Modeling and Analysis of State-of-the-Art VM-Based Cloud Management Platforms

Saif U. R. Malik, Samee U. Khan, *Senior Member, IEEE*, and Sudarshan K. Srinivasan, *Member, IEEE*

Abstract—Virtualization is one of the key aspects used in cloud computing environment to achieve scalability and flexibility. To cope with the large number of virtual machines (VM) involved in the cloud, several solutions have been proposed to automatically monitor and deploy VM in resource pools. Most of the cloud management system, such as Amazon EC2, are proprietary and are not generally available for research. In the said perspective, many open source VM-based cloud platforms launched for general users to research. The existing work performed in VM-based cloud management platforms have mainly focused on the discussion of architecture, feature set, and performance analysis. However, other important aspects, such as formal analysis, modeling, and verification are usually ignored. In this paper, we provide a formal analysis, modeling, and verification of three open source state-of-the-art VM-based cloud management platforms: 1) Eucalyptus, 2) Open Nebula, and 3) Nimbus. We have used high-level Petri nets (HLPN) to model and analyze the structural and behavioral properties of the systems. Moreover, to verify the models, we have used Satisfiability Modulo Theories Library (SMT-Lib) and Z3 Solver. We modeled about 100 VM to verify the correctness and feasibility of our models. The results reveal that the models are functioning correctly. Moreover, the increase in the number of VM does not affect the working of the models that indicates the practicability of the models in a highly scalable and flexible environment.

Index Terms—VM-based cloud management, high-level Petri nets (HLPN), SMT, Z3, modeling, and verification

1 Introduction and Motivation

The emergence of technological advances, such as multi-core processors and networked computing environments, has helped software practitioners to achieve the vision of creating a software paradigm for millions of users to use as a service [1]. Cloud computing is one such paradigm with which a shared pool of resources (networks, servers, storage, applications, and services) can be accessed conveniently and on-demand, and that can be rapidly provisioned or released with minimal management effort or service provider interaction [2]. Moreover, cloud reduced the cost of managing software and hardware resources by pushing the infrastructure to the network, which allowed users to access services anywhere around the world. The three main services provided by the cloud computing architectures are: 1) software as a service (SaaS), 2) platform as a service (PaaS), and 3) infrastructure as a service (IaaS). The SaaS is used to provide access to the applications that are managed by a third-party vendor and whose infrastructure is accessed on the client side. On the other hand, PaaS provides the platform for developing, running, and managing applications, such as Google App Engine (GAE). Finally, the IaaS, which is the focus of this paper, delivers computer infrastructure, such as virtualization, storage, and networking that offers incremental scalability [22]. The IaaS offers different capacity and storage as a virtual machines (VM) with varying prices on a pay-as-you-go basis [24]. The vibrant underlying technology in cloud infrastructure is virtualization that contributes toward the prevalent application and adaptation of cloud computing infrastructure [3].

Virtualization can be defined as the process of abstracting the original physical structure of innumerable technologies, such as hardware platform, operating system, a storage device, or other network resources [4]. Moreover, machines, applications, desktops, networks, and services are also separated from the underlying physical constraints. The cloud takes virtualization to a step further by using VMs that creates the customer independent system regardless of the underlying hardware in a timely manner. Every physical machine in a cloud can host several VMs, which from a user's perspective is equivalent to a fully functional physical machine. Moreover, VMs can start and stop anytime without any changes to the underlying hardware. Furthermore, migration of VMs between the physical machines is also possible without much disruption. Therefore, the cloud service providers deploy services on VMs that allow resource provision with more flexibility [5], [51].

The cloud normally involves a large number of VM and physical machines that makes virtual infrastructure management a cumbersome task. Several solutions are available to cope with the aforementioned problem, such as VMware virtual center, platform orchestration, and enomalism that provides an automatic monitoring and deployment of VMs in resource pools [6]. Numerous cloud providers, such as Amazon EC2 [7], Google App Engine [8], and Science Clouds [9] uses the aforementioned solutions to manage the virtual infrastructure. Most of the existing cloud computing management platforms are either proprietary or contain software that are not programmable for experimentation

• *The authors are with the Department of Electrical and Computer Engineering, North Dakota State University, 1411 Centennial Blvd, Fargo, ND 58105-5285. E-mail: {saif.rehmanmalik, samee.khan, sudarshan.srinivasan}@ndsu.edu.*

Manuscript received 14 Feb. 2013; revised 31 July 2013; accepted 2 Aug. 2013; published online 13 Aug. 2013.
Recommended for acceptance by U. Bellur.
For information on obtaining reprints of this article, please send e-mail to: tcc@computer.org, and reference IEEECS Log Number TCC-2013-02-0024.
Digital Object Identifier no. 10.1109/TCC.2013.3.

purposes. In the said perspective, several open source VM-based cloud management platforms have been launched, such as Eucalyptus [10], oVirt [11], and Enomaly Elastic Compute Platform (ECP) [12], so that researchers from every field can participate toward further development of management platforms in the cloud. Recently, OpenStack [25] has attained a significant status in the field of cloud computing. A number of companies that includes some big names, such as IBM, Dell, AMD, and Intel, have joined the OpenStack project.

Several open source IaaS providers have emerged as a result of recent development in open source virtualization [13]. Two hypervisors: 1) Xen [14] and 2) KVM [15] are the most widely used open source hypervisors in the recent IaaS providers [34]. In this paper, we have studied and analyzed three open source state-of-the-art VM-based cloud management platforms: 1) Eucalyptus, 2) Open Nebula, and 3) Nimbus. The said systems have different design interests (as advocated in [16]) and that is why we have selected these systems for the study. The differences in the designs make each system suitable for an explicit environment. An important aspect that influences the choice of selecting a particular system for a private cloud is the level of customization. Among all of the aforementioned three cloud platforms, Open Nebula provides the highest level of customizability that allow users to switch almost every component from the underlying virtual machine monitor (VMM) to the front-end. Both, the end user and the administrator, relishes the available customization. The customization provided by Open Nebula is suitable in an experimental environment, where one wants to explore every component and crack new results from the computational perspective. Besides Open Nebula, Nimbus also provides a high level of customization. However, the major portion of customization in Nimbus is available to the administrator. Nimbus is more suitable for an environment, where one is less interested in technical details of the systems, but requires a broad level of customization, such as cooperative scientific communities. Eucalyptus mimics the implementation of Amazon EC2 and is an open source implementation of Amazon web service (AWS) API. The customization level is Eucalyptus is very low that makes it appropriate for a private company, where one needs a cloud for own use and wants to avoid mistakes from the users. The suitability of any of the cloud management platforms depends on the requirements of the user or organizations.

Numerous studies are available, such as [17], [18], [19] that discuss and compare the aforementioned cloud management platforms but the previous work largely focused on the architecture and feature set of the systems. Another mutual aspect observed among the previous studies is the high level of abstraction, while discussing the architectures of the systems. In this study, we made an effort to diminish the level of abstraction through detailed modeling and formal analysis of the platforms being discussed. We have used high-level Petri nets (HLPN) and Z language for the modeling and analysis of the systems. HLPN is used to: 1) simulate the systems and 2) provide mathematical representation, to analyze the behavior and structural properties of the system. The model of the systems will help analyze: 1) the interconnection of the components and processes, 2) the fine-grain details of the flow of information among the processes, and 3) how the information is processed. Moreover, we performed the verification of the models in two-fold. First, we performed the automated verification of the models by following the bounded model checking technique using Satisfiability Modulo Theories Library (SMT-Lib) and Z3 solver. To verify using SMT, the Petri Net models are first translated into SMT along with the specified properties. Then, Z3 solver is used to check either the model satisfies the properties or not. Second, to verify the feasibility of the models as the number of VMs scales, we model about hundred instances of VMs for each platform (Eucalyptus, Open Nebula, and Nimbus) and verify the correctness. The results generated reveals that the models are working correctly. To the best of our knowledge, no work has been done to model, analyze, and verify the open source cloud management platforms. This research work will provide the basis for the researchers to understand the design and implementation of the state-of-the-art VM-based cloud platforms. As the inception of the cloud is based on distributed computing (grid and cluster) and virtualization, the research is more inclined toward the computing and storage aspect of the cloud and another crucial aspect of cloud, the connectivity (networking), is usually forgotten [13]. This paper will focus on the intercommunication and behavior of the components of the systems rather than the computing and performance measurements.

The remainder of the paper is organized as follows: Section 2 will discuss some preliminaries tools and technologies used in the paper; modeling, analysis, description, and comparison of VM-based cloud management systems will be discussed in Section 3; verification and results of the models of the VM-based cloud systems (Eucalyptus, Open Nebula, and Nimbus) will be explained in Section 4; in Section 5, we will provide some of the related work done in the field of VM-based cloud systems; we will conclude the paper in Section 6, followed by the references and bibliographies of the authors.

