneously.

**4)** An admin cannot access more than n VMs of a tenant simultaneously (e.g. in the same session) for protecting possible aggregation of the critical information.

### 4.2.2 ABCL Specification for Public Cloud IaaS

We presents ABCL constraints specification for the above given security requirements. There are sets of administrative users ($U$) and subjects ($S$) where each subject belongs to a particular administrative user. In this system, objects are virtual machines (VMs) which are represented as a set ($O$). We will see in the following chapters that there will be other classes of virtual resources such as virtual networks, virtual storages. Table 4.1 shows user, subject, and object attributes, their types and ranges and descriptions of their purpose. The declaration and initialization of the required ABCL sets are shown in table 4.2. *UMETnt*, *UMEGrp*, and *UMERole* represents the mutual exclusive conflicts of the user attributes *tenant*, *adminGrp*, and *role* respectively. Mutual exclusive values of the subject attribute *acctnt* are represented in *SMETnt*. *OConsTnt* and *OMETnt* contain values of *otnt* having mutual exclusive and consistency conflicts respectively. ABCL constraints for the policies are as follows:

**A. VM resource management Constraints**

**Req# 1:** High sensitive VMs of a tenant cannot reside on same server that contains VMs from competing tenants.

**Expr:** $sensitivity(\mathsf{OE}(O))$=high $\wedge$ $otnt(\mathsf{OE}(O)) \in \mathsf{OE}(OMETnt)$.attval$\wedge$ $otnt(\mathsf{OE}(\mathsf{AO}(O))) \in \mathsf{OE}(OMETnt)$.attval $\Rightarrow$ $host(\mathsf{OE}(O)) \neq host(\mathsf{OE}(\mathsf{AO}(O)))$

**Req# 2:** VMs of cooperative tenants reside on same server.

**Expr:** $otnt(\mathsf{OE}(O)) \in \mathsf{OE}(OConsTnt)$.attval $\wedge$ $otnt(\mathsf{OE}(\mathsf{AO}(O))) \in \mathsf{OE}(OConsTnt)$.attval $\Rightarrow$ $host(\mathsf{OE}(O))=host(\mathsf{OE}(\mathsf{AO}(O)))$

**Req# 3:** Similar purpose VMs reside in same server.

**Expr:** $otnt(\mathsf{OE}(O))=otnt(\mathsf{OE}(\mathsf{AO}(O)))$ $\wedge$ $purporsetype(\mathsf{OE}(O)) = purporsetype(\mathsf{OE}(\mathsf{AO}(O)))$ $\Rightarrow$ $host(\mathsf{OE}(O))=host(\mathsf{OE}(\mathsf{AO}(O)))$

**Req# 4:** High sensitive VMs of tenants are located to different servers.

**Expr:**$sensitivity(\mathsf{OE}(O))$='high' $\wedge$ $sensitivity(\mathsf{OE}(\mathsf{AO}(O)))$ ='high' $\wedge$ $otnt(\mathsf{OE}(O))=otnt(\mathsf{OE}(\mathsf{AO}(O)))\Rightarrow$ $host(\mathsf{OE}(O))\neq host(\mathsf{OE}(\mathsf{AO}(O)))$

**Req# 5:** Less sensitive VM of a tenant cannot reside in same server of a competing tenant during maintenance.

**Expr:** $status(\mathsf{OE}(O))$='maintenance' $\wedge$ $sensitivity(\mathsf{OE}(O))$='low' $\wedge$ $otnt(\mathsf{OE}(O)) \in \mathsf{OE}(OMETnt).\text{attval} \wedge otnt(\mathsf{OE}(\mathsf{AO}(O)) \in \mathsf{OE}(OMETnt).\text{attval} \Rightarrow host(\mathsf{OE}(O)) \neq host(\mathsf{OE}(\mathsf{AO}(O)))$

**Req# 6:** VMs of tenant cannot connect to same network that is connected to VMs of competing tenants.

**Expr:** $otnt(\mathsf{OE}(O)) \in \mathsf{OE}(OMETnt).\text{attval} \wedge otnt(\mathsf{OE}(\mathsf{AO}(O))) \in \mathsf{OE}(OMETnt).\text{attval} \Rightarrow network(\mathsf{OE}(O)) \neq \text{network}(\mathsf{OE}(\mathsf{AO}(O)))$

Note that, above given security requirements 1-5 are for managing co-location of the VMs to the physical servers and security requirements 6 is for managing mapping between two virtual resources, i.e., VM and NET. We consider these two problems as scheduling problem and configuration management problem respectively. Enforcing such constraints in cloud IaaS may cause various cloud IaaS specific issues including enforcement complexities and resource optimization. In this chapter, we do not analyze these issues, rather, we only show that ABCL is capable to specify various constraints. In chapter 5, we develop a constraint-driven virtual resources management by utilizing the appropriate subset of ABCL where we analyze these issues. Also, in chapter 6, we develop a constraint-aware virtual machine scheduler in which we only use elements from ABCL which are required to solve this problem.

**B. Constraints for admin-user privileges management**

**Req# 1:** An admin can access VMs of a tenant, if he is not an admin of the competing tenants.

**Expr:** $|tenant(\mathsf{OE}(U)) \cap \mathsf{OE}(UMETnt).\text{attval}| \leq \mathsf{OE}(UMETnt).\text{limit}$

**Req# 2:** An administrative user cannot maintain more than 3 tenants.

**Expr:** $|tenant(\mathsf{OE}(U))| \leq 3$

**Req# 3:** A subject cannot access VMs from two ME tenants.

**Expr:** $|acctnt(\mathsf{OE}(S)) \cap \mathsf{OE}(SMETnt).\text{attval}| \leq \mathsf{OE}(SMETnt).\text{limit}$

**Table 4.2**: ABCL Sets Declaration and Initialization:

**1. *Attribute_Set* Declaration and Initialization:**

***Attribute_Set***$_{U,tenant}$ *UMETnt*

*UMETnt*={avset$_1$, avset$_2$, avset$_3$}

avset$_1$=({‘t$_1$’,‘t$_3$’},1), avset$_2$=({‘t$_2$’,‘t$_4$’,‘t$_5$’},2), avset$_3$=({‘t$_7$’,‘t$_8$’},1)

***Attribute_Set***$_{U,adminGrp}$ *UMEGrp*

*UMEGrp*={avset$_1$}

avset$_1$=({‘hardware_maintenance’,‘remote_maintenance’},1)

***Attribute_Set***$_{O,otnt}$ *OMETnt*

*OMETnt*= {avset$_1$, avset$_2$, avset$_3$}

avset$_1$=({‘t$_1$’,‘t$_3$’},1), avset$_2$=({‘t$_7$’,‘t$_8$’},1), avset$_3$=({‘t$_2$’,‘t$_4$’,‘t$_5$’},2)

***Attribute_Set***$_{U,role}$ *UMERole*

*UMERole*={avset$_1$}

avset$_1$=({‘vmMonitor’, ‘vmAdmin’, ‘billAdmin’},1)

***Attribute_Set***$_{S,acctnt}$ *SMETnt*

*SMETnt*={avset$_1$, avset$_2$}

avset$_1$=({‘t$_1$’,‘t$_3$’},1), avset$_2$=({‘t$_2$’,‘t$_4$’,‘t$_5$’},1)

***Attribute_Set***$_{O,otnt}$ *OConsTnt*

*OConsTnt*={avset$_1$, avset$_2$},avset$_1$=({‘t$_1$’,‘t$_4$’},2), avset$_2$=({‘t$_7$’,‘t$_9$’},2)

**2. *Cross_Attribute_Set* Declaration and Initialization:**

***Cross_Attribute_Set***$_{U, \{adminGrp\}, \{role\}}$ *UMECGR*

*UMECGR*= {attfun$_1$, attfun$_2$, attfun$_3$}

attfun$_1$(*adminGrp*)=({‘hardware_maintenance’},1)

attfun$_1$(*role*)=({‘billAdmin’,‘pCreator’,‘vmAdmin’},0)

attfun$_2$(*adminGrp*)=({‘security’}, 1),

attfun$_2$(*role*)=({‘billAdmin’, ‘vmMonitor’,‘vmAdmin’},0))

attfun$_3$(*adminGrp*)=({‘remote_maintenance’},1)

attfun$_3$(*role*)=( {‘pCreator’,‘vmMonitor’},0))

**Req# 4:** A tenant cannot be accessed by more than one subject of an admin.

**Expr:** $\textsf{SubCreator}(\textsf{OE}(S))=\textsf{SubCreator}(\textsf{OE}(\textsf{AO}(S))) \Rightarrow acctnt(\textsf{OE}(S)) \cap acctnt(\textsf{OE}(\textsf{AO}(S)))=\phi$

**Req# 5:** An admin cannot join both hardware and remote maintenance.

**Expr:** $|\textsf{OE}(UMEGrp).\text{attval} \cap adminGrp(\textsf{OE}(U))| \leq \textsf{OE}(UMEGrp).\text{limit}$

**Req# 6:** An admin of hardware-maintenance group cannot get *roles* 'billAdmin' and 'pCreator'.

**Expr:** $|adminGrp(\textsf{OE}(U)) \cap \textsf{OE}(UMECGR).\text{attfun}(adminGrp).\text{attval}| \geq \textsf{OE}(UMECGR).\text{attfun}(adminGrp).\text{limit} \Rightarrow |\textsf{OE}(UMECGR).\text{attfun}(role).\text{attval} \cap role(\textsf{OE}(U))| \leq \textsf{OE}(UMECGR).\text{attfun}(role).\text{limit}$

## 4.3 Attribute Based Isolation Management

Recently, trusted virtual datacenter (TVDc) [18] proposed an isolation management process in cloud IaaS. In TVDc virtual machines and their associated resources, such as virtual bridge and virtual local access network (VLAN), are grouped into trusted virtual domains (TVDs). Each TVD, represented as a security clearance (also referred to as color), enforces an isolation policy towards its group members. More specifically, resources are only allowed to interact with each other if they are assigned to same color. For instance, VMs with same color can communicate and a VM can run on a hypervisor only if this hypervisor has the same color as that VM. The main goal of this process is to isolate customer workloads from each other. As described in [18], the purpose of this isolation is to reduce the threat of co-locating workloads from different customers by preventing any kind of data flow among these workloads where the data might includes sensitive information of the customers or any virus or malicious code. Again, this simple management process also reduces the incidence of misconfiguration of the virtual-resource management tasks. TVDc [18] also develops a hierarchical administration model based on trusted virtual domains.

We develop a formal model for TVDc which we call Formalized-TVDc (also referred as F-TVDc) where we show that the attribute based system can be leveraged to represent different properties of the virtual resources, such as color. Then, resources with similar attributes will be

arranged together to built a particular computing environment. For instance, a VM can run on a hypervisor, physical host or Host, if the Host has same color of the VM. This formal model consists authorization models for three types of administrative-user operations.

### 4.3.1 Background: Trusted Virtual Datacenter (TVDc)

Trusted virtual datacenter (TVDc) [18] manages isolation by defining a trusted virtual domain (TVD) for a set of VMs and their associated resources that constitute a common administrative unit. The boundary of a TVD is maintained by assigning the TVD identifier to the respective VMs and resources. A TVD identifier represents a security clearance (also referred to as a color to emphasize there is no ordering or structure). For instance, a color can represent the virtual resources of a particular customer or virtual resources running specific workloads of a customer. Hence, basically, a color represents a particular context for the assigned VMs and resources. Figure 4.2, from [18], shows the TVDc view of the virtual resources running two physical data centers and their resources, such as servers, storage, and network components. The TVDc view separates the association of physical resources for each color. For instance, in figure 4.2, the red color includes VMs 3, 7, 9, and 11, and associated storages.

In order to manage this isolation process, three different administrative roles are proposed in TVDc: IT datacenter administrator, TVDc administrator , and tenant administrator. Administrative users (also generally referred as admin-users) having IT datacenter administrator role are the super-users in this system. Their main task is to keep track of the physical and the virtual resources and group these resources into TVDcs. They also define the security labels or colors. IT datacenter admin-users further can assign a TVDc group to both TVDc and tenant admin-users, and assign a set of colors to TVDc admin-users in order to delegate isolation management of resources in that specific TVDc group. The tasks of a TVDc admin-user include assigning colors to the resources of a TVDc group, from her assigned set of colors. She also assigns a color to an admin-user if the admin-user is in the same TVDc group and assigned to the tenant administrator role. The job function of a tenant admin-user is to perform basic cloud administrative operations on the cloud

resources, such as boot a VM and connect a VM to a virtual network, if both the resources and the tenant admin-user are in same TVDc group and assigned with same color.

This color-driven isolation management process supports four different types of isolation which are described as follows.

1. **Data Sharing:** In this model, VMs can share data with each other only if they have common colors. In order to constrain this, a VM is allowed to connect to a VLAN only if both the VM and the VLAN have common colors and, therefore, it is restricted to communicate with VMs having same color.

2. **Hyperviosr (Host) Authorization:** A Host is assigned to a set of colors and is only allowed to run a VM having a color in that set. Therefore, a Host's capability to run VMs is isolated to its assigned colors.

3. **Colocation Management:** Two colors can be conflicted with each other if context of operation is mutually exclusive. Colors can be declared to be conflicting and two VMs with conflicted colors are prohibited from running in same Host. For instance, VMs A and B with color red and blue respectively which have been declared to be conflicting cannot run on same Host.

4. **Management Constraints:** For management isolation, tenant administrative roles are created where each user having this role is restricted to perform administrative operation within a single color.

### 4.3.2 Formal Isolation Management Model (F-TVDc)

In this section, we formalize the isolation management process in cloud IaaS which is informally described in TVDc [18]. We call the resulting model as Formalized-TVDc (F-TVDc) for ease of reference and continuity. In F-TVDc, different properties of the cloud entities are represented as assigned attributes to them. For instance, a virtual machine(VM) attribute *color* represents

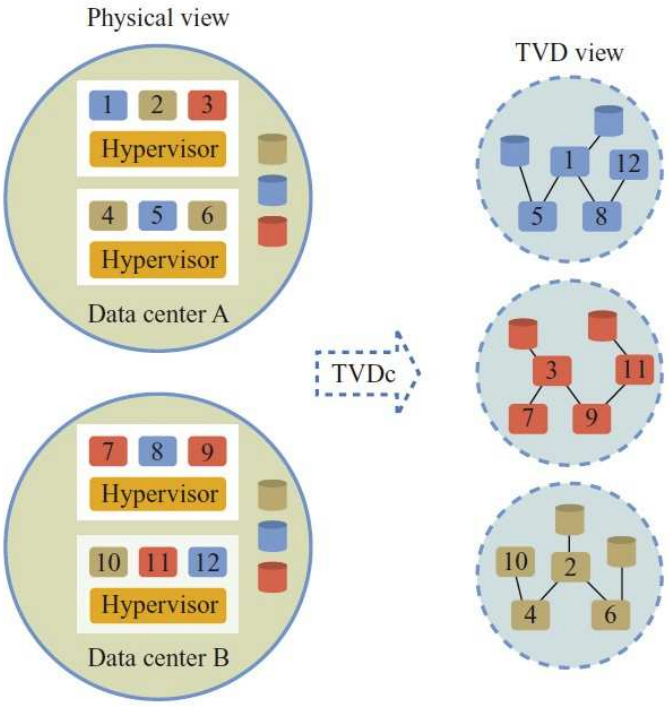


**Figure 4.2**: *TVDc View of the IaaS Cloud Resources [18]*

assigned colors to that VM and an administrative user (admin-user) attribute *adminRole* represents assigned role to that user. For this purpose we utilize the attribute based system [78], specifically its attribute representation for the entities in a system. In this attribute based representation of F-TVDc, the admin-users can manage the resources in data-center by assigning proper attributes to them. For instance, a TVDc admin-user can assign a set of colors to a Host and, consequently, the Host is authorized to run a VM if the assigned color of the VM is an element of the set of colors assigned to the Host. F-TVDc also formally represents an authorization model for these admin-user privileges.

**Basic Components**

The sets that contain basic entities of F-TVDc are shown in table 4.3. In F-TVDc, CLR contains the existing colors/clearances in the system. We will see later that the colors from CLR will be assigned to the cloud-resources' and admin-users' respective attributes, such as the *admincolor* attribute of an admin-user. The data-center is divided into multiple virtual data-centers. VDc con-

**Table 4.3**: Basic Sets for F-TVDc

| |
|---|
| CLR = Finite set of existing colors |
| VDc = Finite set of existing virtual data centers |
| AROLE = {itAdmin, tvdcAdmin, tntAdmin} |
| AU = Finite set of existing admin-users |
| VM = Finite set of existing virtual machines |
| VMM = Finite set of existing hypervisors |
| BR = Finite set of existing virtual bridges |
| VLAN = Finite set of existing virtual LANs |

tains the names of these virtual data-centers. There are three administrative roles: IT administrator (itAdmin), TVDc administrator (tvdcAdmin), and tenant administrator (tntAdmin) which are contained in set AROLE. The set AU contains all admin-users in the system. VMM and VM contains the current existing hypervisors (Hosts) and the virtual machines(VMs) in the system. Similarly, existing virtual LANs and virtual bridges are contained in set VLAN and BR respectively.

**Attributes**

Attributes characterize properties of an entity and are modeled as functions. F-TVDc recognize two types of attribute functions for each entity depending on the nature of the function's values: atomic-valued and set-valued. For instance, an admin-user attribute function *adminRole* can only take a single value that indicates the assigned role to that user. On the other hand, the attribute function *admincolor*, representing the assigned colors to an admin-user, can take multiple values. For convenience we understand attribute to mean attribute function for ease of reference. Attributes of an entity, let's say VM attributes, can be formally defined as follows:

- $\text{ATTR}_{\text{VM}}$ is the finite set of VM attributes, where
  attType: ATTR $\rightarrow$ {set, atomic}.

- For each $att \in \text{ATTR}_{\text{VM}}$, $\text{SCOPE}_{att}$ is a finite set of atomic values which determines the range of *att* as follows:

**Table 4.4**: Attributes Specification

| Entity | Attributes | attType | SCOPE |
|---|---|---|---|
| Admin-User | *adminRole* | atomic | AROLE |
| | *adminvdcenter* | set | VDc |
| | *admincolor* | set | CLR |
| Virtual Machine | *vmvdcenter* | atomic | VDc |
| | *vmcolor* | atomic | CLR |
| | *host* | atomic | VMM |
| | *status* | atomic | {Running, Stop} |
| | *bridge* | set | BR |
| Hypervisor (Host) | *vmmvdcenter* | atomic | VDc |
| | *vmmcolor* | set | CLR |
| | *vm* | set | VM |
| Virtual Bridge | *brvdcenter* | atomic | VDc |
| | *brcolor* | atomic | CLR |
| | *vm* | set | VM |
| | *vlan* | atomic | BR |
| Virtual LAN | *vlanvdcenter* | atomic | VDc |
| | *vlancolor* | set | CLR |
| | *bridge* | set | BR |

$$\text{Range}(att) = \begin{cases} \text{SCOPE}_{att} \text{ if attType}(att) = \text{atomic} \\ \mathcal{P}(\text{SCOPE}_{att}) \text{ if attType}(att) = \text{set} \end{cases}$$

where $\mathcal{P}$ denotes the power set of a set.

- An attribute is a function that maps each VM$\in$VM to a value in range, i.e.,

$$\forall att \in \text{ ATTR}_{\text{VM}}.\ att : \mathsf{VM} \rightarrow \text{Range}(att)$$

Similary, attributes of other entities can be defined. Table 4.4 shows the necessary attributes for the entities in F-TVDc which are described as follows.

- Admin-User (aUser) attributes: *adminRole* attribute of an admin-user (aUser) specifies the assigned administrative role to aUser. Note that, an aUser can get only one administrative role, hence, *adminRole* is an atomic attribute. Attribute *adminvdcenter* represents the assigned virtual data-center of an aUser. If the aUser is an IT administrator then its *adminvdcenter* contains all the members in VDc. Otherwise *adminvdcenter* of an aUser contains only one element from VDc. Similarly, *admincolor* specifies the assigned colors to an aUser. If an aUser is an IT administrator then her *admincolor* contains all the elements of CLR. On the other hand, an aUser having tvdcAdmin role can get subset of colors from CLR and a tntAdmin gets only one color. Section 4.3.3 represents the operations to assign values of these aUser attributes.

- Virtual machine (VM) attributes: The VM attribute *host* represents the hypervisor (Host) where a VM is running. Attribute *bridge* represents the connected BRs of a VM. *vmvdcenter* represents the virtual data-center a VM belongs to and *vmcolor* specifies the assigned color to that VM.

- Hypervisor (Host) attributes: The *vm* attribute represents the running VMs in a Host. The *vmmvdcenter* attribute represents its virtual data-centers and *vmcolor* the assigned colors to it.

- Virtual Bridge (BR) attributes: The *vm* attribute of BR specifies the connected VMs to a BR. Similarly, *vlan* specifies the VLAN to which a bridge is connected. Similar to the other entities, *brcolor* and *brvdcenter* represent the virtual data-center and color assigned to a BR. Note these are atomic in this instance.

- Virtual LAN (VLAN) attributes: The *bridge* attribute of a VLAN specifies the connected virtual bridges to it. Also, *vlancolor* and *vlanvdcenter* represents the virtual data-center and colors assigned to a VLAN.

### 4.3.3 Administrative Models

In this section, we discuss administrative operations for the three types of admin-users. Table 4.5 formally specifies the set of administrative operations for the IT admin-user. The first column specifies the operation name and parameters. The second column specifies the conditions that need to be satisfied to authorize the operation. Attributes and sets that will be updated after an authorized operation are listed in the third column, with the $'$ symbol indicating the value after the update. Administrative operations of Table 4.5 are discussed below.

- **CreateVDC**: First column of table 4.5 shows that this function takes two parameters: users u and a virtual data-center vdc. Then, in second column, these parameters need to satisfy the given formula which checks if u belongs to AU, *adminRole* of u is itAdmin and vdc is not present in VDc. If the precondition is satisfied, in column 3, vdc is created by adding it to set VDc.

- **CreateCl** and **RemoveCl**: Using these two operations, an itAdmin can create a new color cl and remove an existing color cl.

- **Add_Cl$_{\text{TVDcAdmin}}$**: This function takes three parameters: users u1 and u2, and a color cl. These parameters need to satisfy the given formula in column 2 which checks if u1 has role itAdmin, u2 has role tvdcAdmin and cl is a valid member in CLR. If so, color cl is assigned to tvdcAdmin u2 by adding cl to *tvdcAdmincolor* attribute of u2, as shown in column 3.

- **Rem_Cl$_{\text{TVDcAdmin}}$**: Using this operation, an itAdmin removes a color cl from an admin-user having role tvdcAdmin.

- **Assign_VDC$_{\text{Admin}}$**: Using this operation an itAdmin user assigns a virtual data-center vdc to attribute *adminvdcenter* of a tvdcAdmin or tntAdmin user.

**Table 4.5**: IT admin-user Operations

| Operation | Precondition | Updates |
|---|---|---|
| **CreateVDC**(u, vdc) /*Creates a virtual data-center vdc*/ | u∈AU∧vdc∉VDc∧ *adminRole*(u)=itAdmin | VDc′=VDc∪{vdc} |
| **CreateCl**(u,cl) /*Creates a color cl*/ | u∈AU∧cl∉CLR∧ *adminRole*(u)=itAdmin | CLR′=CLR∪{cl} |
| **RemoveCl**(u,cl) /*Removes a color cl*/ | u∈AU∧cl∈CLR∧ *adminRole*(u)=itAdmin | CLR′=CLR-{cl} |
| **Add_Cl$_{TVDcAdmin}$**(u1,u2,cl) /*Adds cl to tvdcAdmin u2*/ | u1∈AU∧*adminRole*(u1)= itAdmin∧u2∈AU∧cl∈CLR∧ *adminRole*(u2)=tvdcAdmin∧ cl∉*admincolor*(u2) | *admincolor′*(u2)← *admincolor*(u2)∪{cl} |
| **Rem_Cl$_{TVDcAdmin}$**(u1,u2,cl) /*Removes cl from tvdcAdmin u2*/ | u1∈AU∧*adminRole*(u1)= itAdmin∧u2∈AU∧ *adminRole*(u2)=tvdcAdmin∧ cl∈*admincolor*(u2) | *admincolor′*(u2)← *admincolor*(u2)-{cl} |
| **Assign_VDC$_{Admin}$**(u1,u2,vdc) /*Assigns virtual datacenter vdc to tvdcAdmin or tntAdmin u2*/ | u1∈AU∧*adminRole*(u1)= itAdmin∧u2∈AU∧ (*adminRole*(u2)=tvdcAdmin∨ *adminRole*(u2)=tntAdmin)∧ vdc∈VDc | *adminvdcenter′*(u2)← {vdc} |
| **Assign_VDC$_{VM}$**(u,vm,vdc) /*Assigns virtual datacenter vdc to a VM vm*/ | u ∈AU∧vm∈VM∧vdc∈ VDc∧*adminRole*(u)=itAdmin | *vmvdcenter′*(vm)← vdc |
| **Assign_VDC$_{VMM}$**(u,vmm,vdc) /*Assigns virtual datacenter vdc to a Host vmm*/ | u ∈AU∧vmm∈VMM∧vdc∈ VDc∧*adminRole*(u)=itAdmin | *vmmvdcenter′*(vmm)← vdc |
| **Assign_VDC$_{BR}$**(u,br,vdc) /*Assigns virtual datacenter vdc to a BR br*/ | u ∈AU∧br∈BR∧vdc∈ VDc∧*adminRole*(u)=itAdmin | *brvdcenter′*(br)← vdc |
| **Assign_VDC$_{VLAN}$**(u,vlan,vdc) /*Assigns virtual datacenter vdc to a VLAN vlan*/ | u ∈AU∧vlan∈VLAN∧vdc∈ VDc∧*adminRole*(u)=itAdmin | *vlanvdcenter′*(vlan)← vdc |

- **Assign_VDC$_{VM}$**: A virtual data-center vdc is assigned to the *vmvdcenter* attribute of a VM called vm. This value specifies that vm belongs to virtual data-center vdc.

- **Assign_VDC$_{VMM}$**: Similarly, a virtual data-center vdc is assigned to the *vmmvdcenter* attribute of a Host called vmm.

- **Assign_VDC$_{BR}$**: This operation assigns a virtual data-center named vdc to *brvdcenter* attribute of a BR called `br`.

- **Assign_VDC$_{VLAN}$**: A virtual data-center, vdc, is assigned to the *vlanvdcenter* attribute of a VLAN called vlan.

Similarly, table 4.6 shows the operations for TVDc admin-users. The TVDc admin-users are responsible to assign colors to the tntAdmins and the resources in data-centers where the TVDc admin-users are authorized to exercise their priviledges. The description of these operations are as follows:

- **Assign_Cl$_{TAdmin}$**: A tvdcAdmin u1 assigns a color cl to a tntAdmin u2. Authorization of this operation needs to satisfy the precondition that u1 and u2 are in the same virtual data-center. Also, the *admincolor* attribute of u2 must contain cl.

- **Rem_Cl$_{TAdmin}$**: Using this operation a tvdcAdmin removes a color from tntAdmin.

- **Add_Cl$_{VMM}$**: This operation adds a color cl to a Host named vmm if vmm and tvdcAdmin are in same virtual data-center. Note that, a Host can contain multiple colors.

- **Assign_Cl$_{VM}$**, **Assign_Cl$_{BR}$**, and **Add_Cl$_{VLAN}$**: Using first two operations operations a tvdcAdmin u assigns a color cl to a VM vm and a bridge br respectively. Also, using **Add_Cl$_{VLAN}$** the tvdcAdmin adds a color to the *vlancolor* attribute of a VLAN vlan. Note that, *vlancolor* attribute can contain multiple colors since vlan can be connected to multiple virtual bridges.

**Table 4.6**: TVDc-ADMIN Operations

| Operation | Precondition | Updates |
|---|---|---|
| **Assign_Cl$_{\mathbf{TAdmin}}$**(u1, u2,cl) /*Assigns a color cl to a tntAdmin u2*/ | u1∈AU∧u2∈AU∧*adminRole*(u1)= tvdcAdmin∧*adminRole*(u2)= tntAdmin∧ *adminvdcenter*(u1)= *adminvdcenter*(u2)∧cl∈ *admincolor*(u1)∧ cl∉*admincolor*(u2) | *admincolor′*(u2)← {cl} |
| **Rem_Cl$_{\mathbf{TAdmin}}$**(u1, u2,cl) /*Removes the color cl from a tntAdmin u2*/ | u1∈AU∧u2∈AU∧*adminRole*(u1)= tvdcAdmin∧*adminRole*(u2)= tntAdmin∧ *adminvdcenter*(u1)= *adminvdcenter*(u2)∧cl∈ *admincolor*(u1)∧ cl∈*admincolor*(u2) | *admincolor′*(u2)← $\phi$ |
| **Add_Cl$_{\mathbf{VMM}}$**(u,vmm,cl) /*Adds a color cl to a Host vmm*/ | u∈AU∧vmm∈VMM∧cl∈ *admincolor*(u)∧*adminRole*(u)= tvdcAdmin∧*adminvdcenter*(u)= *vmmvdcenter*(vmm) | *vmmcolor′*(vmm)← *vmmcolor*(vmm)∪ {cl} |
| **Assign_Cl$_{\mathbf{VM}}$**(u,vm,cl) /*Assigns a color cl to a VM vm*/ | u∈AU∧vm∈VM∧cl∈ *admincolor*(u)∧*adminRole*(u)= tvdcAdmin∧*adminvdcenter*(u)= *vmvdcenter*(vm) | *vmcolor′*(vm)← {cl} |
| **Assign_Cl$_{\mathbf{BR}}$**(u,br,cl) /*Assigns a color cl to a BR br*/ | u∈AU∧br∈BR∧cl∈ *admincolor*(u)∧*adminRole*(u)= tvdcAdmin∧*adminvdcenter*(u)= *brvdcenter*(br) | *brcolor′*(br)←{cl} |
| **Add_Cl$_{\mathbf{VLAN}}$**(u,vlan,cl) /*Adds a color cl to a VLAN vlan*/ | u∈AU∧vlan∈VLAN∧cl∈ *admincolor*(u)∧*adminRole*(u)= tvdcAdmin∧*adminvdcenter*(u)= *vmmvdcenter*(vlan) | *vlancolor′*(vlan)← *vlancolor*(vlan)∪ {cl} |

Now, table 4.7 shows the administrative operations for tenant admin-users and preconditions to authorize these operations. The operations of the tenant admin-users are to manage cloud resources within their assigned TVD groups, i.e., colors. The operations are described as follows:

- **Boot**: Using this operation a tenant admin-user u boots a VM vm in a Host vmm. Table 4.7

**Table 4.7**: Tenant-ADMIN Operations

| Operation | Precondition | Updates |
|---|---|---|
| **Boot**(u,vm,vmm) /*Boots a VM vm in a Host vmm*/ | *vmcolor*(vm)∈*admincolor*(u)∧ *admincolor*(u)∩ *vmmcolor*(vmm)≠ ∅∧*adminvdcenter*(u)= *vmvdcenter*(vm)∧*adminvdcenter*(u)∈ *vmmvdcenter*(vmm)∧*vmcolor*(vm)∈ *vmmcolor*(vmm)∧vm∈VM∧vmm∈ VMM∧ Evaluate_CLocConst(vm,vmm)∧ u∈AU∧*adminRole*(u)=tntAdmin∧ *status*(vm)=Stop | *host′*(vm)← vmm *vm′*(vmm)← *vm*(vmm)∪vm *status′*(vm)← Running |
| **ConVmToBr**(u,vm,br) /*Connects a VM vm to a BR br*/ | u∈AU∧*vmcolor*(vm)∈*admincolor*(u)∧ *brcolor*(br)=*vmcolor*(vm)∧*brcolor*(br)∈ *admincolor*(u)∧br∈BR∧vm∈VM∧ *adminvdcenter*(u)=*vmvdcenter*(vm) ∧ *adminvdcenter*(u)=*brvdcenter*(br)∧ *adminRole*(u)=tntAdmin | *bridge′*(vm)← br *vm′*(br)← *vm*(br)∪ {vm} |
| **ConBrToVLAN**(u,br,vlan) /*Connects a BR br to a VLAN vlan*/ | u∈AU∧*brcolor*(br)∈*admincolor*(u)∧ *brcolor*(br)∈ *vlancolor*(valn)∧ *vlancolor*(vlan)∩*admincolor*(u)≠ ∅∧br∈BR∧vlan∈VLAN∧ *adminvdcenter*(u)=*vlanvdcenter*(vm)∧ *adminvdcenter*(u)=*vlanvdcenter*(br) ∧*adminRole*(u)=tntAdmin | *bridge′*(vlan)← *bridge*(vlan)∪ {br} *vlan′*(br)← vlan |

shows the necessary precondition in order to authorize this operation. The precondition verifies if the u has same color of the vm which is basically an implementation of the management isolation constraint shown in section 4.3.1. In addition to that, the precondition also checks if these three entities belong to the same data-center. It also verifies if the Host vmm's *vmmcolor* attribute contains the color of the vm's assigned color in *vmcolor* which is an implementation of Host authorization isolation which is also shown in section 4.3.1. The authorization process of this operation also calls Evaluate_CLocConst function to satisfy the co-location management constraint, also given in section 4.3.1, for vm with other running VMs in vmm. The algorithm 4.1 shows the evaluation process of Evaluate_CLocConst. Upon successful checking of these conditions vm is scheduled to the vmm. In chapter 6, we

**Algorithm 4.1** Colocation Constraints Verification

```
1: procedure Evaluate_CLocConst (reqVm,vmm)
2:     Flag=True
3:     for all vm ∈ VM do
4:      if host(vm)=vmm then
5:        for all conele∈ConflictColor do
6:          if vmcolor(reqVm)≠ vmcolor(vm) then
7:            if vmcolor(vm)∈conele∧vmcolor(reqVm)∈conele∧status(vm)=Running then
8:              Flag=False
9:              Return Flag
10:            end if
11:          end if
12:       end for
13:     end if
14:   end for
15:   Return Flag
16: end procedure
```

develop and analyze a novel constraint-driven virtual resource scheduling process.

- **ConVmToBr**: It connects a VM vm to a BR br. A VM can only connects to br if they have same color and they belongs to the same data-center.

- **ConBrToVLAN**: Using this function a tenant admin-user connects a BR br to a VLAN vlan. They can be connected if color of br is present in the *vlancolor* attribute of vlan.

Algorithm 4.1 shows the evaluation algorithm of the co-location constraints. It takes two inputs: requested VM (reqVm) and the Host (vmm). For each VM running in the vmm, this algorithm verifies if there is any conflict between the assigned color to the *vmmcolor* attribute of VM with the assigned color to the *vmmcolor* of reqVm. Attribute values can have different type of conflicts that can represent various relationships among these such as mutual-exclusion, precondition, etc. A generalized approach to represent the various types of attribute conflict-relations are shown chapter 3. Here, the conflicting values, i.e. colors, of the attribute *vmmcolor* are stored in a set called ConflictColor where each element in the set contains a set colors that are conflicting with each-other. Formally this set is defined as follows,

ConflictColor = {conele$_1$, conele$_2$, ..., conele$_n$} where conele$_i$ $\subseteq$ CLR

If algorithm 4.1 identifies no conflicts between reqVm and all running VMs in vmm, it returns True. Otherwise, it returns False.

# Chapter 5: THE CVRM MODEL

*The materials in this chapter are published in following venues [21,26]*:

1. Khalid Bijon, Ram Krishnan and Ravi 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 (CODASPY), March 2-4, 2015, San Antonio, Texas.

2. Khalid Bijon, Ram Krishnan and Ravi Sandhu, Automated Constraints Specification for Virtual Resource Orchestration in Cloud IaaS. Under Review in IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), 2015.

We discuss our developed attribute based CVRM (constraint-driven virtual resource management process). CVRM provides a language to specify constraints which is a customized version of ABCL that is suitable for this purpose. We also provide an enforcement guideline for these constraints in the OpenStack cloud platform.

## 5.1 Motivation

Migrating line-of-business applications to IaaS can be risky for cloud 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. Currently commercial cloud IaaS providers, including Amazon and Rackspace, offer at best rudimentary capabilities for such configuration management. For instance, AWS-IAM [2] 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 (STR) 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 since there is no independent mechanism to detect or prevent those misconfiguration.

Motivated by these considerations, we aim to develop CVRM that offers a tenant means 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, a 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 5.1 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 applications 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. Different kinds of virtual resource to virtual resource mappings are briefly discussed

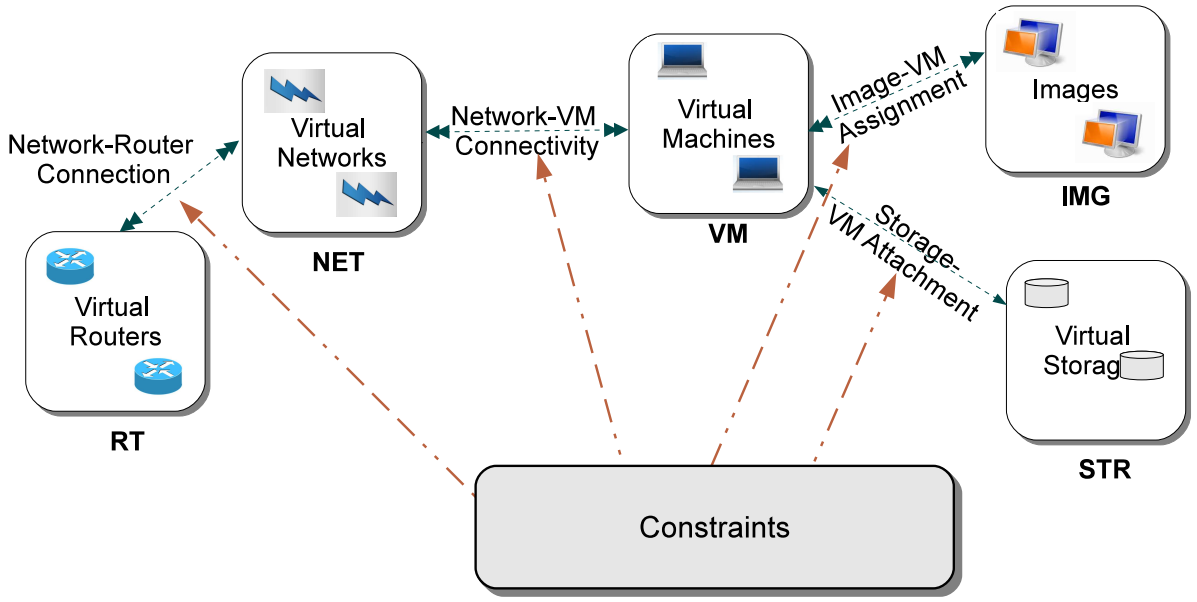


**Figure 5.1**: Constraints on Virtual Resources Arrangement Configurations

below.