2 Preliminaries

This section will discuss some of the tools and technologies used in this work that will help the reader to understand the paper easily.

2.1 High-Level Petri Nets

Petri nets are graphical and mathematical modeling tool that is applicable to many systems characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, or stochastic [20]. In this paper, we have used a variant of the classical Petri Net model, namely, HLPN. (Readers are encouraged to see [20], [21] for an elaborate introduction to Petri nets.)

Definition 1 (HLPN) [20]. *A HLPN is a 7-tuple $N = (P, T, F, \varphi, R, L, M_0)$ here:*

1. *P is a set of finite places.*
2. *T is a set of finite transitions such that $P \cap T = \emptyset$.*
3. *F is a flow relation such that $F \subseteq (P \times T) \cup (T \cup P)$.*

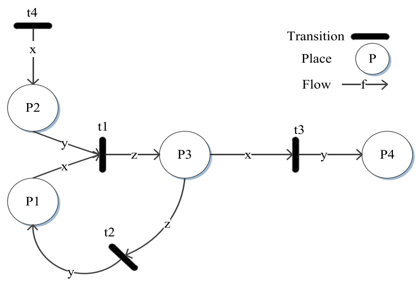

Fig. 1. An example high-level Petri net.

4. *φ is a mapping function that maps P to data types such that $\varphi : P \rightarrow Type$.*
5. *R define rules that maps T to predicate logic formulas such that $R : T \rightarrow Formula$.*
6. *L is a label that maps F to labels such that $L : F \rightarrow Label$.*
7. *M_0 is the initial marking where $M : P \rightarrow Tokens$.*

The first three variables (P, T, F) provide information about the structure of the net and the next three variables $(\phi, \mathrm{R}, \mathrm{L})$ provide the static semantics, which means the information does not change throughout the system.

The use of HLPN is preferred over low-level Petri nets (LLPN) because in LLPN: 1) no distinction is available between the tokens, no types, or just one type, 2) for transition enablement there is no selection of specific tokens even using flow capacity, and 3) a place may be viewed as a structural variable, such as array that is not possible to depict.

Let α and β be the nodes of the HLPN N iff $\alpha, \beta \in P \cup T$. A node α is an input node of another node β iff there is a directed arc from α to β such that $(\alpha, \beta) \in F$. Node α is an output node of β iff $(\beta, \alpha) \in F$. For any node $\alpha \in P \cup T$, the precondition is $\bullet\, \alpha = \{\beta \mid (\beta, \alpha) \in F\}$ and postcondition is $\bullet\, \alpha = \{\beta \mid (\alpha, \beta) \in F\}$.

In HLPN, places can have tokens of different types and can also be a cross product of two or more types, such as in Fig. 1 the places are mapped to the types: $\varphi(P1) = \mathrm{Bool}$, $\varphi(P2) = \mathrm{ID}$, $\varphi(P3) = \mathbb{P}(\mathrm{Int})$, $\varphi(P1) = \mathrm{Char}$.

The preconditions must hold for any transition to be enabled. Moreover, the variables from the incoming flows are used to enable a certain transition. For example, preconditions for *t1* will use x and y from *P1* and *P2*, respectively. Similarly, postcondition uses variables from outgoing flows for transition firing, and can be written as: $R(t1) = (\mathrm{x} = \mathrm{true}) \wedge (\mathrm{y} \geq 50) \wedge (4 \leq \mathrm{z} \leq 20)$.

2.2 SMT-Lib and Z3 Solver

Satisfiability modulo theories (SMT) is an area of automated deduction for checking the satisfiability of formulas over some theories of interest and has the roots from Boolean satisfiability solvers (SAT) [23]. The difference between SMT and SAT is that SMT solvers check the satisfiability of first-order formulas containing operations from several theories, such as Bit-vector and arithmetic, whereas SAT solvers check the satisfiability of propositional formulas [26]. SMT-Lib provides a common input platform and benchmarking framework that helps in the evaluation of the systems [27]. SMT has been used in many fields including deductive software verification. Moreover, recent applications of computer science including planning, model checking, and automated test generation finding, also considers SMT as an important verification tool [27]. (Readers are encouraged to read [28] for the use of SMT-Lib in the verification of OSPF routing protocol [50].) Multiple solvers are available that supports SMT-LIB, such as Beaver, Boolector, CVC4, MathSAT5, Z3, and OpenSMT. The solver can be distinguished among the features they provide, such as, underlying logic (example first order or temporal), background theories, input formulas, and interface [26].

We used Z3 solver in our study, which is a high-performance theorem prover developed at Microsoft research. Z3 is an automated satisfiability checker. Moreover, Z3 also checks whether the set of formulas are satisfiable in the built-in theories of SMT-Lib. Readers are encouraged to see [30], for the detailed information about the working and commands of Z3 solver.

3 Modeling and Analysis of VM-Based Cloud Management Platforms

VM-based cloud management platforms offer several advantages that include

1. better isolation,
2. scalability,
3. availability, and
4. flexibility.

Looking at the benefits provided by the VM-based systems to the cloud, a renewed interest of research has emerged in different classes of virtualization, such as desktop, server, application, storage, and network from several industry giants, such as VMware, Red Hat, and Microsoft [32]. In this section, we will discuss, model, and analyze three VM-based cloud management platforms: 1) Eucalyptus, 2) Open Nebula, and 3) Nimbus.

3.1 Components of Open Source Cloud

Before going into the details and modeling of the systems, in this section, we will provide a quick overview of the components of the generic open source cloud computing systems. The components are classified as follows:

1. hardware and operating systems (resides on the physical machine and must be setup properly for any software system to work),
2. network (includes DNS, DHCP, and subnet organization of the physical machines),
3. hypervisor (includes Xen and KVM, which provides framework for VMs to run),
4. VM disk images (every cloud has a repository of disk images that can be copied and used as a basis for new virtual disks),
5. interface (front-end tools to request VMs and specify parameters), and
6. cloud framework (includes Eucalyptus or any other framework).

The aforementioned components generally make the entire software stack for the cloud computing systems [34]. We used the said components in our model to depict the working of the systems.

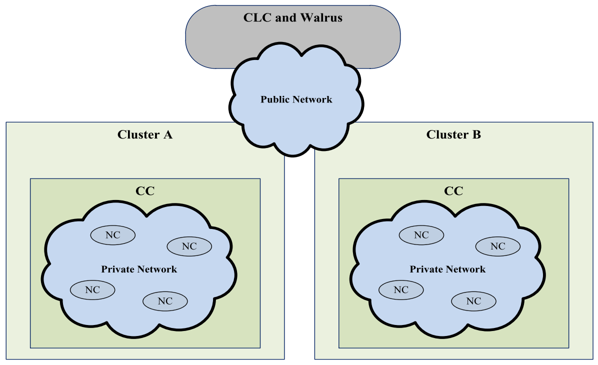

Fig. 2. Eucalyptus architecture.

3.2 Eucalyptus

Eucalyptus is an open source VM-based cloud computing management framework that enables users to run and control the instance of virtual machines deployed at several physical resources [35]. Eucalyptus was first initiated at the University of California at Santa Barbara and is now supported by the Eucalyptus Systems, Inc [36]. The emphasis of the system was to develop an architecture that will allow scientists to experiment cloud related software and architecture. One of the advantages of Eucalyptus is that it uses Amazon Web Service APIs and provides the same interface as of Amazon's EC2 and Simple Storage Service (S3) in a private cluster. Therefore, provides a well-known tool to host and manage VMs.

Installation of Eucalyptus consists of several components:

1. cloud controller (CLC),
2. cluster controller (CC),
3. storage controller (Walrus), and
4. node controller (NC).

The architecture of Eucalyptus (Fig. 2) is kept simple, flexible, and hierarchical, where every component is implemented as a stand-alone web service. The components implemented as a web service has following benefits [35]: 1) exposure to a well-defined WSDL document that contains operations being performed and the input/output data structures and 2) web features can be extended, such as security policies to secure the communication between components.

CLC is the entry point to the cloud that provides configuration interface for managing cluster and instances, Walrus configuration, and user registration. Main responsibilities of CLC include: 1) translation of user initiated commands to CC, 2) making high-level scheduling decisions, and 3) management of underlying virtualized resources. CC chose the compute node to provision the VM on receiving the command from the CLC. Moreover, gathering information, scheduling VM execution on certain NC, and virtual instance overlay network management for smooth transmission of requests are the responsibilities of CC. Walrus provides a storage service to store virtual machine images and user data. Execution, termination, and inspection of VM instances are performed by the NC. A query is performed by NC to discover the nodes physical resources, such as no. of cores, size of memory, and state of VM instances.

3.2.1 Modeling and Analysis

The model of spawning a VM instance in Eucalyptus configuration is illustrated in Fig. 3. As stated in Definition 1, the HLPN is a 7-tuple $N = (P, T, F, \varphi, R, L, M_0)$. To begin modeling the system, we first need to specify P and the associated types. As depicted in Fig. 3, there are 10 places in the model. The names and mapping of P are shown in Table 1. The types used in the model are illustrated in Table 2.