- **IMG-VM Compatibility Relations:** As shown in figure 5.1, a virtual machine image, also referred as IMG, can be used by multiple VMs. Conversely, from a VM multiple snapshots can be imaged. This process provides quick replication of a VM into large numbers of cloned VMs, and also quick migration of a VM to another server. However, incorrect usage of an 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 an IMG can hamper performance and availability.

- **NET-VM Connectivity Relations:** A group of VMs, connect to a virtual 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.

## 5.2 Design of CVRM

As discussed in earlier chapters, 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 constraints specification language. Then, we instantiate CVRM for 3-tier architecture and hadoop cluster in cloud IaaS.

### 5.2.1 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 $\rightarrow$ CLS, is a function that maps each virtual resource to its class.
- TENANTS is the finite set of existing tenants in CSP.
- *tenant* : VR $\rightarrow$ TENANTS, is a function that maps each virtual resource to the tenant that owns it.
- $VR_{tnt}$ is the set of virtual resources that are owned by the tenant `tnt`. Formally, $VR_{tnt}=\{v \in VR \mid tenant(v)=\texttt{tnt}\}$.

Here, $VR_{tnt}$ 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,

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

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 respec-

tively. We can define the relations between elements of every two classes of virtual resources in a tenant as follows,

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

  $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j} \subseteq \mathsf{VR}_{\mathsf{tnt},\mathtt{c}_i} \times \mathsf{VR}_{\mathsf{tnt},\mathtt{c}_j}$.

However, CVRM restricts the following type of relations,

1. Relations between virtual resources of same class cannot be specified (i.e., $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_i}$ cannot be specified).

2. For two classes $\mathtt{c}_i \neq \mathtt{c}_j$ we can define $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ or $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_j,\mathtt{c}_i}$ but not both.

CVRM provides two operations called $\mathcal{A}dd$ and $\mathcal{R}M$ 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 [57]),

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

$$\mathcal{R}M(\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}, \mathrm{vr}_1, \mathrm{vr}_2) \lhd$$
$$\mathrm{vr}_1 \in \mathsf{VR}_{\mathsf{tnt},\mathtt{c}_i} \wedge \mathrm{vr}_2 \in \mathsf{VR}_{\mathsf{tnt},\mathtt{c}_j} \wedge \mathit{consEval}(\delta^{\mathcal{R}M}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}, \mathrm{vr}_1, \mathrm{vr}_2)$$
$$\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}{}' = \mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j} - \{<\mathrm{vr}_1, \mathrm{vr}_2>\} \rhd$$

Here, $\delta^{\mathcal{A}dd}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ and $\delta^{\mathcal{R}M}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ are constraints that are respectively specified for adding and removing a tuple to the relation $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$. 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 $\mathcal{R}M$ 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 $\mathsf{c}_i$ 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 $\mathsf{SCOPE}_{att}$ 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 $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$, at most two constraints can be specified for the operations $\mathcal{A}dd$ and $\mathcal{R}M$ respectively. Each constraint is used to verify if assigned values of specific attributes of two virtual resources $vr_1$ and $vr_2$ of class $\mathsf{c}_i$ and $\mathsf{c}_j$ respectively satisfy certain conditions. CVRM uses the grammar in table 5.1 to specify constraints.

The constraints specification grammar syntax is given in Backus Normal Form (BNF). Basically, it is a restrictive version of the language ABCL given in table 3.2. Each constraint statement contains single or multiple small expressions in the form of implication, A→B, joined by logical

[1] As given in section 5.1, 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.

**Table 5.1**: Constraint Specification Grammar

| |
|---|
| <Quantifier>:= $\forall$(vr1,vr2) $\in \mathcal{R}_{<Cls>,<Cls>}$ . <Stmt> |
| <Stmt>:= <Stmt> <connector><Stmt> \| (<rule>) |
| <rule>:= <Token> $\rightarrow$ <Token> |
| <Token>:= (<Token> <connector> <Token>)\|(<Term>) |
| <Term>:= <Attribute>(<resource>) <comperator> <Scope> |
| <Attribute>::= <letter> \| <digit> \| <Attribute> |
| <Scope>::= <letter> \| <digit> <Scope> |
| <connector>::= $\wedge$ \| $\vee$ |
| <comparator>::= = \| $\neq$ |
| <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 |

connectors. The small expression is also referred to as constraint-rule or just rule. Both A and B in a rule A$\rightarrow$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$\rightarrow$B, verifies that if assigned attribute values of a virtual resource vr1 meet 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>. This is formally defined as follows.

**Predicate format:**

attribute(<Resource>) <comparator> attribute-value

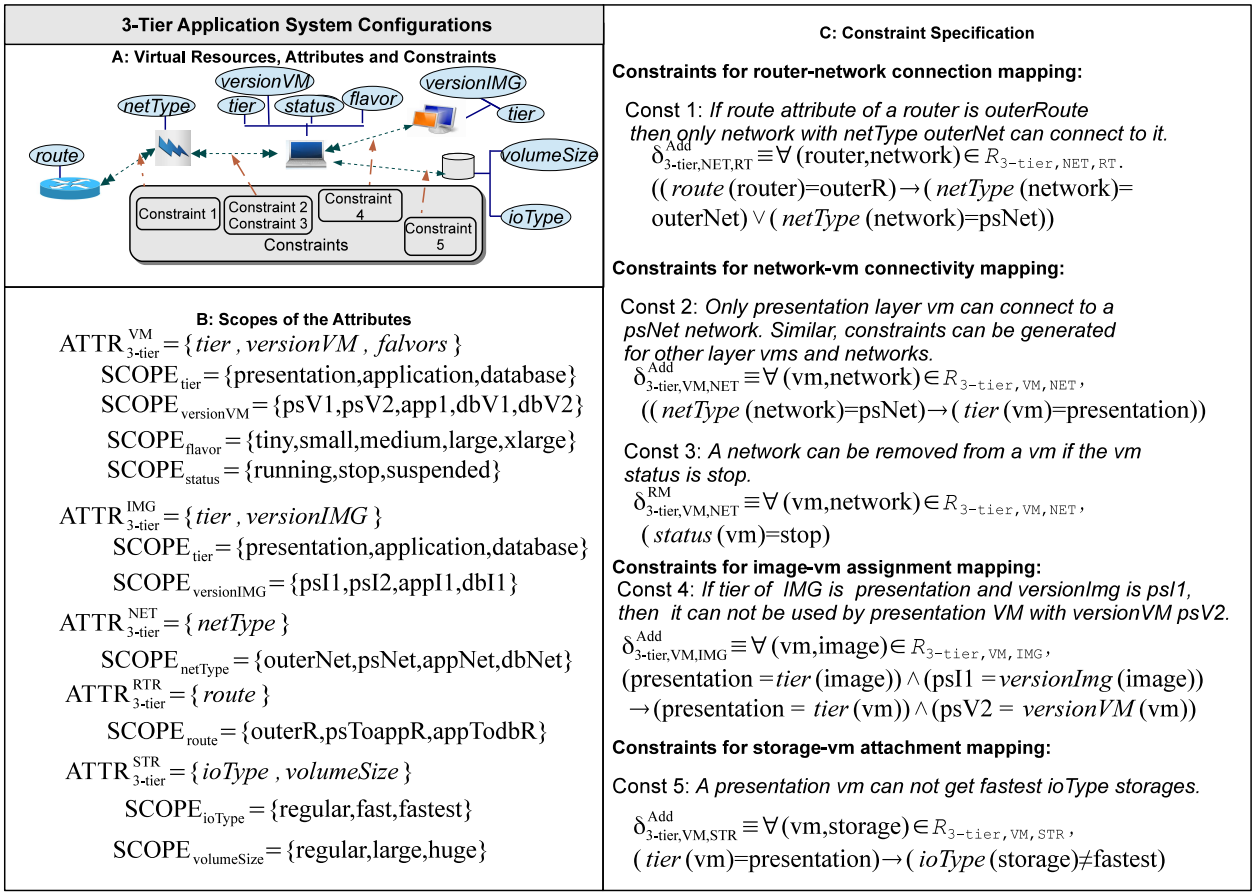


**Figure 5.2**: Constraints Specification for 3-Tier Application System

**Type-Checking Rule:**

$If$ attribute-value $\in$ $\mathsf{SCOPE}_{\text{attribute}}$ $Then$

return $true$

$Else$

return $error$

### 5.2.2 Instantiation

In this section, we instantiate CVRM for an example of 3-tier business application setup and an example of hadoop cluster setup.

**3-Tier Business Application**

We focus on the tenant called `3-tier`. The classes of virtual resources supported by the CSP are VM, NET, RT, STR and IMG. `3-tier` supports relations from VM-to-NET, NET-to-RT, VM-to-IMG and VM-to-STR which are written as $\mathcal{R}_{\texttt{3-tier,VM,NET}}$ , $\mathcal{R}_{\texttt{3-tier,NET,RT}}$, $\mathcal{R}_{\texttt{3-tier,VM,STR}}$ and $\mathcal{R}_{\texttt{3-tier,VM,IMG}}$ 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 5.2-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 5.2-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 represent 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. Figures 5.2-A and 5.2-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 requirements. 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 5.2-A, for their operational purpose. Here, $tier$ is intra-tenant attribute since this attribute is not mean-

ingful 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 research. However, there is literature on attribute administration [79] that might apply in this context.

After the 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 5.2-C. For instance, constraint $\delta^{\text{Add}}_{\texttt{3-tier},\mathsf{VM},NET}$ 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 $\delta^{\text{RM}}_{\texttt{3-tier},VM,NET}$ applies to $\mathcal{R}M$ operation of same relation where it checks if the a vm is in stop state to disconnect it from a virtual network. Figure 5.2-C also shows example constraints for other relations.

**Hadoop Cluster Setup**

The set TENANTS contains the tenant hadoop. The classes of the virtual resources supported by the CSP are VM, NET, and RT and specified relations are between VM-to-NET and NET-to-RT. The relations are represented as $\mathcal{R}_{\mathsf{hadoop,VM,NET}}$ and $\mathcal{R}_{\mathsf{hadoop,NET,RT}}$.

In this simple hadoop setup, we only define one attribute for each virtual resources (shown in figure 5.3-A). Here, a VM attribute $nodeType$ represent the type of operations a vm performs in hadoop cluster and figure 5.3-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

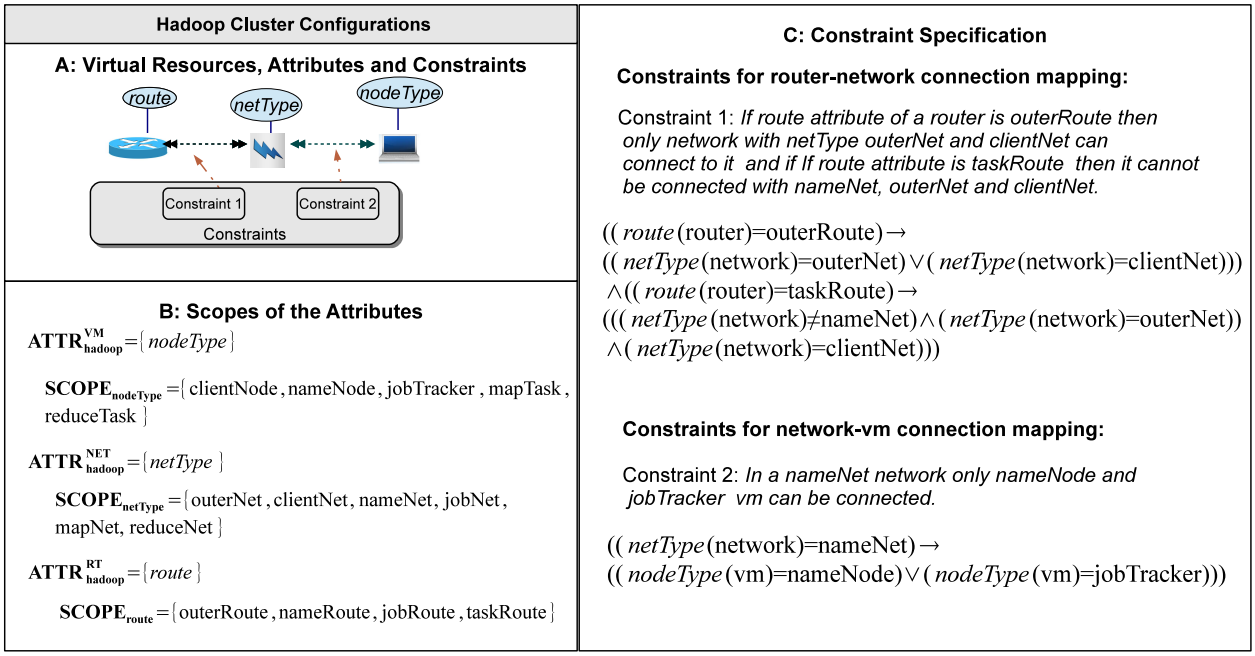


**Figure 5.3**: Constraints Specification for Hadoop Cluster

configuration management.

After attributes specification of the resources, we show two constraints for adding elements in each of the relations (shown in figure 5.3-C). Here, for instance, constraint $\delta^{\text{Add}}_{\text{hadoop,NET,RT}}$ applies to the $\mathcal{A}dd$ 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.

## 5.3 CVRM Enforcement

We describe a CVRM enforcement in OpenStack cloud platform. Basically, this discussion and implementation of the enforcement process are based on the Havana release of OpenStack [8]. We also analyze some security issues of CVRM.

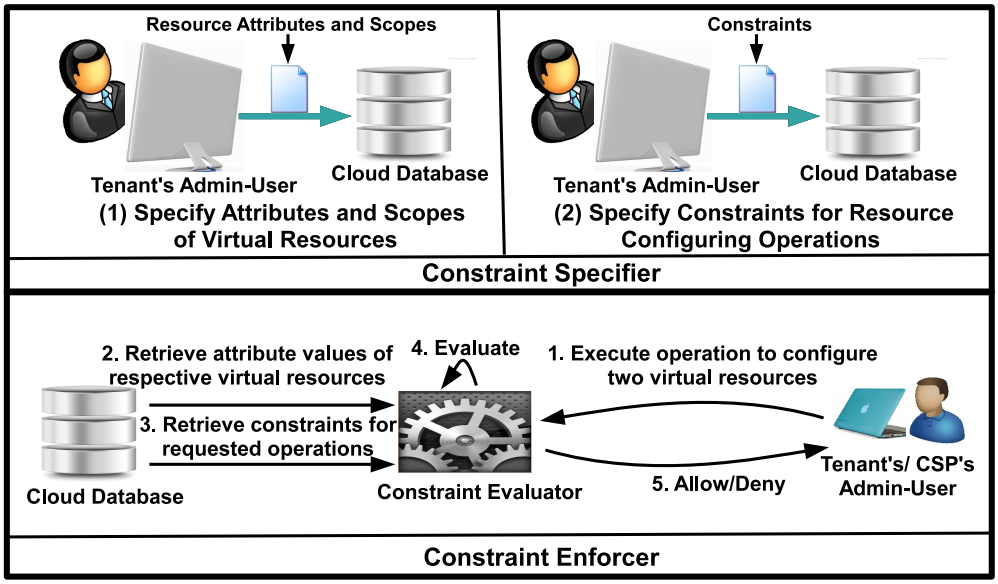


**Figure 5.4**: Components of CVRM Enforcement Process in a Service of OpenStack

### 5.3.1 Enforcement in OpenStack

Figure 5.4 shows a 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 evaluator retrieves the attributes of the virtual resources and the respective constraints from cloud database and evaluates the constraints to make decision.

### 5.3.2 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 service operations on 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 manage the cloud such as creating database tables, create and delete other domains, add/remove users to them, design authorization policies for user access request to service APIs. 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 information including the projects where the user is a member and the assigned roles. The operation is allowed if the project of respective virtual-resources are same as the requesting user.

Figure 5.5 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 if the user is assigned to *project-admin* role. Also, it collects the tenant information of VM and STR from its database and it approves if the given user, VM and STR are in same tenant.

### 5.3.3 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

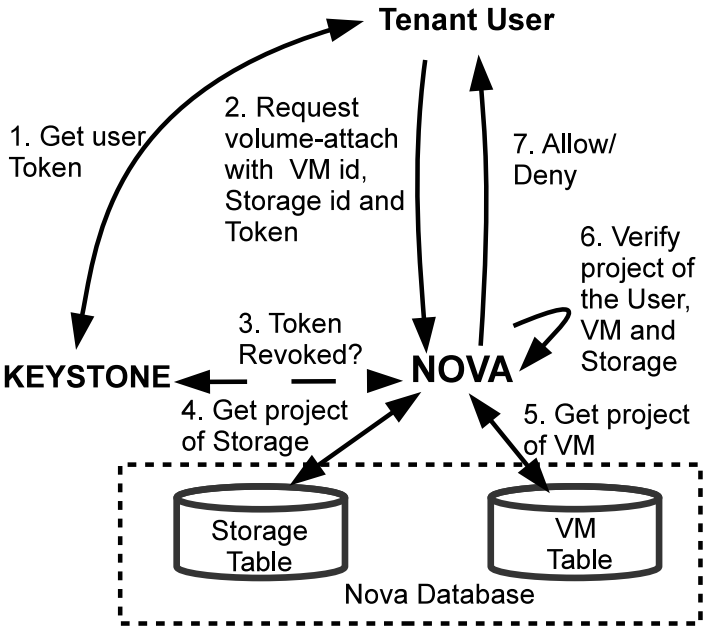


**Figure 5.5**: Operation *volume-attach* in Nova

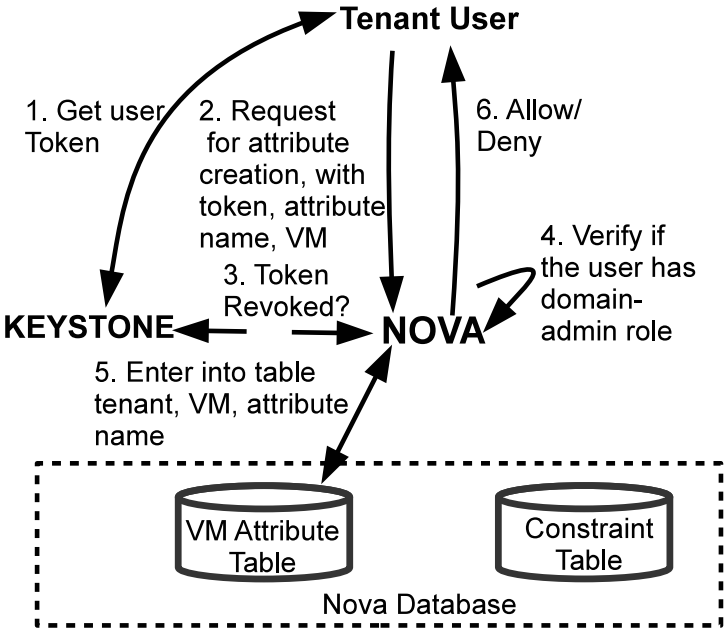


**Figure 5.6**: Constraint Specifier in Nova

domain. Note that, present OpenStack supports *domain-admin* roles only for operations in keystone, however, it can be included to other OpenStack services such as Nova as well. Similarly, specifier also provides operations for constraints specification. Each constraint is mapped to an

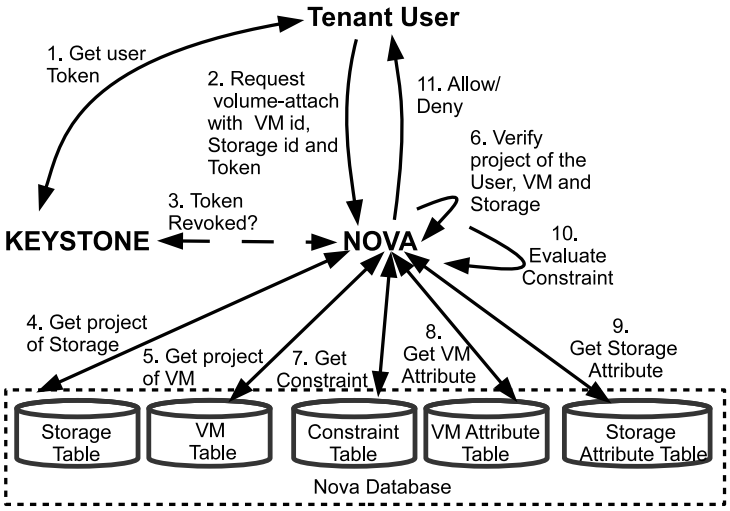


**Figure 5.7**: Constraint Enforcement for *volume-attach*

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 databases of respective service. Entries in a database table across tenants (domains) are isolated by specific domain ids so that admin-users of a domain cannot access other domains' information. Constraint specifier also provides operations to assign values to the attributes of the virtual resources supported by each service in OpenStack. Unlike previous operations in OpenStack, this operation should be authorized for the *project-admin* users of respective projects. Detailed implementation of these operations is discussed in 5.4.

Figure 5.6 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.

### 5.3.4 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, en-

forcer 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 5.7 shows extended view of figure 5.5 for the execution of *volume-attach*. Besides, comparing the project information of the VM, STR 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.

### 5.3.5 Security Concerns

We present different security issues for enforcing CVRM in practice.

**Constraints Specification 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 should be authorized only to selected admin-users of a tenant. 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 [24] 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 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). Here, we do not focus on such access control system, however, existing mechanisms such as [24] 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.

**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 enables the 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, $\text{val}_x$ of an attribute $att_p$ of the virtual resources of specific class $\mathtt{c}_i$ can have mutual exclusion relation with all the values of an attribute $att_q$ of the virtual resources of class $\mathtt{c}_j$. Then, the virtual resources of class $\mathtt{c}_i$ with assigned value $\text{val}_x$ cannot be arranged with any resources of $\mathtt{c}_j$. These constraints are deadlock constraints and tenants need to handle them properly.

- **Redundant CVRM Expressions:** In CVRM, a constraint expression is redundant if it specified multiple times. Redundant expression unnecessarily increase the run-time complexity. One such example of a redundant expression is multiple occurrences of same sub-expression in a constraint expression.

## 5.4 Prototype Implementation in OpenStack

We describe the implemented prototype of CVRM enforcement process. We leverage the DevStack cloud framework [5], 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. Our python-based implementation of constraint specifier, that includes API design to enable users to declare attributes and constraints and the constraint enforcer that includes a constraint parser to evaluate constraint expressions.

### 5.4.1 OpenStack Constraints API

In the following, we describe the developed OpenStack APIs for declaring attributes, their scopes, and constraints in constraint specifier component. We also specify APIs specification for the attributes assignment to virtual resources. Here, we show the APIs for Nova (compute service), however, similar APIs are specified for other services in OpenStack. These APIs are REST APIs where each API comprises a base url to connect to the particular end-point and one of the generic HTTP methods (GET, POST, DELETE or PUT).

Table 5.2 shows the APIs (with HTTP method and URL) that are registered to Nova APIs of version 2.0 in OpenStack. Here, API 1-3 and 4-6 are for managing attributes and their scopes respectively. Then, API 7-9 are for specification of the constraints for a relation of two specific virtual resources. Finally, API 10 is to set and remove attribute-value to the virtual resources. Note that, API 10 is already built-in to OpenStack installations to set and remove meta information of the virtual resources where each field in meta is a *key:value* pair. We use this for assigning attribute-values to VM where the *key* is the name of VM attribute and *value* is a value from the attribute scope. We also develop the same APIs, as shown in table 5.2, in OpenStack block-storage service called Cinder that manages the STRs. Note that, these APIs can be included in an OpenStack deployment by admin-users of the CSP.

In this specification, each POST and PUT request requires a request body whereas DELETE and GET do not. A request body is represented as a JSON dictionary format. For instance, from table 5.2, request body of the *att-create*, which creates the name of an attribute, has the format $\{\text{“}attribute\text{”} : \{\text{“}name\text{”} : \text{“}\{attname\}\text{”}\}\}$.

**Table 5.2**: OpenStack *Nova* API for CVRM Specifications

| Name | URL | Type |
|---|---|---|
| **1. att-create**<br>Create an attribute | $/v2/\{tenant\_id\}/attributes$ | POST |
| **2. att-delete**<br>Delete an attribute | $/v2/\{tenant\_id\}/attributes/\{id\}$ | DELETE |
| **3. att-list**<br>List all attributes | $/v2/\{tenant\_id\}/attributes$ | GET |
| **4. att-value-set**<br>Add a value | $/v2/\{tenant\_id\}/scopes$ | PUT |
| **5. att-value-del**<br>Delete a value | $/v2/\{tenant\_id\}/scopes/\{id\}$ | DELETE |
| **6. att-value-list**<br>Get values of attribute | $/v2/\{tenant\_id\}/scopes$ | GET |
| **7. policy-add**<br>Add a constriant | $/v2/\{tenant\_id\}/policies$ | POST |
| **8. policy-del**<br>Delete a constraint | $/v2/\{tenant\_id\}/policies/\{id\}$ | DELETE |
| **9. policy-list**<br>Get a constraint | $/v2/\{tenant\_id\}/policies/\{id\}$ | GET |
| **10. meta**<br>Assign or deassign attribute-value | $/v2/\{tenant\_id\}/servers/\{resource\_id\}/metadata$ | POST |

| Field | Type |
|---|---|
| id | Integer |
| name | String(255) |
| project_id | String(255) |

(a) The *attribute* Table

| Field | Type |
|---|---|
| id | Integer |
| name | String(255) |
| value | String(255) |
| project_id | String(255) |

(a) The *scope* Table

| Field | Type |
|---|---|
| id | Integer |
| relation_name | String(255) |
| expression | String(255) |
| project_id | String(255) |

(c) The *constraints* Table

| Field | Type |
|---|---|
| vm_id | Integer |
| vm_id | String(255) |
| meta | String(255) |

(d) The instance-*metadata* Table

**Figure 5.8**: Database Schema

Here, $\{attname\}$ is the name of the attribute specified by the users during executing the *att-create* command. Attributes, their scopes and constraints are stored in Nova database (high level database tables are shown in figure 5.6). We design a database schema, as shown in figure 5.8 for Nova which contains 4 main tables: *attribute*, *scope*, *constraints* and *instance_metadata*. The table *attribute* stores the declared attribute names and *scope* stores their values. The constraint represents the specified constraints for a particular relations between two specific virtual resources. Here, *instance_metadata* is a built-in table in Nova that stores the instance meta-data and we use this table to store assigned values to the attributes of VMs. Similar to the API specifications, databases are included in an OpenStack deployment by the CSP's admin-users.

As discussed in section 5.3, specification of attributes, their scopes and the constraint-expressions are authorized only to the *domain-admins*, while the attribute-value assignment to virtual resources and their arrangement are authorized to *project-admins*. API 1-9, shown in table 5.2, are authorized only for the users who are *domain-admins* of a particular tenant. Presently, Nova does not support *domain-admins*, while, the authorization component of OpenStack called *keystone* supports. Therefore, Nova policy specification process needs to be extended so that it can support

the *domain-admins*. In Nova, a file called *policy.json* specifies authorization policies for the APIs. Necessary policies for the APIs, shown in table 5.2, can be included in *policy.json*. For instance, the following policy can be included for *attcreate* API.

*rule:attcreate* $\rightarrow$ *(domain_role*:*domain_admin* $\&\&$ *domain_id*:$\%$*(target_domain_id)s))*

In the above policy, it says that users having the *domain_role* can only call the *attcreate* APIs for creating an attribute within his domain. When a user tries to create an attribute name, sf retrieves this policy from *policy.json* and verifies against the user's information given in the authorization token from keystone. Note that, these policies are specified by the CSP's admin-users and can only be modified and altered by them. The admin-users of individual tenants are not authorized to make any changes in *policy.json*.

### 5.4.2 Constraints Verification in OpenStack

Constraints verification process includes the well-formedness validation of the constraint expression and evaluation of the expression to a boolean value.

An specified constraint expression is a character string generated by the language shown in section 5.2. The expression is a collection of predicates where each predicate contains an attribute name and attribute value. For well-formedness validation of a constraint expression checks if the the attributes name are valid and the specified value belongs to the scope of the attribute. In this implementation, we also capture the concept of intra and inter tenant attributes which we already discussed in section 5.2.2. Intra-tenant attributes are completely owned and managed by an individual tenant using the APIs shown in table 5.2, while, inter-tenant attributes are managed by IaaS admin-users and all tenants can view and use these attributes to specify the constraints. Examples of these attributes in Nova are the system attributes of VMs such as flavors, image-names, etc. During specification of a constraints, the specified attributes and the values are validated accordingly with respect to the intra-tenant and system attributes in OpenStack. Figure 5.9 shows

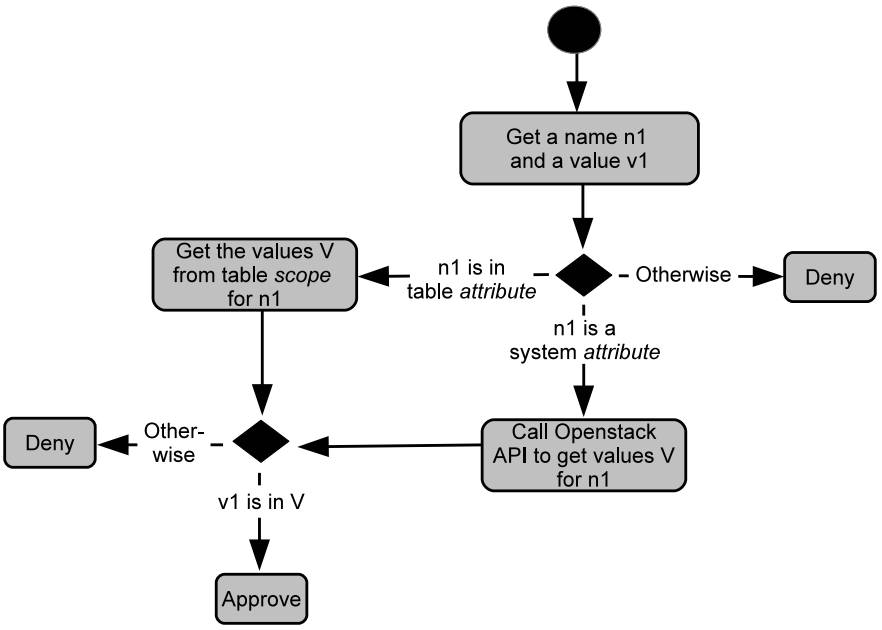


**Figure 5.9**: Constraints Well-Formedness Validation in OpenStack: An Activity Diagram

the well-formedness validation process which for each predicate of a constraint expression takes as input the specified attribute name and value and checks if the name is a valid attribute name and the value belongs to the scope of the attribute. Once validated, a constraint is then stored in database table. Recall figure 5.2 where the example of storage-vm connection mapping constraint expression contains both intra- and inter-tenant attribute which is expressed as follows.

((*tier*(vm)=presentation) $\rightarrow$ (*ioType*(storage)$\neq$fast))

Here, *tier* and *ioType* are intra-tenant and system attributes respectively. Therefore, for *tier*, as shown in figure 5.9, validation process checks in the *attribute* table for name and *scope* table for value. Whereas, for *ioType* respective APIs are checked.

Finally, the specified constraints need to be enforced when a respective *project-admin* user of a project tries to relate two virtual resources (an example is shown in figure 5.7). This process 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 the two virtual resources. The

**Table 5.3**: min_rule Specification Grammar

| |
|---|
| <Quantifier>:= $\forall$(vr1,vr2) $\in \mathcal{R}_{<\text{Cls}>,<\text{Cls}>}$ . <Stmt> |
| <Stmt>:= <Stmt> <connector><Stmt> \| (<min_rule>) |
| <min_rule>:= <Token> $\rightarrow$ <Token> |
| <Token>:= <Attribute>(<resource>)<comperator><Scope> |
| <Attribute>::= <letter> \| <digit> \| <Attribute> |
| <Scope>::= <letter> \| <digit> <Scope> |
| <connector>::= $\wedge$ \| $\vee$ |
| <comperator>::= = \| $\neq$ |
| <Cls>::= $\mathsf{c}_1$ \| $\mathsf{c}_2$ \| . . . \| $\mathsf{c}_n$ |
| <resoruce> ::= vr1 \| vr2 |
| <digit>::= 0\|1\|2\| . . . \|8\|9 |
| <letter>::= a\|b\|. . . \|y\|z\|A\|B\|. . . \|Y\|Z |

parser retrieves the respective attribute values of the virtual resources from the respective tables that stored the metadata and verifies the values against the constraint expressions.

## 5.5 Automated Constraint Construction

In this section, we consider approaches for mining CVRM constraints from already specified relations among the instances of two classes of virtual resources. This process can be considered as the automatic generation of constraints according to a tenant's specific requirements.

In this process, a collection of restricted rules, also refereed as min_rule, are generated where a min_rule is an implication, a$\rightarrow$b, in which both a and b are single predicates. Note that, the actual rule in the form of A$\rightarrow$B as defined in section 5.2.1 allows both A and B to be collections of predicates connected by $\wedge$ and/or $\vee$. Now for a given $\mathcal{R}_{\mathsf{tnt},\mathsf{c}_i,\mathsf{c}_j}$, $\delta^{Add}_{\mathcal{R}_{\mathsf{tnt},\mathsf{c}_i,\mathsf{c}_j}}$ and $\delta^{\mathcal{R}M}_{\mathcal{R}_{\mathsf{tnt},\mathsf{c}_i,\mathsf{c}_j}}$, a min_rule can be generated by the grammar in table 5.3.

Each min_rule is restricted to specify a comparison between only two attributes of virtual resource classes $\mathsf{c}_i$ and $\mathsf{c}_j$. Now let us say $\mathcal{R}_{\mathsf{tnt},\mathsf{c}_i,\mathsf{c}_j}$ is a given set of tuples that specifies the relation between instances of the two classes $\mathsf{c}_i$ and $\mathsf{c}_j$. The min_rule mining problem is to construct all possible min_rules. For given $\mathcal{R}_{\mathsf{tnt},\mathsf{c}_i,\mathsf{c}_j}$, $\mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, $\mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, $\text{att}_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$, $\text{att}_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$, min_rules can be of four following formats, where each val has to

belong to the appropriate attribute $\mathsf{SCOPE}_{att}$.

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

For simplicity, we provide a mining algorithm for the format a$\rightarrow\overline{\text{b}}$ which is also referred as mutual exclusive min_rule. An example of the mutual exclusive constraints is storage-vm attachment mapping constraint in figure 5.2 where the constraint specifies a comparison between the value 'presentation' of VM attribute $tier$ to the value 'fast' of STR attribute $ioType$. Similar algorithms can be also generated for other min_rule 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 [86] where they also mine mutual exclusive roles, so it is feasible to compare mutual exclusive min_rule to mutual exclusive roles.

### 5.5.1 Mining Overview

Figure 5.10 shows the steps of our min_rule mining approach. As seen in the figure, a configuration-log stores the assigned relations between instances of of virtual resources two classes c$_i$ and c$_j$ , e.g., VM-STR attachments, of particular tenant `tnt`. We can consider the configuration-log as $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$. We also assume that the instances of these virtual resources are assigned with values to their attributes. Now, our first step is to find the candidate attribute relations from which appropriate min_rules of one of the four formats can construct. We define a candidate attribute relation as a binary relation between the values in $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$ of attributes att$_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ and att$_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$ respectively which can be calculated by a frequent item-set mining algorithm from $\mathcal{R}_{\mathtt{tnt},\mathsf{c}_i,\mathsf{c}_j}$. In section 5.5.2, we describe our developed mining algorithm for candidate attribute relation which is based on well-known Apriori algorithm [11]. Step 2 is to extract certain attribute

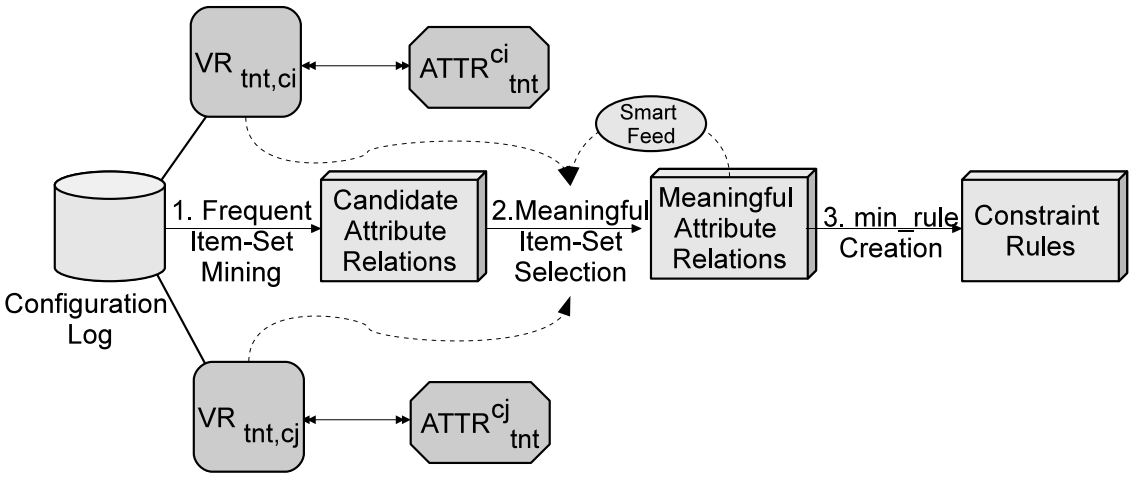


**Figure 5.10**: Overview of the Constraint Mining

relations from candidate attribute relations which have semantic meaning with correspond to the cloud IaaS system. We define meaningful attribute relations as a binary relation between the values in $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$ of attributes $att_p \in \mathsf{ATTR}^{c_i}_{tnt}$ and $att_q \in \mathsf{ATTR}^{c_j}_{tnt}$ respectively which is useful for the current sets of the respective virtual resources in the system such that generated constraints from that relation will have impact on future configuration requests.Then, final step is to construct the min_rule from the set of meaningful attribute relations.