The next step is to define the set of rules, preconditions, and postconditions to map to T. Before going to the next step let us have a quick overview of the process of initiating a VM instance.

To make a request for an instance of VM, the user first needs to configure the front-end. The default front-end is euca2ools, which is similar to the front-end of Amazon EC2.

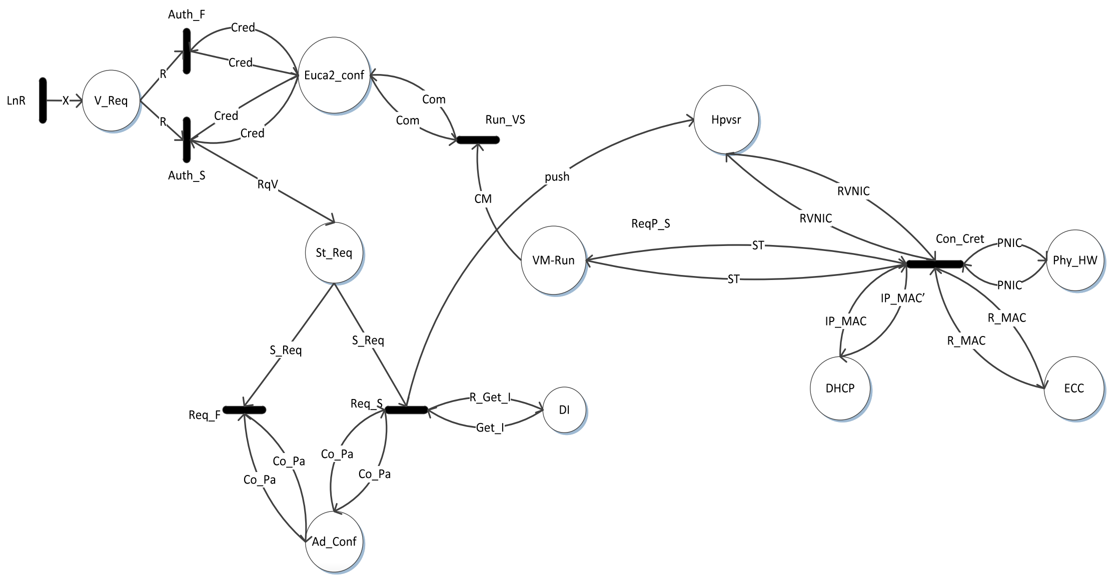

Fig. 3. Model of starting a VM instance in Eucalyptus.

TABLE 1
Places and Mappings of Eucalyptus

Place	Mapping	Description
φ (V_Req)	ℙ(Key×Env_Var× CPU×Mem× Disk)	Holds environment variables and user config.
φ(Euca2_conf)	ℙ(Key×Env_Var)	Pre-set config.
φ (St_Req)	ℙ(CPU×Mem×Disk ×Key)	Intermediate place to hold the configuration values
φ (Ad_Conf)	ℙ(CPU×Mem×Disk)	Hold admin config.
φ (DI)	ℙ(EMI)	Holds the disk images
φ (Hpvsr)	ℙ(CPU×Mem×Disk× EMI× NIC×VID ×Key×Env_Var)	Holds user config. and creates virtual NIC and VM ID
φ (DHCP)	ℙ(IP × MAC)	Holds the mappings of IPs to MAC
φ (ECC)	ℙ(MAC)	Generate and hold random MAC for VM
φ (Phy_HW)	ℙ(CPU×Mem×Disk× NIC)	Holds physical specefication of the system
φ (VM-Run)	ℙ(CPU×Mem×Disk× EMI×NIC×VID×IP× MAC×Key×Env_Var)	Instance of VM is finally created alongwith the specified config.

To configure euca2ools, the user must download some files along with the keys and instruction. Once, certain environment variables are set the euca2ools is ready to work. The user then uses euca2ools to request the VM. Moreover, the user has to select a configuration for required memory, CPU, and the hard drive space from one of the five preset configurations set by the administrator, for the requested VM. When the head node receives the request, a VM template disk image is extracted from the disk repository and is pushed toward the compute node. The disk image is padded and packaged to be used for the hypervisor. Eucalyptus Cloud Controller (ECC) generates a random MAC address and assigns it to VM instance.

CC setup a static entry of MAC/IP pair and passes it on to NC. The NC maps virtual NIC of the instance to the physical NIC of the node through network bridging. The instance is initiated on the hypervisor and then the user can directly interact with the VM instance.

We have discussed the process of instantiating the VM and now we can define formulas (pre and postconditions) to map on transitions. The set of transitions $T = \{LnR, Auth\_F, Auth\_S, Req\_S, Req\_F, Con\_Cret, Run\_VS\}$.

New tokens can enter the model only through LnR transition. The rule for the token creation can be stated as: $R(LnR) = \exists t \in T \mid \bullet\ t = \emptyset$.

The next two transitions are $Auth\_S$ and $Auth\_F$, which authenticate the configuration of euca2ools front-end. The said transitions are mapped to the following formulas:

$$\begin{aligned} \boldsymbol{R}(Auth\_S) = \forall\, cr \in Cred \mid R[1] = cr[1] \\ \wedge R[2] = cr[2] \\ \wedge \forall\, rq \in R \mid RVM := rq \\ \wedge Cred' := Cred, \end{aligned} \tag{1}$$

TABLE 2
Data Types Used In The Model of Eucalyptus

Types	Description
Key	A string type for euca2ools key authentication
Env_Var	A string type for euca2ools environment variables authentication
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
IP	A string type for the IP address of VM
MAC	A string type for the MAC address of the VM
NIC	A string type for NIC of the physical machine
VID	An Integer type for VM ID

$$\begin{aligned} \boldsymbol{R}(Auth\_F) = \forall\, cr \in Cred \mid R[1] \neq cr[1] \\ \wedge R[2] \neq cr[2] \\ \wedge Cred' := Cred. \end{aligned} \tag{2}$$

The formula in (1) depicts the success scenario when the euca2ools is able to find both of the credentials (the key and environment variables) in the systems and both are set properly. Similarly, in (2) if the euca2ools is unable to locate the specific environment variable or if the key is mismatched, then no further transitions will be fired. After the authentication is successfully performed the next step is to check either the configurations provided by the user for the size of memory, disk, and CPU for the requested VM are same as the ones set by the administrator.

$$\begin{aligned} \boldsymbol{R}(Req\_S) &= \forall\, co \in Co\_Pa \mid S\_Req[1] \\ &= co[1] \wedge S\_Req[2] = co[2] \wedge S\_Req[3] \\ &= co[3] \wedge \exists\, rg \in R\_Get\_I, \exists\, cn \in push \mid cn[4] := rg \\ &\quad \wedge\ S\_Req[1] := cn[1] \wedge S\_Req[2] := cn[2] \\ &\quad \wedge S\_Req[3] := cn[3], \end{aligned} \tag{3}$$

$$\begin{aligned} \boldsymbol{R}(Req\_F) = \forall\, co \in Co\_Pa \mid S\_Req[1] \neq co[1] \\ \wedge S\_Req[2] \neq co[2]\ R[2] \\ \wedge S\_Req[3] \neq co[3]. \end{aligned} \tag{4}$$

The administrator configurations reside in the Ad_Conf and the user configurations are placed in St_Req after the transition $Auth\_S$ is fired. In (3) and (4) both of the configurations (user and administrator) are compared. If (3) is fired, then a disk image from a disk image repository is extracted and is transferred to $Hpvsr$ along with the configuration parameters. If (4) is fired, then no further transition will be fired because of the configurations mismatch.

Mapping is performed in (5), where a random MAC from ECC is generated and assigned to the VM. Moreover, physical NIC from Phy_HW and a virtual NIC from $Hpvsr$ are also mapped. The relation (mapping) between virtual NIC and physical NIC is one-to-many, which means that

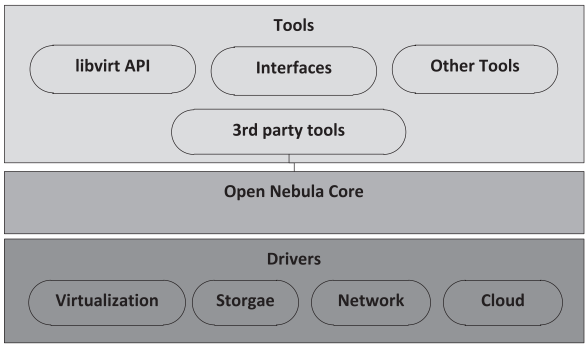

Fig. 4. The architecture of Open Nebula.

many virtual $NICs$ can be mapped to one physical NIC. However, the relation between MAC/IP pair and virtual NIC is one-to-one because only a single virtual NIC and a MAC/IP pair can be assigned to a single instance of VM. After all the mapping is completed, the instance of the VM is created with the specified configuration parameters placed at VM_Run. The user can directly interact with the VM instance after it is created using Run_VS.

$$\begin{aligned}
\boldsymbol{R}(Con\_Cret) &= \{\forall\, im[1] : IP\_MAC| \\
&\quad \forall\, ma : R\_MAC \mid \forall\, vn[5] : RVNIC, \\
&\quad (ma \leftrightarrow im[1]) \leftrightarrow vn[5]\} \\
&\wedge \forall\, pn[4] \in PNIC \mid pn[4] \mapsto vn[5] \\
IP\_MAC' &= IP\_MAC \cup \{(im[1], ma)\} \\
R\_MAC' &= R\_MAC \cup \{ma\} \\
ST' &= ST \cup \{(vn[1], vn[2], vn[3], vn[4], \\
&\quad vn[5], vn[6], vn[7], vn[8], im[1], ma)\}.
\end{aligned} \tag{5}$$

The design of Eucalyptus supports corporate enterprise computing settings, where the administration space is separated from the user space. The users are only allowed to use the system through web interface or specified front end tools. Eucalyptus is easy to deploy on top of the existing resources. Moreover, Eucalyptus is suitable for experimentation because of having modular design and open source in nature.

3.3 Open Nebula

Open Nebula was a research project that started in a year 2005 as a management tool for the orchestration and configuration of VMs in datacenter [38], [39]. Open Nebula is now available as an open source and can be used as a toolkit to build private, public, and hybrid clouds. The key technical aspect of Open Nebula is its architecture that provides a great level of customization and centralization. Moreover, the architecture also supports multiple storage back ends and different hypervisors, such as Xen, VMware, and KVM [39]. Shared file system is adopted in Open Nebula for storing all functional and disk images files. The aforementioned exposes the underlying features of libvirt to administrators and users that involves operations, such as VM live migrations. The centralization makes administration of Open Nebula easier. However, one drawback of the default customization with NFS file system is that large amount of space is required to hold all the files.

The architecture of Open Nebula (Fig. 4) is divided into three layers: 1) Tools, 2) Core, and 3) Drivers. The first layer contains management tools that can be developed using the Open Nebula core interfaces, such as command line interface, new Open Nebula cloud API, or third party tools that can be created easily using the XML-RPC interface [38]. The Open Nebula core performs orchestration and configuration of other components. Moreover, the core also has a set of components that are used to control and monitor VM, virtual networks, hosts, and storage. The drivers are pluggable modules that provide a layer of abstraction over the lower level operations, such as virtualization hypervisor, cloud services, and file transfer mechanism. Moreover, the drivers are used by the core layer to perform certain actions, such as cancelling a VM.

3.3.1 Modeling and Analysis

The model for initializing the VM using typical Open Nebula configuration is demonstrated in Fig. 5. The first

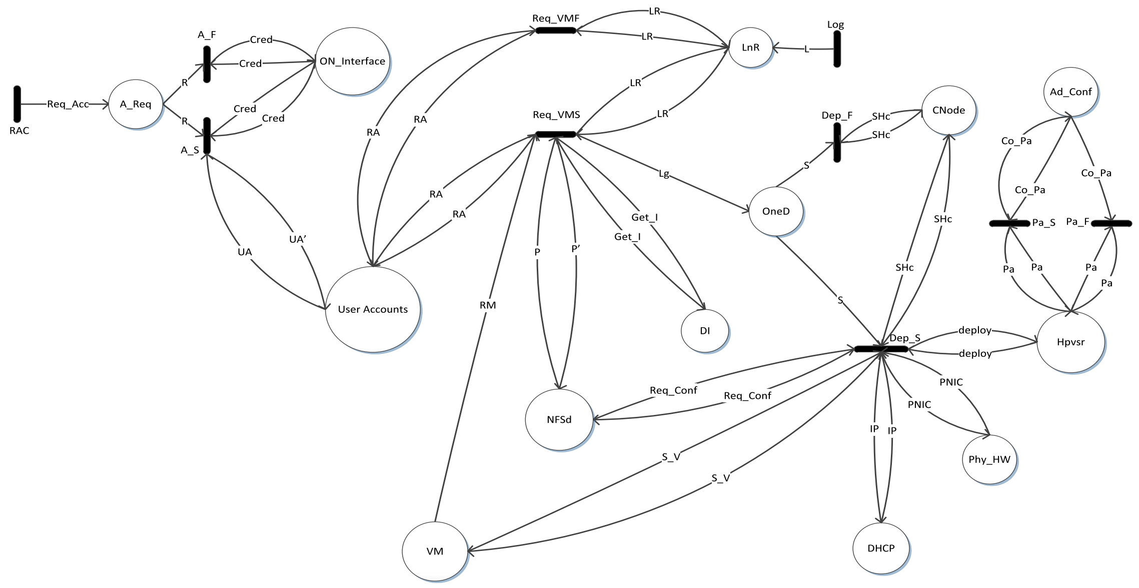

Fig. 5. Open Nebula model for instantiating a VM.

TABLE 3
Data Types Used in the Model of Open Nebula

Types	Description
Email	A string type for email authentication.
Pass	A string type for password authentication.
UName	A string type for password authentication.
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
NMI	Type for the machine image.
SSH_Cert	A string type for the SSH login
SSH_Pass	String type for the SSH encrypted password.
IP	A string type for the IP address of VM
MAC	A string type for MAC address of the VM
NIC	A string type for NIC of the physical machine
VID	An integer type for Virtual Machine ID

step toward modeling the system is to identify the required types, P, and mapping. The types and the descriptions are shown in Table 3, and the mapping of P to types is depicted in Table 4.

To use an Open Nebula cloud the user needs to have an account that Open Nebula provides on demand. After a successful sign up, the user can login with any one of the interface being used, such as sunstone, OCCI, and EC2. The user can request a VM using a command *onevm*, which allows user to manage VMs, such as allocate, deploy, suspend, and shutdown. The NFS directory at the head node holds all the functional and disk image files. As a result of *onevm*, the VM template disk image file is copied from the disk image repository, padded to the required size and configuration, and is saved to the NFS directory. At that point, the Open Nebula Daemon (oned) that is responsible for the control of VM life cycle and to coordinate the operations of all modules, logs into the compute node. The compute node provided a virtual NIC and MAC, and mapped it to physical NIC through network bridging. Finally, the instance is created with the specified configurations at the hypervisor. In the previous section, we have provided a short overview for the process of instantiating the VM in a typical Open Nebula configuration. Now, we can define formulas (pre- and postconditions) to map on transitions. The set of transitions

$$T = \{RAC, log, A\_F, A\_S, Req\_VMF, Req\_VMS, Dep\_S, \\ Dep\_F, Pa\_S, Pa\_F\}.$$

New tokens can only be produced by transition RAC and Log. As seen in Fig. 5, no arc is incident on any of the two aforementioned transitions, which is why no precondition exists and the rules for the transitions can be written as: $\boldsymbol{R}(RAC) = \exists r \in R \mid \bullet\, r = \emptyset$ and $\boldsymbol{R}(Log) = \exists lg \in L \mid \bullet\, lg = \emptyset$.

The first step performed by the user is to request an account for an Open Nebula cloud. The transitions A_F and A_S authenticate if the requested user already holds an account or not. The transitions are mapped to the following formulas:

TABLE 4
Places Used In the Model of Open Nebula

Places	Mappings	Descriptions
$\varphi\,(A\_Req)$	*ℙ(Email×Pass)*	*Holds user requests*
$\varphi\,(ON\_Interface)$	*ℙ(Email×Pass)*	*Holds existing users*
$\varphi\,(LnR)$	*ℙ(Pass×UName×CPU×Mem×Disk)*	*Holds user login and config.*
$\varphi\,(User\ Accounts)$	*ℙ(Pass×UName)*	*Holds user accounts*
$\varphi\,(CNode)$	*ℙ(Cert×SSH_Pass)*	*Holds login information for oned*
$\varphi\,(DI)$	*ℙ(NMI)*	*Holds the disk images*
$\varphi\,(NFSd)$	*ℙ(CPU×Mem×Disk×NMI×UName)*	*Holds username and user config.*
$\varphi\,(OneD)$	*ℙ(Cert×SSH_Pass)*	*Holds oned login detail*
$\varphi\,(Ad\_Conf)$	*ℙ(CPU×Mem×Disk)*	*Holds admin config.*
$\varphi\,(Hpvsr)$	*ℙ(CPU×Mem×Disk×NMI×NIC×VID×MAC)*	*Hold config., creates virtual NIC, MAC, and VM ID*
$\varphi\,(DHCP)$	*ℙ(IP)*	*Creates IP for the VM*
$\varphi\,(Phy\_HW)$	*ℙ(CPU×Mem×Disk×NIC)*	*Holds physical specefication of the system*
$\varphi\,(VM)$	*ℙ(CPU×Mem×Disk×NMI×NIC×VID×UName×IP)*	*VM instance is created alongwith the specified config.*

$$\begin{aligned} \boldsymbol{R}(A\_S) = \forall\ cr \in Cred \mid cr[1] \neq R[1] \\ \wedge\ UA' := UA \cup\ \{(R[1], R[2])\}, \end{aligned} \tag{6}$$

$$\begin{aligned} \boldsymbol{R}(A\_F) = \forall\ cr \in Cred \mid \exists\ cr[1] \\ = R[1]\ Cred' := Cred. \end{aligned} \tag{7}$$