### 5.5.2 Candidate Attribute Relation Construction

The values in $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$ of attributes $att_p \in \mathsf{ATTR}^{c_i}_{tnt}$ and $att_q \in \mathsf{ATTR}^{c_j}_{tnt}$ respectively is considered as candidate attribute relations is specified as a relation called $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$. $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is reflexive and symmetric, but not transitive. Hence, each element in $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is an unordered pair. For any two attributes $att_p \in \mathsf{ATTR}^{c_i}_{tnt}$ and $att_q \in \mathsf{ATTR}^{c_j}_{tnt}$, $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is defined as follows.

$$\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}} \subseteq \{\ \{x,y\} \mid x \neq y \text{ and } x \in \mathsf{SCOPE}_{att_p} \text{ and } y \in \mathsf{SCOPE}_{att_q}\}$$

For instance, in order to construct min_rule for STR-VM attachment constraint in figure5.2, a

candidate attribute relation is $\text{CAR}_{\text{SCOPE}_{tier},\text{SCOPE}_{ioType}}$ where an element of it is {presentation,fast}. In the following, we develop a mining approach to identify the elements in a $\text{CAR}_{\text{SCOPE}_{att_p},\text{SCOPE}_{att_q}}$. Specifically, the approach identifies the mutual exclusive relations that will construct mutual exclusive min_rules. However, similar approaches can be developed for other three type of min_rules.

**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 [11] 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) [86] which is developed on top of the Apriori algorithm [11]. In RBAC, *u* and *role* 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 $\in$ *role* is an implication of the form R1$\rightarrow\overline{\text{R2}}$ where R1 $\subset$ *role* and R2 $\subset$ *role* and R1 $\cap$ R2=$\emptyset$ and $user\_roles$(u) $\subseteq$ R1 $\rightarrow$ $user\_roles$(u) $\cap$ R2 = $\emptyset$ for each user u $\in$ *u*. Let *D* be a set of user-role assignments, the constraint R1$\rightarrow$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 R1$\rightarrow\overline{\text{R2}}$ holds for *D* if it has certain user-specified minimum support and confidence.

**Mining Candidate Attribute Relation in CVRM**

In this section, we discuss the mining approaches for mutual exclusive candidate attribute relations. We first utilize the anti-Apriori algorithm [86] for the mining. Then, we customize the anti-Apriori algorithm, which we call CVRM-Apriori, in order to get better performance.

**A. Reduction to RBAC constraint mining:** In this approach, we identify inputs of a mutual exclusive candidate attribute relation mining algorithm and reduce them to the inputs of anti-Apriori.

Then, we collect the outputs from anti-Apriori algorithm and construct candidate attribute relation.

**Inputs of a min_rule mining algorithm:** In CVRM, each mutual exclusive attribute relation is 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 $\mathcal{R}_{\mathtt{tnt},c_i,c_j}$, $\mathsf{VR}_{\mathsf{tnt},c_i}$, $\mathsf{VR}_{\mathsf{tnt},c_j}$, and for each $att_p \in \mathsf{ATTR}^{c_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{c_j}_{\mathsf{tnt}}$, the inputs are $\mathsf{VR}_{\mathsf{tnt},c_i}$, $\mathsf{VR}_{\mathsf{tnt},c_j}$, $\mathcal{R}_{\mathtt{tnt},c_i,c_j}$, $att_p$, $att_q$, $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$.

**Inputs of anti-Apriori:** The inputs of the anti-Apriori algorithm are *u*, *role*, the user-role assignment matrix M (M is a u×r dimension boolean matrix where u and r is the size of *u* and *role* and for each $u_i \in$ *u* and $r_j \in$ *role*, M[$u_i$][$r_j$]=1 if $r_j \in user\_roles(u_i)$ and 0 otherwise), matrix O where 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. U = $\mathsf{VR}_{\mathsf{tnt},c_i} \times \mathsf{VR}_{\mathsf{tnt},c_j}$ and R = $\mathsf{SCOPE}_{att_p} \cup \mathsf{SCOPE}_{att_q}$. Without loss of generality, we assume the values in $\mathsf{SCOPE}_{att_p}$ and $\mathsf{SCOPE}_{att_q}$ are disjoint.

2. M is a $|U| \times |R|$ dimensional boolean matrix where, for each u∈U and for each r ∈ R, M[u][r]=1 where (vr1,vr2)=u and $att_p$(vr1)=r or $att_q$(vr2)=r. Also, 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 $R_i \subseteq$ R in a set F where the support of $R_i$ is greater than minsup.

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

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

**Creation of Mutual Exclusive Candiate Attribute Relation:** Mutual Exclusive Candiate Attribute Relation is $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ where each element in $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is {val$_x$,val$_y$} such that val$_x \in$ R$_i$ and val$_y \in \overline{\text{R}_j}$ for each R$_i \rightarrow \overline{\text{R}}_j \in$ Rules.

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 candidate attribute relation should require much simpler approach since it only needs to identify relations between every two values of two different attributes of the virtual resources.

**B. Anti-Apriori for candidate attribute relation (CVRM-Apriori):**

We customize the anti-Apriori algorithm for mining mutual exclusive candidate attribute relations. We specify that {val$_x$,val$_y$} can be a member of $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ where val$_x \in \mathsf{SCOPE}_{att_p}$ and val$_y \in \mathsf{SCOPE}_{att_q}$ for each $att_p \in \mathsf{ATTR}^{\mathsf{c}_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{\mathsf{c}_j}_{\mathsf{tnt}}$, if it satisfies certain user-specified minsup and minconf for an already specified $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ of given $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_i}$ and $\mathsf{VR}_{\mathsf{tnt},\mathsf{c}_j}$. The support and confidence is calculated as follows,

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

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

$support^{att_p}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_x) = \frac{|insideR^{att_p}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_x)|}{|\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}|}$,

that calculates the ratio of the number of tuples in $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ that contain $\text{val}_x$ of the attribute $att_p \in \mathsf{ATTR}^{\mathtt{c}_i}_{\mathsf{tnt}}$ with all tuples in $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$.

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

- Finally, a function called $confidence^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ calculates the confidence which is the ratio of the number of elements in $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ that have a value $\text{val}_x$ of an attribute $att_p \in \mathsf{ATTR}^{\mathtt{c}_i}_{\mathsf{tnt}}$, but, simultaneously, do not have a value $\text{val}_y$ of an attribute $att_q \in \mathsf{ATTR}^{\mathtt{c}_j}_{\mathsf{tnt}}$ with the total number of elements in $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ that have a value $\text{val}_x$ of an attribute $att_p \in \mathsf{ATTR}^{\mathtt{c}_i}_{\mathsf{tnt}}$. Formally,

  $confidence^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_x,\text{val}_y) =$
  $\frac{|insideR^{att_p}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_x) \cap outsideR^{att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_y)|}{|insideR^{att_p}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}(\text{val}_x)|}$

Now, for a given $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$, for each $att_p \in \mathsf{ATTR}^{\mathtt{c}_i}_{\mathsf{tnt}}$ and for each $att_q \in \mathsf{ATTR}^{\mathtt{c}_j}_{\mathsf{tnt}}$, user specified $min\_sup^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ and $min\_conf^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$, algorithm 5.1 constructs the min_rules. In algorithm 5.1, procedure Identify_Frequency identifies each attribute value $\text{val}_x \in \mathsf{SCOPE}_{att_p}$ and each attribute value $\text{val}_y \in \mathsf{SCOPE}_{att_q}$ whose supports satisfy $min\_sup^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$ and returns them in sets F and $\overline{\text{F}}$ respectively. Now, the Gen_Candidate procedure takes the sets F and $\overline{\text{F}}$ and for each $\text{val}_x \in$ F and for each $\text{val}_y \in \overline{\text{F}}$ adds $\{\text{val}_x,\text{val}_y\}$ to $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ if it satisfies the value of $min\_conf^{att_p,att_q}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$. 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 $\overline{\text{F}}$ are specified separately from the scopes of two different attributes.

Now, for each $\{\text{val}_x, \text{val}_y\} \in \mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ we can construct a min_rule (formatted as $att_p$(v1)=$\text{val}_x \rightarrow att_q$(v2)$\neq\text{val}_y$) where $\text{val}_x \in \mathsf{SCOPE}_{att_p}$ and $\text{val}_y \in \mathsf{SCOPE}_{att_q}$.

**Algorithm 5.1** CVRM-Apriori

1: **procedure** Identify_Frequency($\mathsf{SCOPE}_{att_p}$,$\mathsf{SCOPE}_{att_q}$, $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$)
2: F = {}, $\overline{\text{F}}$={}
3: **for all** val $\in$ $\mathsf{SCOPE}_{att_p}$ **do**
4: **if** $support^{att_p}_{tnt,c_i,c_j}$(val) $\geq$ $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ **then**
5: Insert val into F
6: **end if**
7: **end for**
8: **for all** val $\in$ $\mathsf{SCOPE}_{att_q}$ **do**
9: **if** $support^{att_q}_{tnt,c_i,c_j}$(val) $\geq$ $min\_sup^{att_p,att_q}_{tnt,c_i,c_j}$ **then**
10: Insert val into $\overline{\text{F}}$
11: **end if**
12: **end for**
13: **end procedure**
14: **procedure** Gen_Candidate(F, $\overline{\text{F}}$, $min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$)
15: **for all** $\text{val}_x$ $\in$ F and $\text{val}_y$ $\in$ $\overline{\text{F}}$ **do**
16: **if** $confidence^{att_p,att_q}_{tnt,c_i,c_j}$($\text{val}_x$,$\text{val}_y$) $\geq$ $min\_conf^{att_p,att_q}_{tnt,c_i,c_j}$ **then**
17: $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ = $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ $\cup$ {$\text{val}_x$, $\text{val}_y$}
18: **end if**
19: **end for**
20: Return Flag
21: **end procedure**

### 5.5.3 Meaningful Attribute Relations

Each invocation to algorithm 5.1 may add new attribute relations to $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ based on the current configuration-logs, and, then min_rules are constructed accordingly. However, in the future, those min_rules may not be useful for various reasons in cloud IaaS including varying configuration requirements of the tenants. We characterize the candidate attribute relations into three different types based on some semantic meaning in context of cloud IaaS systems, and then construct min_rules only from those meaningful relations.

**Definition 1.** ***Strong Meaningful Relation:*** *A relation* $\{val_x, val_y\} \in CAR_{SCOPE_{att_p},SCOPE_{att_q}}$ *is a strong meaningful relation if there exists* $vr2 \in VR_{tnt,c_i}$ *and* $vr1 \in VR_{tnt,c_j}$ *such that* $att_p(vr1) = val_x$ *and* $att_q(vr2) = val_y$ *where* $att_p \in ATTR_{tnt}^{c_i}$ *and* $att_q \in ATTR_{tnt}^{c_j}$.

**Definition 2.** ***Weak Meaningful Relation:*** *A relation* $\{val_x,val_y\} \in CAR_{SCOPE_{att_p},SCOPE_{att_q}}$ *is a weak meaningful relation if there exist* $vr1 \in VR_{tnt,c_i}$ *and* $vr2 \in VR_{tnt,c_j}$ *such that* $att_p(vr1)=val_x$ *or* $att_q(vr2)=val_y$ *(but not both) where* $att_p \in ATTR_{tnt}^{c_i}$ *and* $att_q \in ATTR_{tnt}^{c_j}$.

**Definition 3.** ***Non-Meaningful Relation:*** *A relation* $\{val_x,val_y\} \in CAR_{SCOPE_{att_p},SCOPE_{att_q}}$ *is a non-meaningful relation if there exist* $vr1 \in VR_{tnt,c_i}$ *and* $vr2 \in VR_{tnt,c_j}$ *such that* $(att_p(vr1) \neq val_x \wedge val_x \notin SCOPE_{att_p})$ *or* $(val_y \notin SCOPE_{att_q} \wedge att_q(vr2) \neq val_y)$ *where* $att_p \in ATTR_{tnt}^{c_i}$ *and* $att_q \in ATTR_{tnt}^{c_j}$.

Definition 1 identifies the relations from which generated min_rules are useful since the system contains instances of both classes of virtual resources with respective attribute values assigned to them and those instances of virtual resources might be requested soon in future to put into the respective $\mathcal{R}_{\mathtt{tnt},\mathtt{c}_i,\mathtt{c}_j}$. However, definition 2 finds relatively weak ones since the system only has instances from one class of virtual resources assigned with the respective attribute-values. Absence of the other class instances make this relation weak in the sense that generated min_rule from this relation will not apply to any mapping configurations. However, there is a possibility that, in the future, the system will have instances from other virtual resource classes with the respective

**Algorithm 5.2** Meaningful Attribute Relation and min_rule

1: **procedure** min_rule_Construction($\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$,$\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$)
2: $TempR$=$\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}} \cup \mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$
3: $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$=Find_Meaningful($TempR$)
4: TempR=Rem_Non_Meaningful(TempR)
5: $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}} = TempR \setminus \mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$
6: **for all** {val$_x$,val$_y$} $\in$ $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$
7: Create_Min_Rule($\mathsf{min\_rule}_i, val_x, val_y$)
8: **end for**
9: **end procedure**

attribute values and, then, this relation will be useful. Finally, definition 3 identifies relations between attribute values that will have no use in the future. For certain reasons, system or the tenant might delete these values from the respective attribute scopes and none of the current virtual resource is assigned to these values. Therefore, min_rule generated from this relation will not apply to any future configuration request.

By utilizing all these definitions, we develop algorithm 5.2 to generate min_rules that have apparent application to the most current set of virtual resources. Here, a meaningful attribute relation called $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is created with strong meaningful relations from $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$. Each time procedure min_rule_Construction is called, previous min_rules are replaced by new min_rules. The procedure takes as input the candidate relations $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ which is generated by algorithm 5.1 and the previously created $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$. In figure 5.10, smart-feed is the previously created elements in $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ which will be considered for new meaningful relations creation. First, the procedure combines the elements in $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ and $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ to new relations called $TempR$. After that, it calls procedure Find_Meaningful that takes $TempR$ as input and finds the set of meaningful relations using definition 1 and creates a new $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ with these new relations. Then, the algorithm removes all the non-meaningful relations from $TempR$ (according to definition 3) by calling Rem_Non_Meaningful. Now, a new

$\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ is created by removing all the common elements $TempR$ and $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$. Note that, $\mathsf{CAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ now contains all the weak relations according to the definition 2. The goal is to keep these weak relations in back burner so that they might get chance to promote themselves to strong relations in future. Finally, the procedure constructs min_rules from $\mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ by calling procedure Create_Min_Rule. For each {val$_x$, val$_y$} $\in \mathsf{MAR}_{\mathsf{SCOPE}_{att_p},\mathsf{SCOPE}_{att_q}}$ Create_Min_Rule constructs a min_rule of the format $att_p$(v1)=val$_x$ $\rightarrow$ $att_q$(v2)$\neq$val$_y$ where val$_x$ $\in$ $\mathsf{SCOPE}_{att_p}$ and val$_y$ $\in$ $\mathsf{SCOPE}_{att_q}$.

Note that, each time this algorithm is called all the previously generated min_rules are deleted and new set of min_rules are created which are useful for the present sets of respective virtual resources in the system.

### 5.5.4 Implementation and Analysis

The evaluation of algorithm 5.2 is trivial since its only performs some set operations. The performance of algorithm 5.1 dominates the required time for overall mining process. We compare the performance of anit-Apriori and CVRM-Apriori algorithms. We implemented and evaluated both the mining algorithms 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 5.11 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

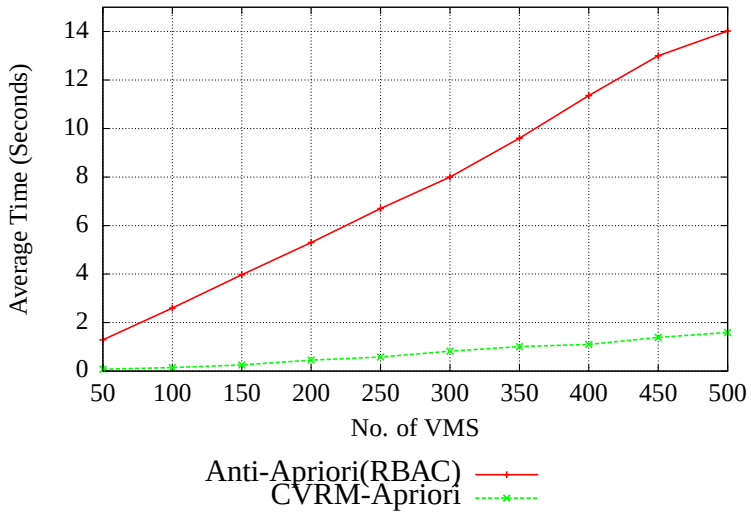


**Figure 5.11**: Mining Time with Increasing No. of VMs

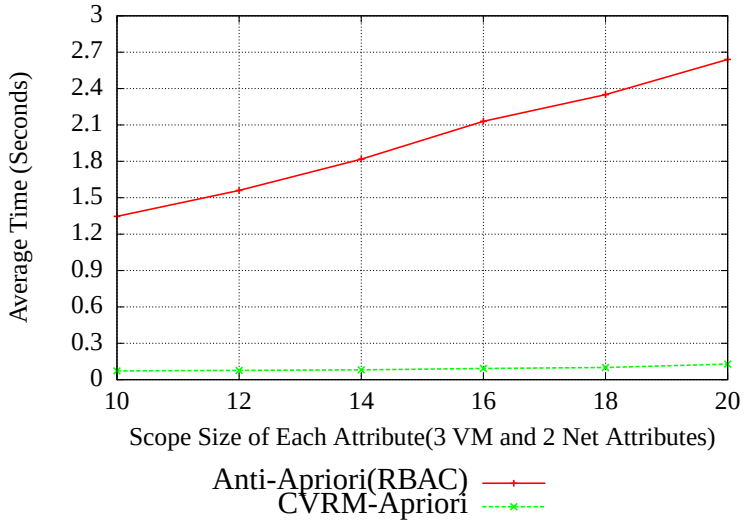


**Figure 5.12**: Mining Time with Increasing Scopes

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 5.12 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-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.

# Chapter 6: CONSTRAINT-AWARE VIRTUAL RESOURCE SCHEDULING

*The materials in this chapter are published in following venues [25]*:

1. Khalid Bijon, Ram Krishnan and Ravi Sandhu, Mitigating Multi-Tenancy Risks in IaaS Cloud Through Constraints-Driven Virtual Resource Scheduling, In Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT), June 1-3, 2015, Vienna, Austria.

In this chapter, we present our developed constraint aware virtual resource scheduling process where tenants can specify their constraints while scheduling their virtual resources in cloud.

## 6.1 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 this chapter 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.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. De-

veloping administration models for such attributes assignment is beyond the scope of this research. 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.

### 6.1.1 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 simply 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 $\in$ HOST has a capacity, represented as a function called *hW*, 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 the capacity of a Host remains constant unless explicitly modified, e.g., increasing RAM size.