The accounts are created based on the email ID. If the email ID is already associated with an account, then the request is denied. Otherwise, the account is created and the new information is stored in the $UserAccounts$. The success and failure scenario is depicted in (6) and (7), respectively. The next step is to login to the Open Nebula cloud and request for the VM.

$$\begin{aligned} \boldsymbol{R}(Req\_VMS) = \forall\ r \in RA \mid r[1] = LR[1] \wedge r[2] \\ = LR[2] \wedge \exists\ np \in P \mid np[1] := LR[3] \\ \wedge\ np[2] : = LR[4] \wedge np[3] := LR[5] \wedge np[5] : \\ = LR[2] \wedge \forall\ g \in Get\_I \mid np[4] := g \\ P' = P \cup \{(np[1], np[2], np[3], np[4], np[5])\}, \end{aligned} \tag{8}$$

$$\begin{aligned} \boldsymbol{R}(Req\_VMF) = \forall\ r \in RA \mid r[1] \neq LR[1] \\ \wedge\, r[2] \neq LR[2] \\ \wedge\, RA' := RA. \end{aligned} \tag{9}$$

The user account information is stored in $UserAccounts$. When the user logs in and requests for a VM, the login credentials are matched and then the command is forwarded, as shown in (8) and (9). If (8) is fired, then the disk image is copied from the disk image repository, padded to correct size and configuration, and is stored in $NFSd$. Moreover, *oned* will login to the compute node only when

(8) is fired. If (9) is fired, then the request for the VM is denied and no further transitions will be fired.

To spawn a VM user provides a configuration file with parameters to be fed into the hypervisor command line. The aforementioned allow users to request for any configuration of memory, disk, and CPU. Therefore, we have performed the authentication of configuration parameters in (10) and (11) when the hypervisor is generating virtual NIC and MAC. In (10), if the configurations provided by the user are same as set by the administrator, then the control is transferred back to $Hpvsr$. Otherwise, (11) is fired and the system is terminated.

$$\begin{aligned}\boldsymbol{R}(Pa\_S) &= \forall\ cp \in Co\_Pa \mid cp[1] = Pa[1] \\ &\wedge cp[2] = Pa[2] \wedge cp[3] = Pa[3],\end{aligned} \tag{10}$$

$$\begin{aligned}\boldsymbol{R}(Pa\_F) &= \forall\ cp \in Co\_Pa \mid cp[1] \neq Pa[1] \\ &\wedge cp[2] \neq Pa[2] \wedge cp[3] \neq Pa[3].\end{aligned} \tag{11}$$

The $Oned$ process uses secure shell (SSH), which is an encrypted network protocol to securely send management functions, to login to the compute node using SSH certificate and password. If (12) is fired, then the virtual NIC and MAC from $Hpvsr$ and physical NIC from Phy_HW are mapped using network bridging. The relation between virtual MAC and NIC is one-to-one. The relation between the pair of MAC/NIC and physical NIC is many-to-one, which means one physical NIC can be mapped to many. Once the mapping is completed an IP from $DHCP$ is assigned to the VM and the instance is ready to use. If (13) is fired, then the model exits because the SSH certificate or the password provided is incorrect,

$$\begin{aligned}\boldsymbol{R}(Dep\_S) &= \forall\ sc \in SHc \mid \exists\ sc[1] = S[1] \wedge sc[2] \\ &= S[2] \wedge \forall\ rc \in Req\_Conf, \exists d \in deploy \mid d[1]: \\ &= rc[1] \wedge d[2] := rc[2] \wedge d[3] := rc[3] \wedge d[4]: \\ &= rc[4] \wedge \{\forall\ d[5] : deploy \mid \forall\ d[7] : deploy \mid \forall\ pn[4]: \\ &\quad PNIC, (d[7] \leftrightarrow d[5]) \ \mapsto\ pn[4]\} \\ &\quad \wedge \{\exists\ i : IP \mid s[6] : S\_V, i \leftrightarrow s[6]\} deploy' \\ &= deploy \cup \{(d[1], d[2], d[3], d[4], d[5], d[6], d[7])\} S\_V \\ &= S\_V \cup\ \{(d[1], d[2], d[3], d[4], d[5], d[6], rc[5])\},\end{aligned} \tag{12}$$

$$\begin{aligned}\boldsymbol{R}(Dep\_F) &= \forall\ sc \in SHc \mid sc[1] \neq S[1] \\ &\wedge sc[2] \neq S[2].\end{aligned} \tag{13}$$

The level of customization available in Open Nebula is suitable for researchers who wish to combine cloud systems with other technologies. However, to utilize the underlying benefits of the customization the user needs to have some technical expertise. Another downside of the customization is that user can make a mistake while providing configuration for a VM. The centralized nature of Open Nebula makes administration easier. Moreover, higher level of customization makes Open Nebula ideal for research community.

3.4 Nimbus

Nimbus is an open source solution that allows clients to lease resources by deploying VM and providing an environment suitable for the user [40]. Nimbus is also affiliated with the Globus project [41] and uses Globus credential for user authentication. Nimbus also provides a high level of customization just like Open Nebula. However, the only difference is that much of the customization in Nimbus is restricted to the administrator. Moreover, several components, such as image repository and credentials for user authentication, are kept constant. Furthermore, Nimbus also provides an extensible implementation that supports web service resource framework (WSRF), Amazon EC2, and other end user services to make a cloud easy to use. A storage cloud implementation called Cumulus, which is compatible with Amazon Web Service S3 REST API, is included in the Nimbus and is tightly with other central services.

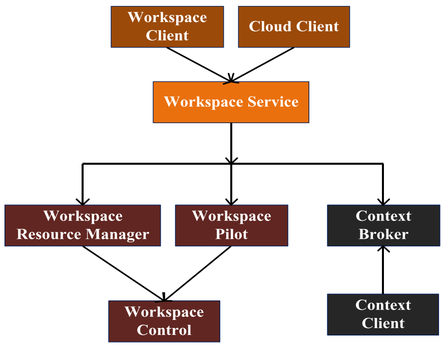

Fig. 6. Nimbus workspace components.

A toolkit is offered to deploy applications on Nimbus that consists of manager service hosting and image repository [42]. The components of Nimbus workspace is shown in Fig. 6. The *Workspace Service* is a standalone VM manager that can be invoked by remote protocol frontends. Currently, Nimbus supports two frontends: 1) EC2 and 2) WSRF. The storage service (Cumulus) is also embedded in a workspace service and can also be installed separately. The *workspace resource manager* deploys and manages workspace nodes. The *workspace pilot* allows the integration of preconfigured resources to VMs. Moreover, the aforementioned component also handles signals and has administration tools. The *workspace control* is responsible for the management and control of VM instances, disk images, VM integration to a network, and assigning MAC and IP addresses. The *workspace client* provides access to the entire feature set of WSRF as a command line client. The aim of *cloud client* is to speed up the process of running a VM using instance launches or one-click clusters. The clients can launch large virtual cluster automatically with the use of the *context broker*. The context client interacts with context broker at VM startup and lives on VM.

3.4.1 Modeling and Analysis

The model for starting a VM in a typical Nimbus configuration is demonstrated in Fig. 7. The first step is to identify the required P and associated types. Table 5 depicts the types used in the model and Table 6 explains the P and mappings.

The Nimbus uses client known as cloud client to interact with the services over multiple protocols. The users first need to download the client and configure it. The use of cloud clients makes life easy for the users. The easiest client to use is *cloud client* that makes the users up and running in

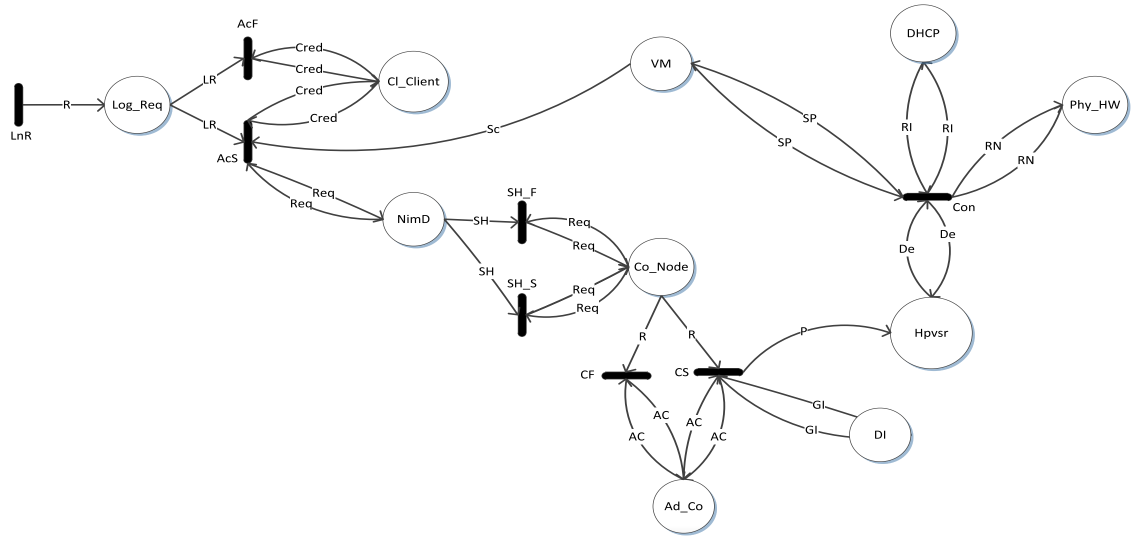

Fig. 7. A model for Nimbus.

a quick time. The user uses cloud clients to request a VM using explicit credentials. When the head node receives the request, then the VM template disk image is extracted from the repository and pushed to the compute node. The compute node performs the necessary padding and configuration to the image. Virtual MAC and NIC are provided by the compute node using network bridging. Moreover, the physical NIC and virtual NIC of the VM are also mapped. In Nimbus, every compute node has a DHCP server. The MAC/IP pair is placed in the DHCP server and the VM is ready to be used. The set $T = \{LnR, AcF, AcS, SH\_F, SH\_S, CF, Con, CS\}$.

The seed point of the model from where new tokens are generated is through a transition LnR and the rule for the transition can be stated as: $\boldsymbol{R}(LnR) = \exists t \in T \mid \bullet\, t = \emptyset$.

The transitions AcF and AcS authenticate the credentials of the cloud client. The said transitions are mapped to the following rules:

$$\begin{aligned}\boldsymbol{R}(AcS) = \forall\ cr \in Cred \mid \exists LR[1] \in cr[1] \\ \wedge\, LR[2] \in cr[2] \\ \wedge\, Cred' := Cred,\end{aligned} \tag{14}$$

$$\begin{aligned}\boldsymbol{R}(AcF) = \forall\ cr \in Cred \mid LR[1] \neq cr[1] \\ \wedge\, LR[2] \neq cr[2] \\ \wedge\, Cred' := Cred.\end{aligned} \tag{15}$$

The formula in (14) represents a success scenario when the cloud client successfully locates both of the credentials (user certificate and user key) and both are configured. In (15), if the credentials are mismatched, then no further

TABLE 5
Data Types Used in the Model of Nimbus

Types	Description
UCert	A string type for cloud client authentication.
UKey	A string type for cloud client authentication.
CPU	An integer type for the number of CPU/core allocated to the VM.
Mem	A float type for the amount of memory allocated to the VM.
Disk	A float type for the amount of disk space allocated to the VM.
IP	A string type for the IP address of VM
MAC	A string type for the MAC address of the VM
NIC	A string type for NIC of the virtual/physical machine
VID	An Integer type for Virtual Machine ID
SSH_Cert	A string type for the login authentication of SSH Certificate.
SSH_Pass	A string type for the login authentication of SSH encrypted password.

TABLE 6
Places Used In the Model of Nimbus

Places	Mappings	Descriptions
φ (*Log_Req*)	$\mathbb{P}$ (*UCert×UKey×CPU×Mem×Disk*)	*Holds user credentials and config.*
φ (*Cl_Client*)	$\mathbb{P}$ (*UCert×UKey*)	*Existing user details*
φ (*NimD*)	$\mathbb{P}$ (*UCert×SSH_Cert ×SSH_Pass×CPU×Mem×Disk*)	*Holds SSH login details and user config.*
φ (*Co_Node*)	$\mathbb{P}$ (*UCert×SSH_Cert ×SSH_Pass×CPU×Mem×Disk*)	*Holds user and SSH login details and specified config.*
φ (*DI*)	$\mathbb{P}$ (*EMI*)	*Holds the disk images*
φ (*Ad_Conf*)	$\mathbb{P}$(*CPU×Mem×Disk*)	*Holds admin configurations*
φ (*Hpvsr*)	$\mathbb{P}$ (*UCert×CPU×Mem×Disk×EMI×NIC×MAC×VID*)	*Hold configurations, creates virtual NIC, MAC, and VM ID*
φ (*DHCP*)	$\mathbb{P}$ (*IP×MAC*)	*Hold maps of IPs to MAC*
φ (*Phy_HW*)	$\mathbb{P}$ (*CPU×Mem×Disk ×NIC*)	*Holds physical specefication of the system*
φ (*VM*)	$\mathbb{P}$(*UCert×CPU×Mem×Disk×EMI×NIC×MAC×VID ×IP*)	*VM instance is created alongwith the specified config.*

transitions will be fired and the system will terminate. SSH protocol is used by $NimD$ to login to Co_Node. The SSH_Cert and SSH_Pass are confirmed using rules as stated in (16) and (17):

$$\begin{aligned}\boldsymbol{R}(SH\_S) = &\forall\, re \in Req \mid \exists SH[2] \in re[2]\\ &\wedge SH[3] \in re[3] \wedge Req'\\ &:= Req \cup \{(SH[1], SH[2], SH[3],\\ &SH[4], SH[5], SH[6])\},\end{aligned} \tag{16}$$

$$\begin{aligned}\boldsymbol{R}(SH\_F) = &\forall\, re \in Req \mid SH[2] \neq re[2]\\ &\wedge SH[3] \neq re[3]\\ &Req' := Req.\end{aligned} \tag{17}$$

At that point the configurations for memory, disk, and CPU provided by the user are matched with the administrator configurations. If (18) is fired, then the VM template disk image is copied from disk image repository using a distributed storage, such as S3, padded to the correct size, configured, and is pushed to the $Hpvsr$. If (19) is fired, then the request will be denied and the system will terminate,

$$\begin{aligned}\boldsymbol{R}(CS) = &\forall\, a \in AC \mid a[4] = R[1] \wedge a[5]\\ &= R[2] \wedge a[6]\\ &= R[3]\ \exists\, g \in GI, \wedge\, pp \in P\ P':\\ &= P \cup \{(R[1], R[4], R[5],\\ &R[6], g, pp[6], pp[7], pp[8])\},\end{aligned} \tag{18}$$

$$\begin{aligned}\boldsymbol{R}(CF) = &\forall\, a \in AC \mid a[4] \neq R[1] \wedge a[5]\\ &\neq R[2] \wedge a[6] \neq R[3]\\ &P' := P.\end{aligned} \tag{19}$$

In the last transition (20), an IP from $DHCP$ and virtual MAC and NIC from $Hpvsr$ are mapped one-to-one. Moreover, the physical NIC from Phy_HW is also mapped to virtual NIC and the relation between them is many-to-one, which means that many virtual $NICs$ can be mapped to one physical NIC. The VM is spawned after all the mappings are performed.

Nimbus has the ability to provide different resource leases to different users as a mean of scheduling. The flexibility and customization available in Nimbus makes it suitable for scientific community to perform experiment. Moreover, the workspace tools available in Nimbus can operate with Xen hypervisor and as well as with KVM,

$$\begin{aligned}\boldsymbol{R}(Con) = &\{\forall\, d[6] : De \mid \forall\, d[7] : De \mid\\ &\forall\, i[1] : RI \mid \forall\, pn[4] : RN,\\ &(i[1] \leftrightarrow d[7]) \leftrightarrow d[6]\}\\ &\wedge\ pn[4] \mapsto d[6]\\ RI' = &RI \cup \{(i[7], d[7])\}\\ De' = &De\ \cup\ \{(d[1], d[2], d[3], d[4],\\ &d[5], d[6], d[7], d[8])\}\\ RN' = &RN\\ SP' = &SP \cup\ \{(d[1], d[2], d[3], d[4],\\ &d[5], d[6], d[7], d[8], i[1])\}.\end{aligned} \tag{20}$$

3.5 OpenStack

The OpenStack is a cloud computing project, launched in July 2010, to provide IaaS. The OpenStack is mainly a collection of three open source projects: 1) OpenStack compute (Nova), to provide and manage large network of VMs, 2) OpenStack object storage (Swift), to provide redundant and scalable data storage using cluster of servers, and 3) OpenStack image service (Glance), to provide discovery, registration, and delivery service for disk images [60]. Similar to other systems, such as Eucalyptus and Open Nebula, the OpenStack supports EC2, S3, and rest interfaces. The networking between the OpenSTack and Eucalypstus also has certain similarities, such as the automatic bridge creation and IP forwarding for public IPs. Moreover, the authentication process, the development operations (DevOps) deployment tools, and hypervisors, are same between OpenStack and Eucalyptus. Despite the widespread popularity and adaptation of OpenStack, it is still in early stages and will need time to mature, as advocated in [29]. The formal analysis, modeling, and verification of the OpenStack are included as a future work in this paper.