[1] An example of an atomic attribute, given in section 1, is *sensitivity* where the values are high, medium and low. A VM can only get one of the three values for *sensitivity*. However, there might be cases that require set-valued attributes such as a *group* attribute of a VM which may take multiple values. For simplicity we only consider atomic attributes. However, the model can be easily extended to set-valued attributes.

[2] Multi-dimensional weights of a Host, e.g., RAM, CPU, etc., can be reduced to one single normalized weight. For instance, in OpenStack [6], 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 such as RAM, host's workload, etc.

- Similar to the capacity of Host, each VM $\in$ VM has a capacity represented by a function called *vW* where, *vW* : VM $\rightarrow$ 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 VM and the codomain is the values of *att* written as $\mathsf{SCOPE}_{att}$ 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 $\mathsf{SCOPE}_{att}$ of an *att* $\in$ $\mathsf{ATTR}_{\mathsf{VM}}$ that conflict with each other are specified as a relation called $\mathsf{ConSet}_{att}$. $\mathsf{ConSet}_{att}$ is irreflexive and symmetric, but not transitive. Hence, each element in $\mathsf{ConSet}_{att}$ is an unordered pair. For each *att* $\in$ $\mathsf{ATTR}_{\mathsf{VM}}$, $\mathsf{ConSet}_{att}$ is defined as follows.

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

Note that, for each *att* $\in$ $\mathsf{ATTR}_{\mathsf{VM}}$, a $\mathsf{ConSet}_{att}$ is similar to the conflict-set ***Attribute_Set*** of ABCL (defined in table 3.3) that captures conflicts among the values of a single attribute.

Part I in figure 6.1 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, {{$tnt_1$, $tnt_2$}, {$tnt_2$,$tnt_3$}, {$tnt_4$, $tnt_6$}} in $\mathsf{ConSet}_{tnt}$ 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 attributes assignment for VMs. For instance, for the VM 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, *vW*(vm1)=0.6.

### 6.1.2 Conflict-Free Host to VM Allocation

Given that the $\mathsf{ConSet}_{att}$ 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

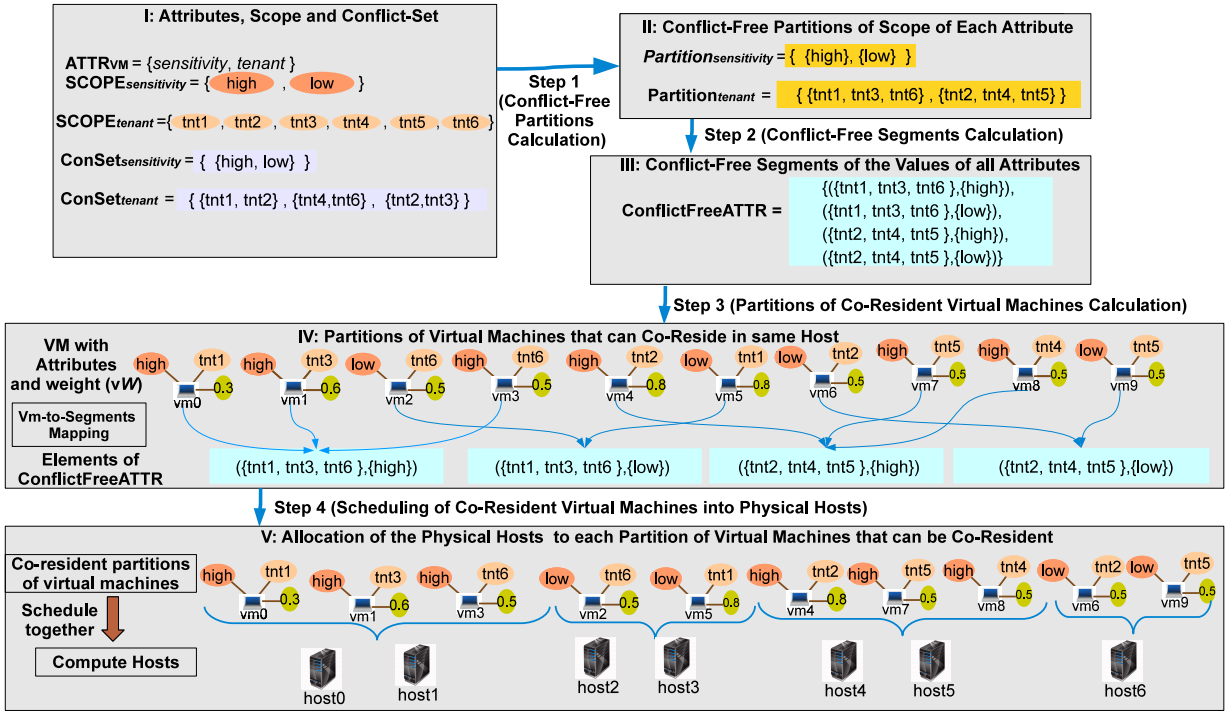


**Figure 6.1**: Conflict-Free VM-Host Allocation

conflict with each other. There are 4 steps in this process as illustrated in figure 6.1. Step 1 is to *partition* the values of each attribute (i.e., $\mathsf{SCOPE}_{att}$ 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 4. (Conflict-Free Partition of Attribute-Values)** *A* conflict-free partition of attribute-values *of each* $att \in \mathit{ATTR}_{VM}$ *is specified as* $\mathit{PARTITION}_{att}$ *that partitions the values in* $\mathit{SCOPE}_{att}$ *where the values of each element in* $\mathit{PARTITION}_{att}$ *do not conflict with each other, i.e., for each* $x \in \mathit{PARTITION}_{att}$ *and for each* $y \in \mathit{ConSet}_{att}$, $|x \cap y| \leq 1$.

We can state that, for an attribute *att*, a $\mathsf{PARTITION}_{att}$ partitions $\mathsf{SCOPE}_{att}$ 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 $\mathsf{SCOPE}_{att}$, 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 $\mathsf{ConSet}_{att}$.

Part II in figure 6.1 shows examples of conflict-free partitions, $\mathsf{Partition}_{tenant}$ and $\mathsf{Partition}_{sensitivity}$, for $\mathsf{ConSet}_{tenant}$ 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 $\mathsf{ConSet}_{att}$ of an attribute $att \in \mathsf{ATTR_{VM}}$. Section 6.2 shows that the selection of an appropriate $\mathsf{PARTITION}_{att}$ is important for 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 $att \in \mathsf{ATTR_{VM}}$. 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 $att \in \mathsf{ATTR_{VM}}$ generates all possible segments of conflict-free values of the attributes.

**Definition 5. (Conflict-Free Segments of the Values of Attributes)** *The* conflict-free segments of the values of attributes *is a set, called* $\mathsf{ConflictFreeATTR}$*, of* n*-tuples where* $n = |\mathsf{ATTR_{VM}}|$ *and each tuple is a result of the cartesian product of* $\mathsf{PARTITION}_{att}$ *of all* $att \in \mathsf{ATTR_{VM}}$*, i.e.,*

$$\mathsf{ConflictFreeATTR} = \prod_{att \in \mathsf{ATTR_{VM}}} \mathsf{PARTITION}_{att}$$

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

Part III in figure 6.1 shows an example $\mathsf{ConflictFreeATTR}$ which is produced from the Cartesian product of conflict-free partitions $\mathsf{Partition}_{tenant}$ and $\mathsf{Partition}_{sensitivity}$. A tuple ({$\text{tnt}_1$, $\text{tnt}_3$, $\text{tnt}_6$},{high}) is an element in $\mathsf{ConflictFreeATTR}$ since {$\text{tnt}_1$, $\text{tnt}_3$, $\text{tnt}_6$} and {high} are members of $\mathsf{Partition}_{tenant}$ and $\mathsf{Partition}_{sensitivity}$ respectively.

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

**Definition 6. (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* $\in$ $ATTR_{VM}$ *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 6.1 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 (*hW*) 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 7. (Conflict-Free Host to VM Allocation)** *Given VM, HOST,* $ATTR_{VM}$, *CoResidentVMGrp, hW and vW, 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* $\in$ *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,*

$$lvm \subseteq VM \wedge \bigvee_{x \in CoResidentVMGrp} lvm \subseteq x \wedge (\sum_{vm \in lvm} vW(vm)) \leq hW(cs)$$

Part V in figure 6.1 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.

### 6.1.3 Conflict-Free Scheduling of Other Virtual Resources to Physical Resources

The above described virtual machine to physical machine scheduling process (sections 6.1.1 and 6.1.2) can be easily applied to virtual storage to physical storage and virtual router to network host scheduling with the following modifications.

In physical storage to virtual storage allocation, two sets VM and HOST, defined in section 6.1.1, 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, $\mathsf{ConSet}_{att}$ 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. $\mathsf{ConSet}_{att}$ and definition 1-4 can be modified for network host to virtual router allocations.

## 6.2 Optimization Problem Definition and Solution Analysis

In this system, the specified conflicts restrict certain VMs from co-locating in the same Host. As a result, certain Hosts can no longer schedule VMs that conflict with currently scheduled VMs in those Hosts, despite having the required capacity. That increases the required number of Hosts a compared to 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 8. (Host Optimization Problem)** *The* Host optimization problem *seeks to minimize the number of Hosts in the mapping, allocate :* $\mathsf{HOST}' \hookrightarrow \mathcal{P}(VM)$*, specified in* Conflict-Free Host to VM Allocation (Definition 7).

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

### 6.2.1 MIN_PARTITION: Minimum Conflict-Free Partitions of Attribute-Values

More than one $\mathsf{PARTITION}_{att}$ can be generated for a given $\mathsf{ConSet}_{att}$. In figure 4, for the given $\mathsf{ConSet}_{tenant}$, 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 conflict-free attribute-values of *att*. Number of elements in $\mathsf{PARTITION}_{att}$ affects the total number of conflict-free segments (definition 5) 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} $\in$ 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 K_PARTITION problem states that given $\mathsf{SCOPE}_{att}$ and $\mathsf{ConSet}_{att}$ of an *att* $\in \mathsf{ATTR}_{\mathsf{VM}}$, and a positive integer k $\leq$ $|\mathsf{SCOPE}_{att}|$, can the values in $\mathsf{SCOPE}_{att}$ 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 $\mathsf{SCOPE}_{att}$, $\mathsf{ConSet}_{att}$, and k to K_PARTITION. The complexity of this conversion is |V| $\times$ |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 $\in$ V, color(u) $\in$ C and for any u, v $\in$ V, color(u)=color(v) only if {u, v} $\notin$ E. Also, for all u $\in$ $\mathsf{SCOPE}_{att}$, u belongs to cfs $\in$ CFS where #CFS is k, and for any u, v $\in$ $\mathsf{SCOPE}_{att}$, u, v belongs to the same cfs $\in$ CFS, if {u, v} $\notin$ $\mathsf{ConSet}_{att}$. Thus, G is an *yes instance* of K_PARTITION.

$\Longleftarrow$ Assume $\mathsf{SCOPE}_{att}$, $\mathsf{ConSet}_{att}$ is an *yes* instance of K_PARTITION and there exists a family of CFS of size k. Thus, for all u $\in$ $\mathsf{SCOPE}_{att}$, u belongs to a cfs $\in$ CFS, and for any u, v $\in$ $\mathsf{SCOPE}_{att}$, u, v belongs to the same cfs $\in$ CFS, if {u, v} $\notin$ $\mathsf{ConSet}_{att}$. Thus, the vertices in same cfs $\in$ 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

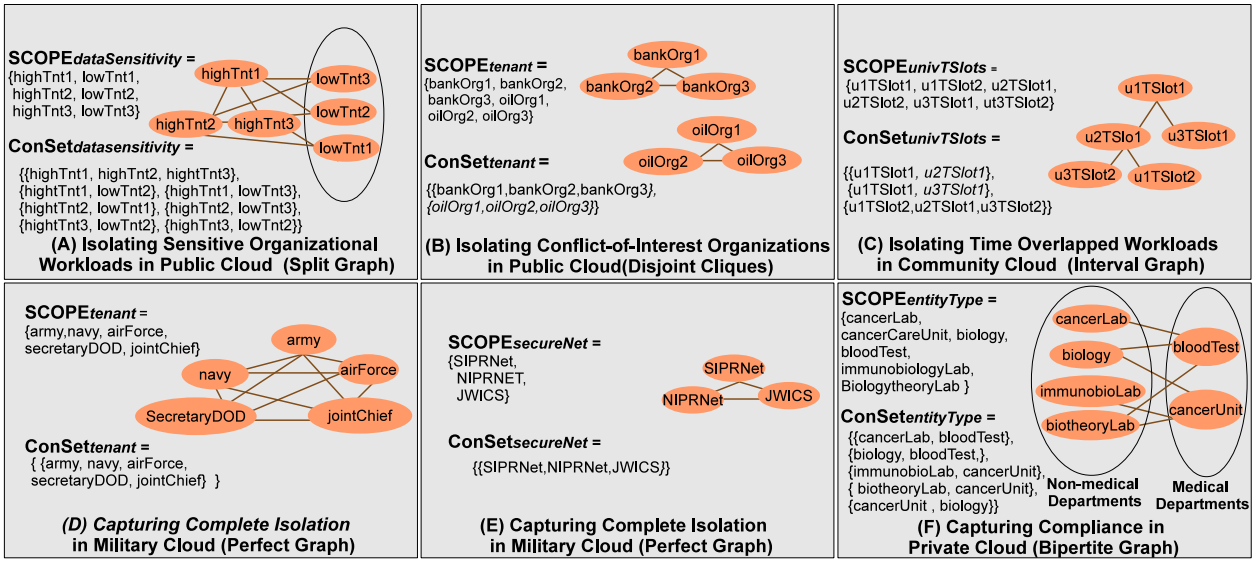


**Figure 6.2**: Conflicts of different Systems and Corresponding Conflict Graphs

approximate in the sense that they may not provide the minimum size of PARTITION$_{att}$, i.e., MIN_PARTITION may not be optimal. This is useful, although not optimal, because the conflicts are still satisfied. In the following, we discuss approximate algorithms for graph-coloring and their applications to MIN_PARTITION.

### Restricted Conflict Graphs

Certain graphs such as perfect graphs have polynomial graph-coloring solutions. We identify that certain restricted versions of attribute conflict specification generates such graphs. We explore restricted graphs having polynomial-time solutions and demonstrate their usage scenarios for private, public, and community cloud deployment scenarios.

### 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.

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 6.2-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 [62].

**2. Conflict-of-Interest:** 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 [62]. Figure 6.2-B shows such conflict-of-interest use cases.

### 6.2.2 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 6.2-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 [62].

### 6.2.3 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, e.g., the private cloud operated by Amazon for the CIA [9].

**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 *militaryOrg* 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 6.2-D which can be solved in polynomial-time [62].

Figure 6.2-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:** Compliance is another major concern in a private cloud. 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 6.2-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, thus, solved in polynomial-time [62].

We also develop an exact algorithm, shown in algorithm 6.1, based on backtracking. The complexity of this algorithm is NP since it is an adaptation of the general backtracking algorithm for the graph-coloring [16]. However, for attributes whose size of the scope is small enough (e.g. *sensitivity*), the algorithm computes the partition relatively fast. In algorithm 6.1, the Make_Partition procedure is called with scope $\mathsf{SCOPE}_{att}$ of an attribute *att* $\in \mathsf{ATTR}_{\mathsf{VM}}$, $\mathsf{ConSet}_{att}$, and a partition $\mathsf{PARTITION}_{att}^{k}$ that can contain $k$ elements. Make_Partition is 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 $\mathsf{ConSet}_{att}$. 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 $\mathsf{ConSet}_{att}$. In section 6.3, we analyze the performance of this algorithm for various sizes of attribute scopes and conflict sets.

### 6.2.4 ConflictFreeATTR Generation

This is a trivial algorithm that calculates the values of ConflictFreeATTR specified in definition 5. 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 $\mathsf{PARTITION}_{att}$.

### 6.2.5 Co-Resident VM Partitions Generation

This algorithm takes ConflictFreeATTR and VM sets as input, creates a family of sets, called CoResidentVMGrp (definition 6), 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 CoResidentVM-

**Algorithm 6.1** Conflict-Free Partition using Backtracking

```
1: procedure Check_Validity(attval, ConSet_att, CSet)
2:     for all attval_i ∈ CSet do
3:      if {attval, attval_i} ∈ ConSet_att    then
4:         Return False
5:       end if
7:     end for
8:     Return True
9:  end procedure
10: procedure Make_Partition(SCOPE_att, ConSet_att, PARTITION^k_att)
11:    if attval ∈ SCOPE_att then
12:       for all par ∈ PARTITION^k_att do
13:         if Check_Validity(attval, ConSet_att, par) then
14:             par = par ∪ attval
15:             if Make_Partition(SCOPE_att-{par},ConSet_att,PARTITION^k_att) then
16:                 Return True
17:             end if
18:             par = par − attval
19:         end if
20:       end for
21:    end if
22:    Return False
23:  end procedure
```

Grp 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 $O(\text{VM} \times \text{ConflictFreeATTR} \times \text{ATTR}_{\text{VM}})$.

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, based on assigned values to attributes, to an element in ConflictFreeATTR and adds the VM to a corresponding element in CoResidentVMGrp.

### 6.2.6 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 [50] problem which is NP-Hard. However, there are a number of known heuristic approaches that can be applied here [87]. Note that the scheduling of VMs to `hosts` in an optimal way based on capacity is orthogonal to MIN_PARTITION since MIN_PARTITION is solved before this scheduling begins.