4 Verification of Models Using SMT-Lib and Z3 Solver

Verification is the process of demonstrating the correctness of an underlying system. Two parameters are required to verify a model or a system: 1) specification and 2) properties. In this study, we use the bounded model checking [44] technique to perform the verification, using SMT-Lib and Z3 solver. In bounded model checking, the description of any system is verified, whether any of the acceptable inputs drives the system into a state where the system always terminates after finite number of steps. The process of bounded model checking involves several tasks: 1) *Specification*, the description of the system that states the properties or rules, which must be satisfied by the system to be deemed correct, 2) *Modeling*, representation of the system, and 3) *Verification*, use a tool to check whether the specifications has been satisfied by the model.

Definition 2 (Bounded Model Checking) [44]. *Formally, given a Kripke Structure $M = (S, S_0R, L)$ and a k bound, the bounded model checking problem is to find $\{M \models_k Ef\}$ where: S is the finite set of states, S_0 is a set of initial states, R is the set of transitions, such that $R \subseteq S \times S$, L is the set of labels.*

The bounded model checking problem is to find an execution path in M of length k that satisfies a formula f.

Kripke structure, which is a state transition graph, is used to represent the behavior of the system [45]. In Kripke structure nodes are the set of reachable states of the system, edge represents the transitions, and label functions map nodes to the set of properties hold in the state. Fig. 8 shows an example Kripke structure and computational tree where:

A path in a Kripke structure can be stated as an infinite sequence of states represented as $\rho = S_1, S_2, S_3, \ldots$ such that for $\forall i \geq 0, (S_i, S_{i+1}) \in R$. The model M may produce a path set $= S_1, S_2, S_1, S_2, S_3, S_3, S_3, \ldots$. To describe the property of a model some formal language, such as CTL*, CTL, or LTL is used. As stated in section 1, the "*" represents that CTL*

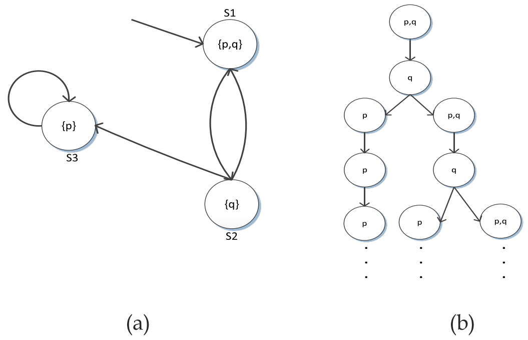

Fig. 8. An example of: (a) Kripke structure and (b) Computational Tree.

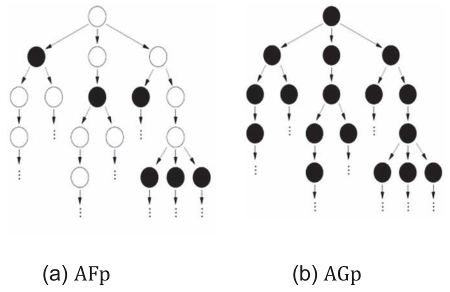

Fig. 9. An example CTL operators.

is a hybrid LTL and CTL. Some operators used in CTL* are shown in Table 7.

To demonstrate the use and meaning of operators an example is provided in Fig. 9. The black circle in Fig. 10 represents a state p. Moreover, $\mathbf{AF}p$ in Fig. 9a means, that a future state p is reachable from every path. Furthermore, $\mathbf{AG}p$ in Fig. 9b means, that state p is globally reachable from every path. (Readers are encouraged to see [46] for more details about the CTL*.)

For a model to be correct, the states must satisfy the formulas (Definition 2) under a specific bound. The formulas are represented in terms of properties of the systems.

Definition 3 (SMT Solver) [31]. *Given a theory* $\bullet$ *and a formula* f*, the SMT solvers perform a check whether* f *satisfies* $\bullet$ *or not.*

To perform the verification of the models using Z3 (an SMT Solver), we unroll the model M and the formula f that provides M_k and f_k, respectively. Moreover, the said parameters are then passed to Z3 to check if $M_k \models_\Gamma f_k$ [56]. The solver will perform the verification and provide the results as satisfiable (*sat*) or unsatisfiable (*unsat*). If the answer is *sat*, then the solver will generate a counter example, which depicts the violation of the property or formula f. Moreover, if the answer is *unsat*, then formula or the property f holds in M up to the bound k (in our case k is exec. time).

For the models to be correct, the solver should be able to find a terminating state in a model. The failure transitions in all of the models are considered as a terminator of the models. Moreover, the other terminating state in the models is the last state when the instance is successfully created. One property to verify the models is that, *whenever a request is made for a VM and there are no failures, then the instance should be created*. To explain the verification of the models, an example, computational tree of a model in Fig. 3 is developed and shown in Fig. 10. The tree is drawn by following the success transitions only. If we closely analyze, we can see that $VM - Run$ (the final terminating) state is reachable from every path in the tree, which shows that the model terminates after some iterations. Therefore, satisfies the property. In Fig. 10, the state $VM - Run$ at second level shows a scenario when the instance is already been created and user can directly SSH the instance. The states labeled with *"x"* represents, that from the point forward the tree will repeat the predecessor. Similar process has been followed to verify all the models of the systems in this study. We have specified the properties of the VM-based systems in a similar passion and verified whether the properties are satisfied by the models. The properties we have verified for our models are:

1. if a request is made and there are no failures, then the instance of VM must be created,
2. the instance of VM must have the same configurations as requested by the user,
3. the instance should be distinct.

TABLE 7
Operators (Op) Used in CTL* and Description (Desc)

Op	Desc	Op	Desc
A	For all paths	$\wedge$	Logical AND
E	For some paths	$\vee$	Logical OR
X	Next	$\neg$	Negation
F	For future paths	$\rightarrow$	Implication
G	Globally	$\leftrightarrow$	Double implication

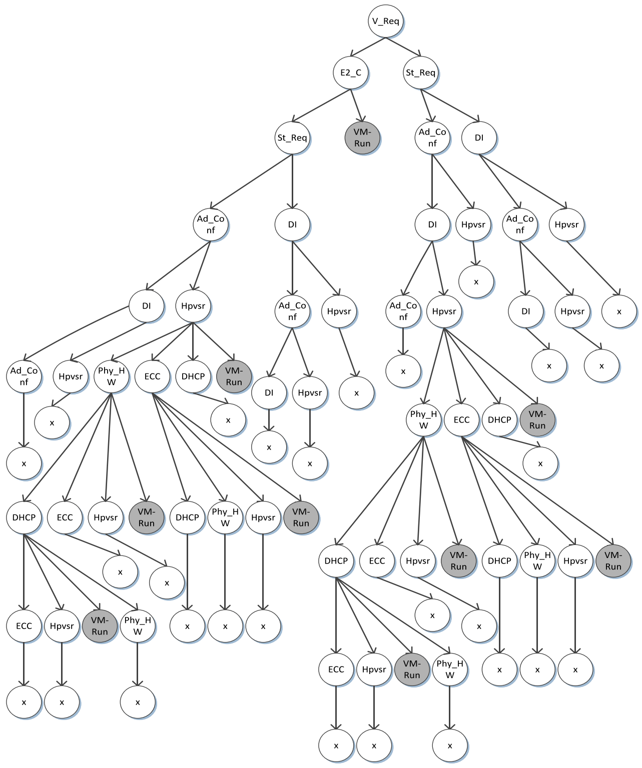

Fig. 10. An example computational tree of Eucalyptus.

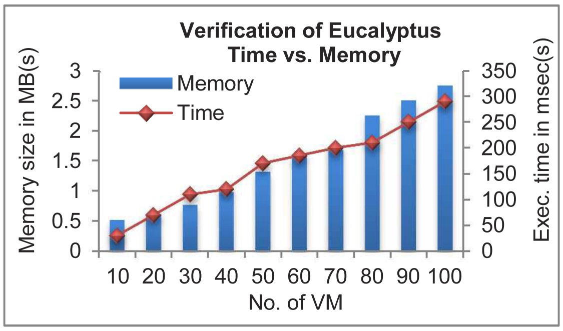

Fig. 11. Verification results of Eucalyptus.