## 6.3 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 [5], a quick installation of

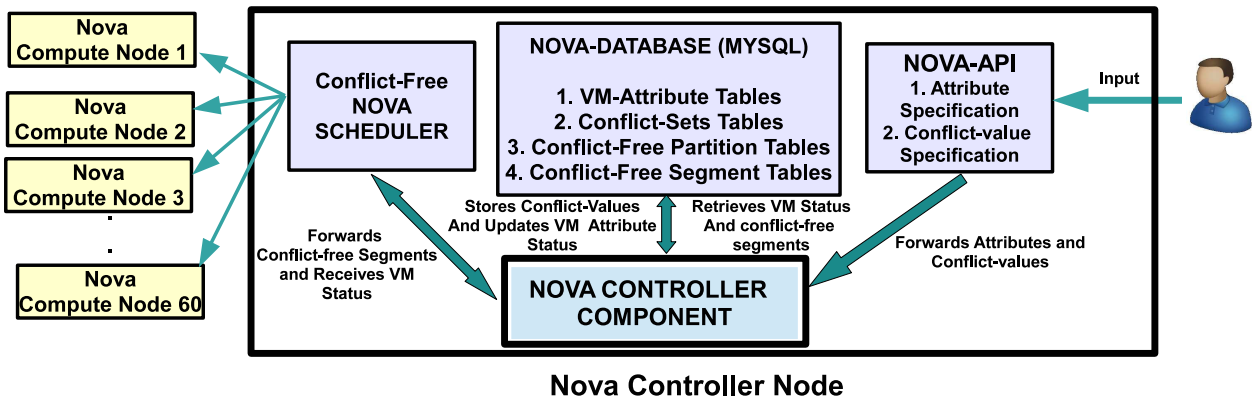


**Figure 6.3**: Experimental Setup in OpenStack

OpenStack ideal for experimentation, using those VMs as physical hosts to create a virtual cloud for the purpose of experimentation. Basically, we use the Havana release of OpenStack [8]. 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 6.3 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. 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 6.1). Our python based conflict free segments calculation process (steps 1 and 2 in figure 6.1) has 153 lines of code. Finally, our implementation of conflict-free Host to VM scheduling (steps 3 and 4 in figure 6.1) 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 6.1.* This experiment analyzes the runtime of Al-

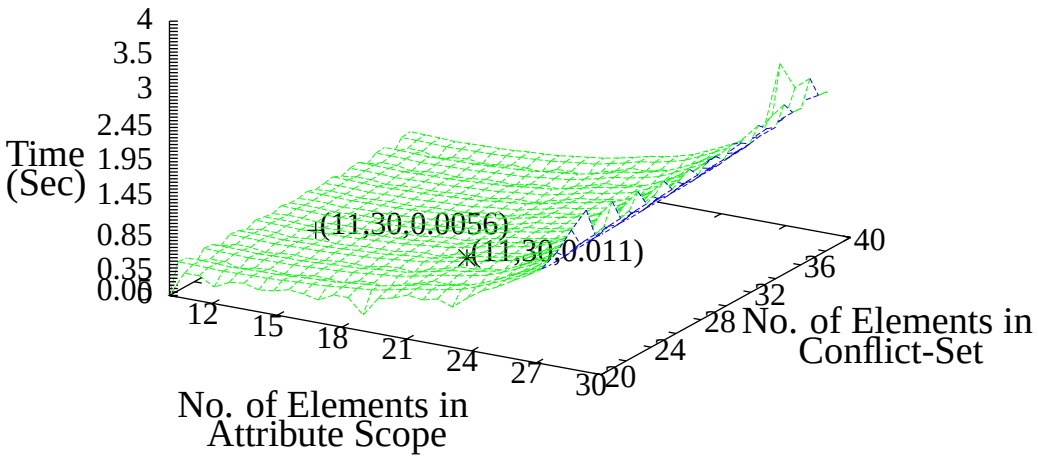


**Figure 6.4**: Required Time for Small Scope and Confilct-Set

gorithm 6.1. 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. Also, we repeat this process 50 times where conflicts are generated randomly. Figure 6.4 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 6.5 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

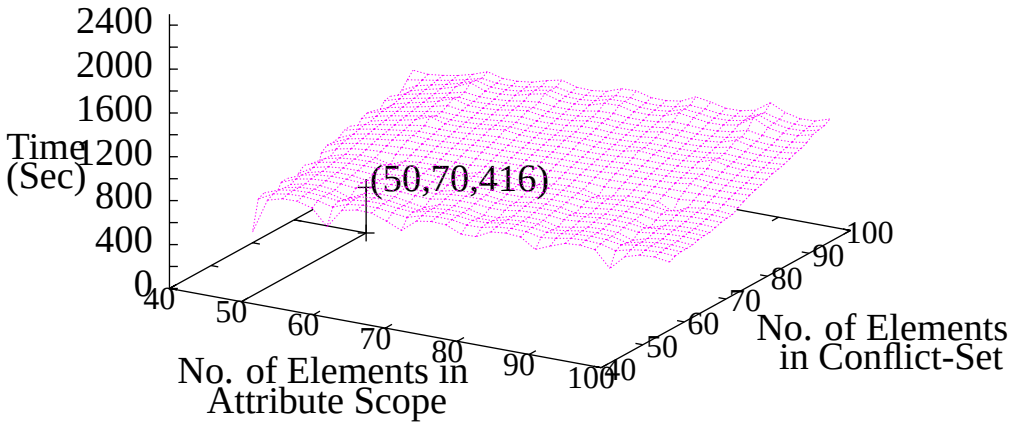


**Figure 6.5**: Required Time for Large Scope and Conflict-Set

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 6.6, 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 6.7, 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

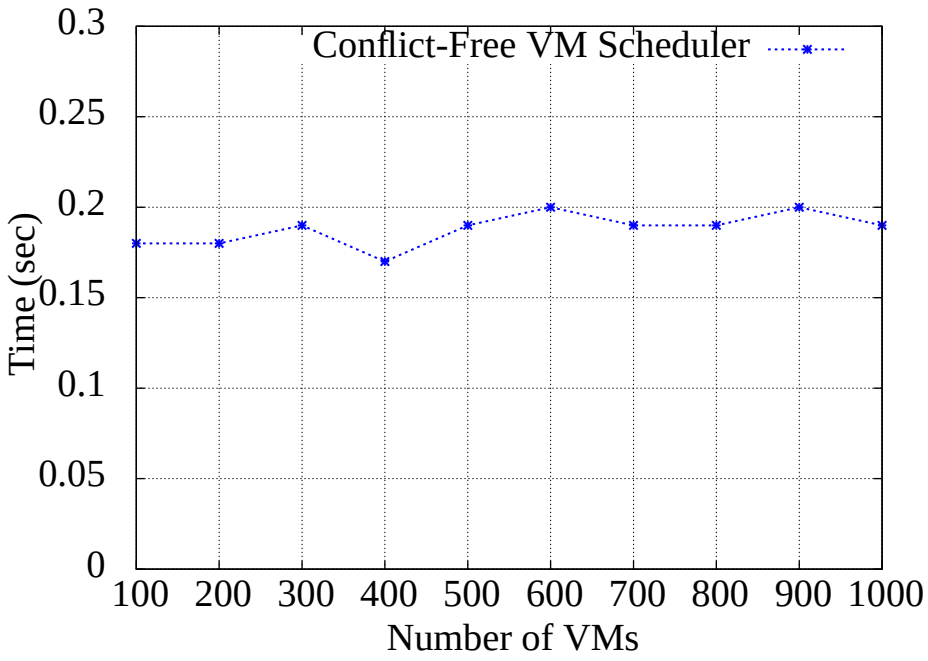


**Figure 6.6**: Latency for Conflict-free Scheduling

conflict set is referred to as the maximum degree of conflict for that conflict set. In figure 6.7, we fix the maximum degree to 2. In figure 6.8, 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 6.9, 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

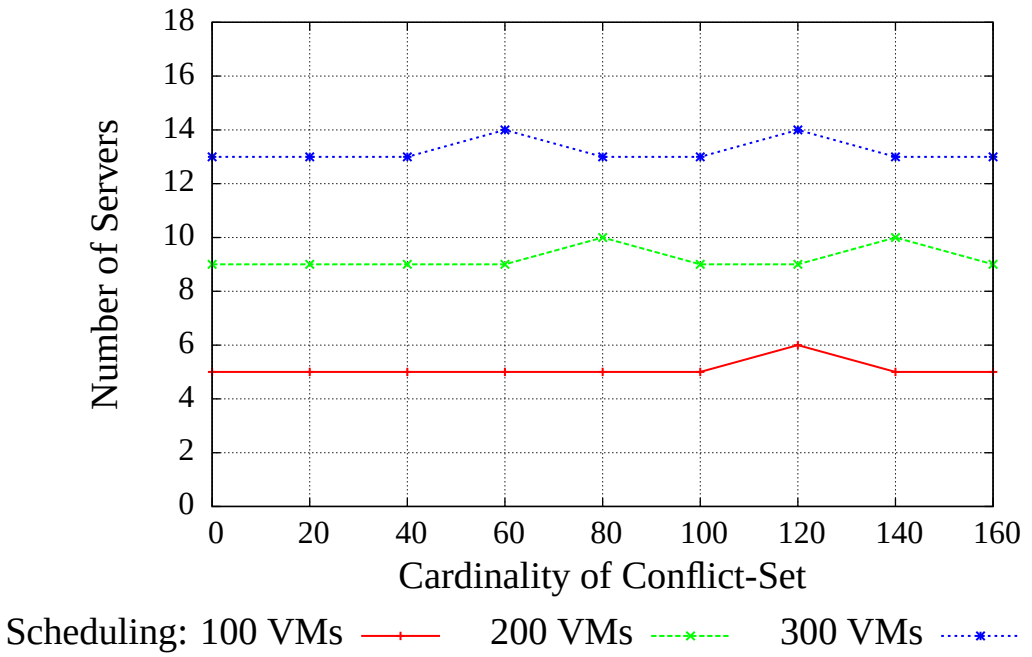


**Figure 6.7**: Required Number of Hosts for Varying Number of Elements in Conflict-Set

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.

**Experiment 5 -** *Limitation of the OpenStack Scheduler.*

In Nova, multiple groups can be created, called anti-affinity groups, where VMs in same group cannot co-locate and a user can manually put a VM to these groups. Then, the Nova scheduler called 'filter-scheduler' schedules the VMs while satisfying the anti-affinity groups. Unlike our scheduling, 'filter-scheduler' does not consider the host optimization problem. It selects a Host with highest available capacity for a requested VM if already scheduled VMs to that Host are not in same anti-affinity groups. Let's say, for example, 3 Hosts $h_1$, $h_2$, and $h_3$ have enough capacity to schedule 15 VMs ($vm_1$ to $vm_{15}$). There are 3 anti-affinity groups $af_1$, $af_2$ and $af_3$ where $vm_1$, $vm_4$, $vm_7$, $vm_{10}$, $vm_{13}$, are in $af_1$, $vm_2$, $vm_5$, $vm_8$, $vm_{11}$, $vm_{14}$ are in $af_2$ and $vm_3$, $vm_6$, $vm_9$, $vm_{12}$, $vm_{15}$ in $af_3$. If $vm_1$ to $vm_{15}$ are sequentially requested, after scheduling $vm_1$ to $vm_9$ no Host will be available. Scheduling of new VM now require migration of already scheduled VMs that incurs

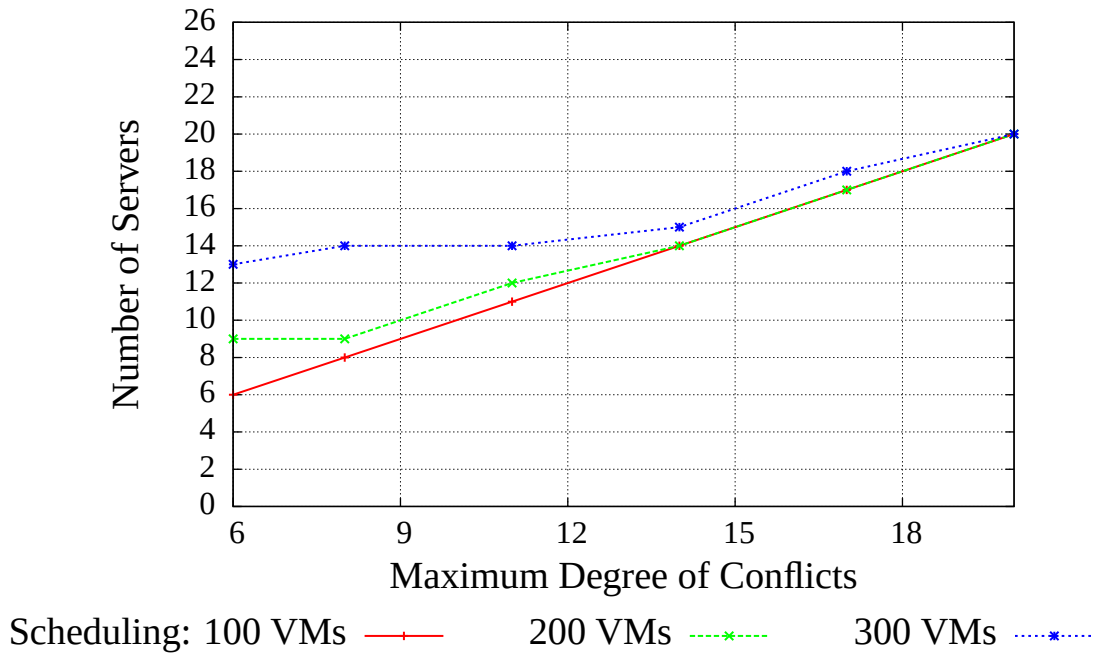


**Figure 6.8**: Required Number of Hosts for Max Degree of Conflicts

additional cost. However, our scheduling process can schedule all the 15 VMs since it optimizes Hosts.

## 6.4 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.

### 6.4.1 Types of Conflict Change

In general, a conflict-set changes if a new conflict is added or an existing conflict is removed. Given a $\mathsf{ConSet}_{att}$ and a $\mathsf{PARTITION}_{att}$ of an $att \in \mathsf{ATTR}_{\mathsf{VM}}$, $\mathsf{ConSet}_{att}$ can change to a new conflict

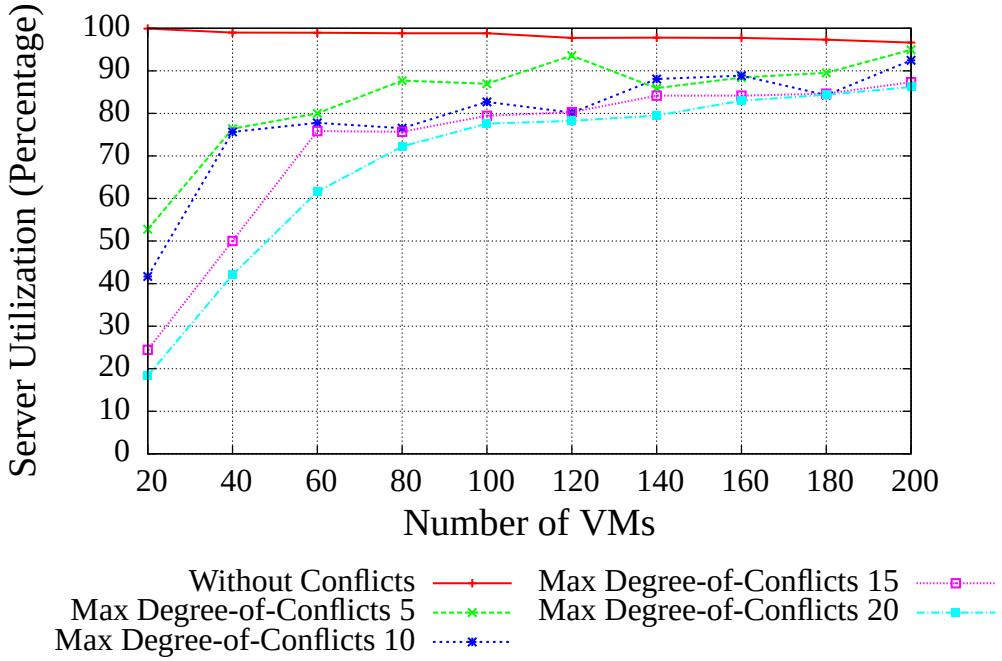


**Figure 6.9**: Host Utilization Overhead

set $\mathsf{ConSet}'_{att}$ ( a new partition $\mathsf{PARTITION}'_{att}$ can be calculated accordingly) in three different ways.

• $\Delta_1$—this type of change involves operations that only remove an element from $\mathsf{ConSet}_{att}$ where $|\mathsf{PARTITION}'_{att}| < |\mathsf{PARTITION}_{att}|$. Evidently, it does not add new conflicts, hence, the scheduled VMs need not migrate.

• $\Delta_2$—this type of change involves operations that add an element to $\mathsf{ConSet}_{att}$. 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.

• $\Delta_3$—this type of change adds an element to $\mathsf{ConSet}_{att}$ 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 $\mathsf{SCOPE}_{att}$ = {a1,a2,a3,a4,a5,a6}, where the initial conflict-set $\mathsf{ConSet}_{att}$ = {{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}}.

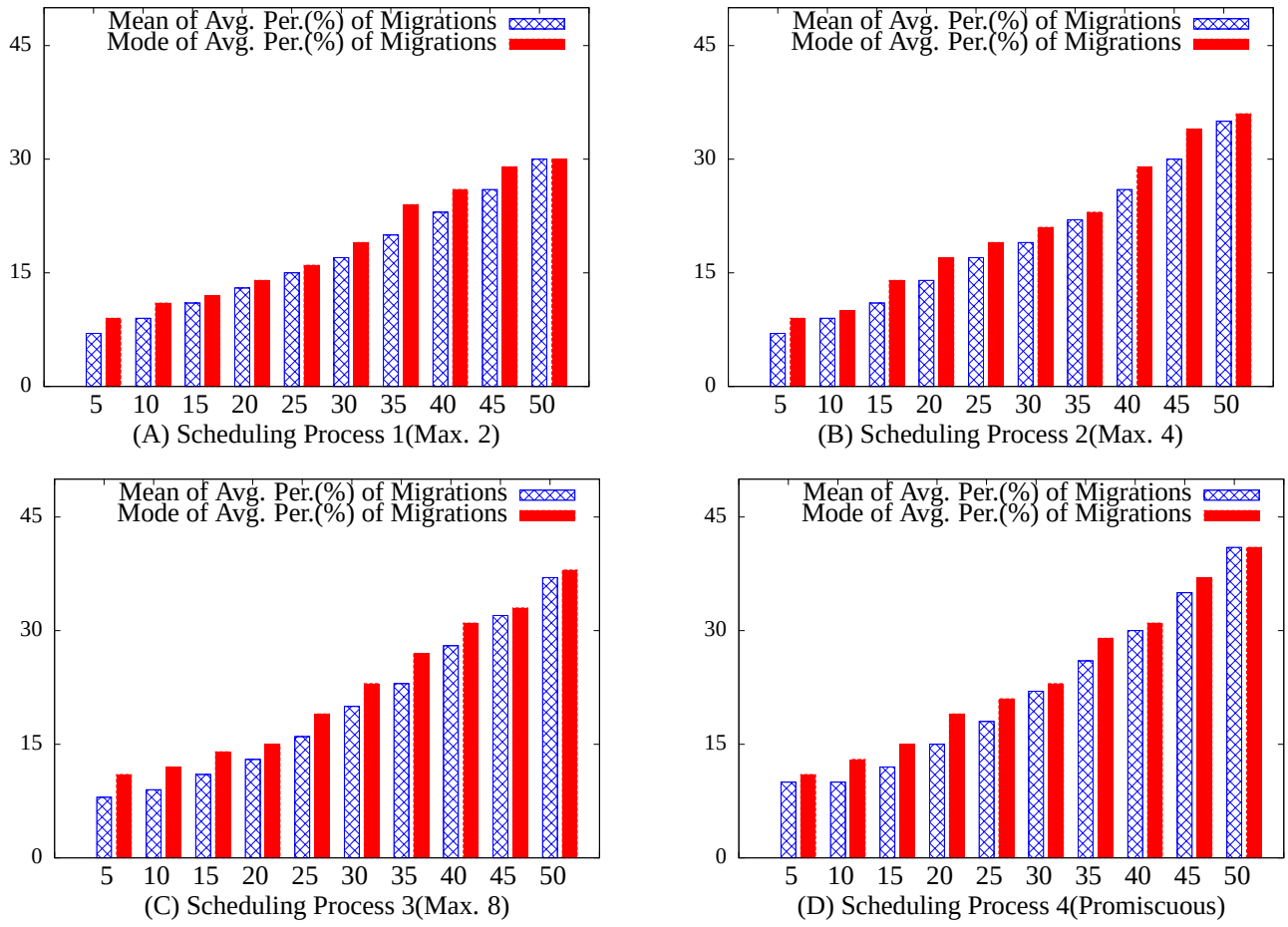


**Figure 6.10**: Cost Analysis: X-axis(% of the Total Conflicts for Given Scopes), Y-axis(% of Total VMs that Require Migrations)

**Consider a change of type** $\Delta_1$ that removes {a2,a4} from $\mathsf{ConSet}_{att}$ where resultant conflict set $\mathsf{ConSet}^1_{att}$={{ 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** $\Delta_2$ that adds {a2,a3} to $\mathsf{ConSet}_{att}$ where new conflict set $\mathsf{ConSet}^2_{att}$={{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** $\Delta_3$ that adds {a1,a6} to $\mathsf{ConSet}_{att}$ where $\mathsf{ConSet}^3_{att}$= {{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.

### 6.4.2 Cost Analysis

In this section, we analyze the cost of continuing to satisfy the conflicts as they change, when the change is of type $\Delta_3$. 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 $\Delta_3$-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: (step-1) We consider a single VM attribute called *att* where we vary the size of $\mathsf{SCOPE}_{att}$ from 10 to 35 with an increment of 5. (step-2) For each $\mathsf{SCOPE}_{att}$, initially, we randomly populate $\mathsf{ConSet}_{att}$

with 5 to 50 elements and calculate PARTITION$_{att}$. We repeatedly perform this step for 50 times for every step 1. (step-3) For each step-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 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 6.10 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 6.10(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 6.10(C), 50% conflicts cost 79% migrations, where 10 different 5% conflicts cost 10×9%=90% migrations.

### 6.4.3 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 [35]. 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 [35].

## 6.5 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 6.1, *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 6.1.1, 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 6.1 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 where a simple approach could be the classification of attribute-values based on some class, as shown in section 6.2.1 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 6.1, 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 6.1, the values of $\mathsf{ConSet}_{sensitivity}$ and $\mathsf{ConSet}_{tenant}$ 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 [77]. A potential future extension is to consider conflicts between Hosts and VMs for the scheduling decisions while optimizing the number of Hosts.

# Chapter 7: CONCLUSION

The following sections summarize contributions of this dissertation and discuss some future research directions that can be further studied.

## 7.1 Summary

In this research, first, we developed a constraints specification language for ABAC. We identified different type of relations among attributes and develop a hierarchical relationship structure. We verified the expressiveness of the language by configuring various separation-of-duty constraints for role based access control and security policies for banking organizations.

We also developed constraints specification mechanisms in cloud IaaS. First, we formalized a simple constraints specification mechanism for isolation management at both user level and resource level in the cloud. We also developed a formal administrative model for the management of user privileges. We then presented CVRM, the very first constraints 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 would be installed in every cloud service provided by the IaaS providers. We also identified that virtual-resource management policies can be discovered and constructed from log file which is similar to the well-known frequent-itemsets mining problem in database systems. We demonstrated a constraint mining algorithm for CVRM where the algorithm leverages standard Apriori algorithm from the data mining literature. We also analyzed whether the mined constraints preserve semantic meaning with respective to the configuration requirements of the tenant.

Finally, we presented a generalized attribute-based constraints 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. We developed and analyzed the implementation of this mechanism in OpenStack cloud platform.

## 7.2 Future Work

There are several opportunities for extending the work presented in this dissertation.

In CVRM, there is always option to improve the mining. For instance, in this dissertation, we consider that the log file is noise free. An obvious future work would be to introduce noise in this system and develop a more dynamic mining algorithm that can mitigate the effect of noise on mining results. Another future work, would be to develop a misconfiguration detection mechanism on top of the mining of CVRM.

Another potential future work is to extend our constraint-aware scheduling mechanism to address the limitations discussed in section 6.5. Also, a future direction is to develop a suitable front-end application program interface for specification and management of the conflicts. The vision behind this is to expose resource management capabilities to the tenants so they can retain control when moving to cloud.

This dissertation builds foundations for an attribute based constraint specification system. A potential future work is to investigate the usefulness of this system in other dynamic domains such as managing privileges of the android apps in smartphones.

## BIBLIOGRAPHY


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

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

[3] AWS security best practices. *http://media.amazonwebservices.com/AWS-Security-Best-Practices.pdf*.

[4] Clique. *http://en.wikipedia.org/wiki/Clique*.

[5] DevStack. *https://wiki.openstack.org/wiki/DevStack*.

[6] KeyStone. *http://docs.openstack.org/developer/keystone/*.

[7] OpenStack. *https://wiki.openstack.org/*.

[8] OpenStack Havana Release. *http://www.openstack.org/software/havana*.

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

[10] The internet engineering task force (IETF). In *http://www.ietf.org/*, 2013.

[11] Rakesh Agrawal, Tomasz Imieliński, and Arun 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.

[12] Gail Joon Ahn and Ravi Sandhu. Role-based authorization constraints specification. *ACM Trans. Inf. Syst. Secur.*, 3(4):207–226, November 2000.

[13] A. Almutairi, M. Sarfraz, S. Basalamah, W.G. Aref, and A. Ghafoor. A distributed access control architecture for cloud computing. *IEEE Software*, 29(2), 2012.

[14] Abdulrahman Almutairi, Muhammad Sarfraz, Saleh Basalamah, Walid Aref, and Arif Ghafoor. A distributed access control architecture for cloud computing. *Software*, 29(2), 2012.

[15] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Antony IT Rowstron. Towards predictable datacenter networks. In *Procedings of The ACM Special Interest Group on Data Communication*, pages 242–253, 2011.

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

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

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

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

[20] Khalid Bijon, Ram Krishman, and Ravi Sandhu. Constraints specication in attribute based access control. *ASE Science Journal*, 2(3):pp–131, 2013.

[21] Khalid Bijon, Ram Krishman, and Ravi Sandhu. Automated constraints specification for virtual resource orchestration in cloud IaaS. *Under Review in IEEE Transactions on Dependable and Secure Computing*, 2015.

[22] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Towards an attribute based constraints specfication language. In *PASSAT*. IEEE, 2013.

[23] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Towards an attribute based constraints specification language. In *Proc. of the PASSAT*, 2013.

[24] Khalid Bijon, Ram Krishnan, and Ravi 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.

[25] Khalid Bijon, Ram Krishnan, and Ravi Sandhu. Mitigating multi-tenancy risks in iaas cloud through constraints-driven virtual resource scheduling. In *Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT)*, SACMAT '15. ACM, 2015.

[26] Khalid Bijon, Ram Krishnan, and Ravi 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*, CODASPY '15, pages 183–194, New York, NY, USA, 2015. ACM.

[27] Khalid Bijon, Ravi Sandhu, and Ram Krishnan. A group-centric model for collaboration with expedient insiders in multilevel systems. In *International Symp. on Security in Collaboration Technologies and Systems*, 2012.

[28] Khalid Zaman Bijon, Tahmina Ahmed, Ravi Sandhu, and Ram Krishnan. A lattice interpretation of group-centric collaboration with expedient insiders. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 200–209. IEEE, 2012.

[29] Khalid Zaman Bijon, Ram Krishnan, and Ravi Sandhu. Risk-aware RBAC sessions. In *Information Systems Security*, pages 59–74. Springer, 2012.

[30] K.Z. Bijon, R. Krishnan, and R. Sandhu. A framework for risk-aware role based access control. In *IEEE Conference on Communications and Network Security (CNS)*, pages 462–469, Oct 2013.

[31] 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.

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

[33] Piero A Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security*, 10(3), 2002.

[34] Tyson R Browning and Ali A Yassine. Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 126(2):212–228, 2010.

[35] Daniel Bryce and Subbarao Kambhampati. A tutorial on planning graph based reachability heuristics. *AI Magazine*, 2007.

[36] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *HPCS'09*, pages 1–11. IEEE, 2009.

[37] Vincent C. Hu et al. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 2013.

[38] Nicolò Maria Calcavecchia, Ofer Biran, Erez Hadad, and Yosef Moatti. VM placement strategies for cloud scenarios. In *Procedings of The International Conference on Cloud Computing*. IEEE, 2012.

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

[40] Antonio Celesti, Francesco Tusa, Massimo Villari, and Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Procedings of The IEEE International Conference on Cloud Computing*, 2010.

[41] Melissa Chase. Multi-authority attribute based encryption. In *Theory of Cryptography*, pages 515–534. Springer, 2007.

[42] Fang Chen and Ravi S. Sandhu. Constraints for role-based access control. In *Proceedings of the First ACM Workshop on Role-based Access Control*, 1996.

[43] Ge Cheng et al. A prioritized chinese wall model for managing the covert information flows in virtual machine systems. In *IEEE ICYCS*, 2008.

[44] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *The International Conference on on Social Computing (SocialCom)*, pages 646–655. IEEE, 2012.

[45] Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *Data and applications security and privacy XXVI*, pages 8–24. Springer, 2012.

[46] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Preserving user privacy from third-party applications in online social networks. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 723–728. International World Wide Web Conferences Steering Committee, 2013.

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

[48] David D. Clark and David R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE S&P*, 1987.

[49] Bill Claybrook. Comparing cloud risks and virtualization risks for data center apps. *http://searchdatacenter.techtarget.com/tip/ 0,289483,sid80 gci1380652,00.html.*

[50] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[51] Jason Crampton. Specifying and enforcing constraints in role-based access control. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 43–50. ACM, 2003.

[52] Jason Crampton and Hemanth Khambhammettu. A framework for enforcing constrained rbac policies. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, volume 3, pages 195–200. IEEE, 2009.

[53] Ernesto Damiani, S De Capitani Di Vimercati, and Pierangela Samarati. New paradigms for access control in open environments. In *proceedings of the ISSPIT*, 2005.

[54] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. Infrastructure as a service security: Challenges and solutions. In *Procedings of International Conference on Informatics and Systems*, pages 1–8, 2010.

[55] Leah Epstein and Asaf Levin. On bin packing with conflicts. In *Approximation and Online Algorithms*, pages 160–173. Springer, 2007.

[56] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of the 11th ACSAC*, 1995.

[57] David F. Ferraiolo et al. Proposed NIST standard for role-based access control. *ACM Tran. Inf. Sys. Sec.*, 2001.

[58] Michael R Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[59] Michael R Garey and David S Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*, volume 174. Freeman New York, 1979.

[60] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers & Operations Research*, 31(3):347–358, 2004.

[61] Virgil D Gligor et al. On the formal definition of separation-of-duty policies and their composition. In *Proceedings of the IEEE Security and Privacy*, 1998.

[62] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.

[63] José Fernando Gonçalves, Jorge JM Mendes, and Mauricio GC Resende. A genetic algorithm for the resource constrained multi-project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190, 2008.

[64] Vipul Goyal et al. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of The ACM Conference on Computer and Communications Security*, 2006.

[65] Eric Grosse, John Howie, James Ransome, Jim Reavis, and Steve Schmidt. Cloud computing roundtable. In *Proc. of the IEEE S&P*, 2010.

[66] Ajay Gulati, Irfan Ahmad, Carl A Waldspurger, et al. Parda: Proportional allocation of resources for distributed storage access. In *FAST*, volume 9, pages 85–98, 2009.

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

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

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

[70] Sreekanth Iyer. Top 5 challenges to cloud computing. *Cloud Computing Central, https://www.ibm.com/ developerworks /community/blogs/c2028fdc-41fe- 4493-8257-33a59069fa04/entry/top-5- challenges-to-cloud-computing 4?lang=en*, 2011.

[71] Trent Jaeger. On the increasing importance of constraints. In *Proceedings of the ACM RBAC*, 1999.

[72] Trent Jaeger, Reiner Sailer, and Yogesh Sreenivasan. Managing the risk of covert information flows in virtual machine systems. In *Proceedings of The ACM Symposium on Access Control Models and Technologies*, 2007.

[73] Sushil Jajodia et al. Flexible support for multiple access control policies. *ACM TODS*, 26(2):214–260, 2001.

[74] Klaus Jansen. An approximation scheme for bin packing with conflicts. In *Algorithm Theory SWAT'98*, pages 35–46. Springer, 1998.

[75] Klaus Jansen. An approximation scheme for bin packing with conflicts. *Journal of combinatorial optimization*, 3(4):363–377, 1999.

[76] Amarnath Jasti, Payal Shah, Rajeev Nagaraj, and Ravi Pendse. Security in multi-tenancy cloud. In *Proceedings of the International Carnahan Conference on Security Technology (ICCST)*, pages 35–41. IEEE, 2010.

[77] Ravi Jhawar, Vincenzo Piuri, and Pierangela Samarati. Supporting security requirements for resource management in cloud computing. *IEEE CSE*, 0:170–177, 2012.

[78] Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *DBSec*, 2012.

[79] Xin Jin, Ram Krishnan, and Ravi Sandhu. A role-based administration model for attributes. In *Proceedings of the First International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12. ACM, 2012.

[80] David Karger, Rajeev Motwani, and Madhu Sudan. Approximate graph coloring by semidefinite programming. In *Proc. of 35th Annual Symposium on Foundations of Computer Science*, pages 2–13. IEEE, 1994.

[81] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 350–361, 2010.

[82] Ibrahim S Kurtulus and Subhash C Narula. Multi-project scheduling: Analysis of project performance. *IIE transactions*, 17(1):58–66, 1985.

[83] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.

[84] William Leinberger, George Karypis, and Vipin Kumar. Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints. In *Proc. of the International Conference on Parallel Processing*, pages 404–412. IEEE, 1999.

[85] Kangkang Li, Jie Wu, and A. Blaisse. Elasticity-aware virtual machine placement for cloud datacenters. In *IEEE 2nd International Conference on Cloud Networking*, pages 99–107, Nov 2013.

[86] Xiaopu Ma, Ruixuan Li, Zhengding Lu, and Wei Wang. Mining constraints in role-based access control. *Mathematical and Computer Modelling*, 55(1):87–96, 2012.

[87] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *IEEE Transactions on Cloud Computing*, 1(2):215–228, 2013.

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

[89] I Morihara, T Ibaraki, and T Hasegawa. Bin packing and multiprocessor scheduling problems with side constraint on job types. *Discrete applied mathematics*, 6(2):173–191, 1983.

[90] Michael J Nash and Keith R Poland. Some conundrums concerning separation of duty. In *Research in Security and Privacy*, pages 201–207. IEEE, 1990.

[91] Dang Nguyen, Jaehong Park, and Ravi Sandhu. Dependency path patterns as the foundation of access control in provenance-aware systems. In *Proceedings of the 4th USENIX conference on Theory and Practice of Provenance*, pages 4–4. USENIX Association, 2012.

[92] Dang Nguyen, Jaehong Park, and Ravi Sandhu. A provenance-based access control model for dynamic separation of duties. In *The Eleventh Annual International Conference on Privacy, Security and Trust (PST)*, pages 247–256. IEEE, 2013.

[93] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of The ACM Conference on Computer and Communications Security*, 2007.

[94] Jaehong Park, Dang Nguyen, and R. Sandhu. A provenance-based access control model. In *The Tenth Annual International Conference on Privacy, Security and Trust (PST)*, pages 137–144, July 2012.

[95] Jaehong Park and Ravi Sandhu. The UCON$_{ABC}$ usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1), 2004.

[96] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *Proceedings of The ACM Conference on Computer and Communications Security*, pages 99–112, 2006.

[97] P Ranjith, Chandran Priya, and Kaleeswaran Shalini. On covert channels between virtual machines. *Journal in Computer Virology*, 8(3):85–97, 2012.

[98] Thomas 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.

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

[100] Francisco Rocha and Miguel Correia. Lucy in the sky without diamonds: Stealing confidential data in the cloud. In *Proc. of the IEEE DSN-W*, 2011.

[101] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proceedings of the EUROCRYPT*. 2005.

[102] Ravi Sandhu. Transaction control expressions for separation of duties. In *Proceedings of the 4th ACSAC*, 1988.

[103] Ravi Sandhu, Khalid Zaman Bijon, Xin Jin, and Ram Krishnan. Rt-based administrative models for community cyber security information sharing. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 473–478. IEEE, 2011.

[104] Ravi Sandhu, David Ferraiolo, and Richard Kuhn. The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, pages 47–63. ACM, 2000.

[105] Ravi S Sandhu. A lattice interpretation of the chinese wall policy. In *Proceedings of the 15th NIST-NCSC National Computer Security Conference*, pages 329–339. Citeseer, 1992.

[106] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11), 1993.

[107] Ravi S Sandhu and Pierangela Samarati. Access control: Principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

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

[109] Christian Schläger, Manuel Sojer, Björn Muschall, and Günther Pernul. Attribute-based authentication and authorisation infrastructures for e-commerce providers. In *Proceedings of the EC-Web*. 2006.

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

[111] Richard T Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *Proceedings of the IEEE CSFW*, 1997.

[112] Sankaran Sivathanu, Ling Liu, Mei Yiduo, and Xing Pu. Storage management in virtualized cloud environment. In *Procedings of The International Conference on Cloud Computing*, pages 204–211. IEEE, 2010.

[113] K. Sohr, M. Drouineaud, G.-J. Ahn, and M. Gogolla. Analyzing and managing role-based access control policies. *IEEE Transactions on Knowledge and Data Engineering*, 20(7):924–939, July 2008.

[114] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova. Resource allocation using virtual clusters. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 260–267, May 2009.

[115] M. Stillwell, F. Vivien, and H. Casanova. Dynamic fractional resource scheduling for HPC workloads. In *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pages 1–12, April 2010.

[116] Jakub 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.

[117] Bo Tang, Qi Li, and R. Sandhu. A multi-tenant RBAC model for collaborative cloud services. In *Procedings of The International Conference on Privacy, Security and Trust*, pages 229–238, 2013.

[118] Bo Tang and R. Sandhu. Cross-tenant trust models in cloud computing. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 129–136, 2013.

[119] Venkatanathan 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.

[120] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M. Swift. Resource-freeing attacks: Improve your cloud performance (at your neighbor's expense). In *ACM CCS*, 2012.

[121] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proc. of the ACM FMSE*, 2004.

[122] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75, April 2011.

[123] Jos B Warmer and Anneke G Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.

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

[125] Avi Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM (JACM)*, 30(4):729–735, 1983.

[126] Ruoyu Wu et al. ACaaS: Access control as a service for IaaS cloud. In *Procedings of The IEEE International Conference on Social Computing and Networking*, 2013.

[127] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1107–1117, 2013.

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

[129] Eric Yuan and Jin Tong. Attributed based access control (ABAC) for web services. In *proceedings of the IEEE ICWS*, 2005.

[130] Fengzhe 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.

[131] Xinwen Zhang, Ravi Sandhu, and Francesco Parisi-Presicce. Safety analysis of usage control authorization models. In *Proc. of the ASIACCS*, 2006.

[132] Yinqian 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.

[133] Yinqian Zhang et al. Cross-tenant side-channel attacks in PaaS clouds. In *ACM CCS*, 2014.

# VITA

Khalid Bijon was born and grew up in Rajshahi, Bangladesh. Following graduation from Govt. Laboratory High School, Rajshahi and New Govt. Degree College, Khalid received his Bachelor of Science degree with a major in Computer Science and Engineering from American International University Bangladesh, Dhaka, Bangladesh in 2008. He subsequently entered the doctoral program in the Department of Computer Science at the University of Texas at San Antonio in Fall 2009. He joined the Institute for Cyber Security at UTSA and started working with Dr. Ravi Sandhu since 2010. His research interests include security and privacy in cyber space. In particular, his focus is on developing constraints specification mechanism in attribute based access control and cloud Infrastructure-as-a-Service system.