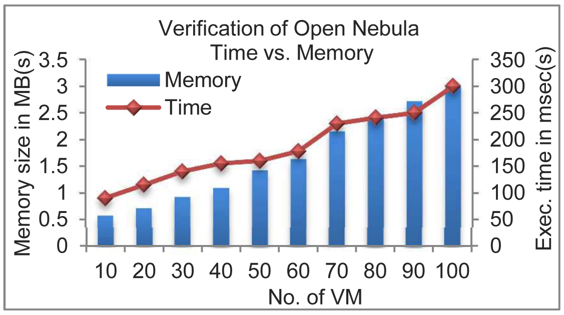

Fig. 12. Verification results of Open Nebula.

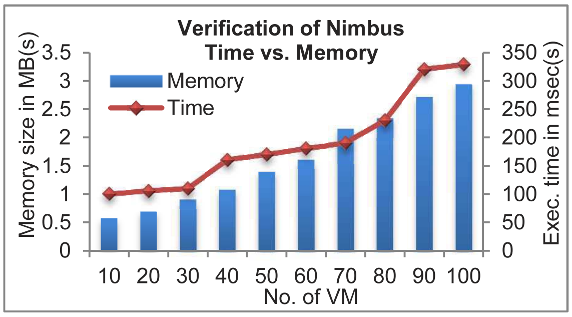

Fig. 13. Verification results of Nimbus.

Results. To verify, the models of the VM-based systems are translated to SMT. Moreover, the properties are also translated and specified in SMT. The model along with the properties are given to the Z3 solver to check either the model satisfies the properties or not. If there are no errors, then the model specifications can be stated as correct. Note that our goal in this section is to verify the correctness of the models and not to measure or analyze the performance of the systems. Fig. 11 depicts the execution time and memory taken to verify the model of Eucalyptus. We have instantiated 100 VMs to verify the properties stated above and to observe the effect of scalability on the working of the models.

The model of Eucalyptus works fine and produces results as expected. The results in Fig. 11 shows that an increasing trend is followed in both, the execution time and memory, as the number of VMs increases. The increase in the values is obvious, as the number of VMs will increase more time will be required by the processor to verify and more space will be needed to store the variables.

Figs. 12 and 13 depict the verification results for the model of Open Nebula and Nimbus, respectively. The execution time and memory increase in both of the models as the number of VMs increases. The increase in the values has the same reason that is stated in the results of Eucalyptus.

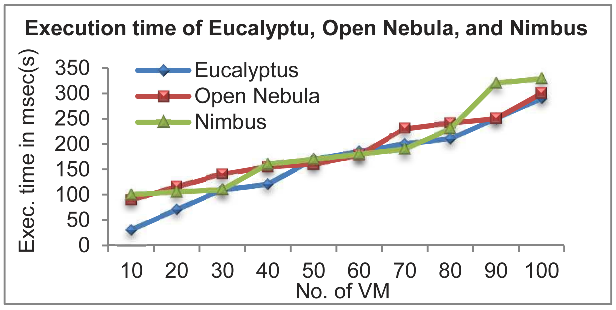

Fig. 14. Execution time comparison of Eucalyptus, Open Nebula, and Nimbus.

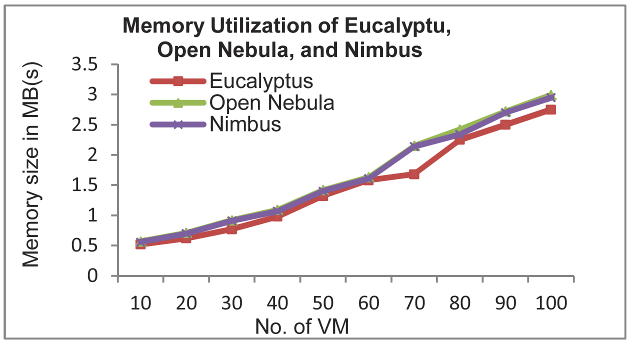

Fig. 15. Memory utilization of the systems.

Figs. 14 and 15 plot the execution time of all the models (Eucalyptus, Open Nebula, and Nimbus) and memory consumption, respectively. As seen in Fig. 15, the memory consumption of Open Nebula and Nimbus has almost similar values, which is due to the fact that Eucalypstus keeps the records of environment variable and configurations. Note that the results indicate the time taken by Z3 solver to verify the models based on the specified properties. The results in no manner characterize the performance of the models or the systems. Moreover, the goal is to demonstrate the correctness of the models and to highlight the feasibility of the models with respect to scalability and execution time.

5 Related Work

Virtualization has been studied extensively in the domain of cloud computing. The research is usually focused toward the security or resource provisioning of VM. In [3], to make the VM more secure, the authors have proposed to remove the virtualization layer, while retaining the key features enabled by the cloud. In [48], the authors proposed HyperSafe, which is a lightweight approach that endows existing hypervisors with a self-protection capability to provide lifetime flow integrity. In [49], authors proposed new security architecture in a hypervisor-based virtualization technology to secure the cloud environment. Similarly, [47], also discussed security aspects of VMs in the cloud. In [43], a generic model is proposed for resource allocation of VMs in multitier distributed environment that describes every VM as a multidimensional vector. In [37], a two level resource manager is proposed that allocate resources to individual containers using local controllers. Several studies are also available, such as [16], [17], [18], [19] that discuss and compare the cloud management platforms. The focus of the said studies was on the discussion and comparison of architecture and feature set of the systems. However, little amount of work has been done in the area of modeling and analysis of cloud management systems, specifically VM-based systems. HLPN has been widely used for the modeling of systems from various domains of computer science, such as cloud computing, web service framework, grid infrastructures, scheduling, and load balancing. In [33], the authors have used colored Petri nets (CPN) to model the Open Pmodel-based diagnosisrovenance Model (OPM) for the purpose of model-based diagnosis in the cloud (MBD). Virtualization and modeling (including formal analysis and verification) are very rich research domains, when considered separately. However, a diminutive amount of research is performed when both (modeling, including formal analysis and verification of virtualization) are combined.

6 Conclusions and Future Work

In this paper, we have studied and analyzed three state-of-the-art VM-based open source cloud management platforms: 1) Eucalyptus, 2) Open Nebula, and 3) Nimbus. To model the systems with the advantage of providing a firm mathematical representation and to analyze the structural and behavioral properties, we used HLPN. Moreover, the models are verified using SMT-Lib and Z3 solver. We used the model checking approach to verify our models. Several properties are specified (using the specification and details available in documentation) and if the models satisfy those properties, then the model is declared correct. The verification results shown in the paper indicate that the models are correct and feasible as the numbers of VM grow. This paper provides an in-depth formal analysis, modeling, and verification of the systems that will be helpful for the research community to further explore and understand the systems. Moreover, the paper also provides a strong foundation for new researchers to apprehend the meticulous knowledge of the systems. In future, we will analyze, model, and verify some other cloud management platforms, such as OpenStack, oVirt, and ECP. Moreover, we will also perform a detailed feasibility analysis of the aforesaid platforms under different SLA constraints.

References

[1] R. Buyya, S.Y. Chee, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities," *Proc. IEEE 10th Int'l Conf. High Performance Computing and Comm. (HPCC '08),* pp. 5-13, Sept. 2008.

[2] P. Mell and T. Grance, "Definition of Cloud Computing," technical report, NIST, 2009.

[3] E. Keller, J. Szefer, J. Rexford, and R.B. Lee, "NoHype: Virtualized Cloud Infrastructure without the Virtualization," *Proc. ACM 37th Ann. Int'l Symp. Computer Architecture (ISCA '10),* pp. 350-361, June 2010.

[4] M. Eisen Marcum Technology, Introduction to Virtualization, "The Long Island," *Chapter of the IEEE Circuits and Systems (CAS) Soc.,* Apr. 2011.

[5] A. Vichos, "Agent-Based Management of Virtual Machines for Cloud Infrastructure," Master's thesis, School of Informatics, Univ. of Edinburgh, 2011.

[6] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster, "Capacity Leasing in Cloud Systems Using the OpenNebula Engine," *Proc. Workshop Cloud Computing and its Applications (CCA '08),* Oct. 2008.

[7] *Amazon Elastic Compute Cloud (Amazon EC2),* http://aws.amazon.com/ec2/, 2013.

[8] *Google App Engine,* https://developers.google.com/appengine/, 2013.

[9] *Science Clouds,* http://scienceclouds.org/, 2013.

[10] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud Computing System," *Proc. Ninth IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '09),* pp. 124-131, May 2009.

[11] *oVirt,* http://www.ovirt.org/Home, 2013.

[12] *Enomaly,* http://www.enomaly.com/, 2013.

[13] F. Panzieri, O. Babaoglu, S. Ferretti, V. Ghini, and M. Marzolla, "Distributed Computing in the 21st Century: Some Aspects of Cloud Computing," *Technology-Enhanced Systems and Tools for Collaborative Learning Scaffolding,* pp. 393-412, Springer, 2011.

[14] R. Wojtczuk, "Subverting the Xen Hypervisor," *Black Hat USA,* 2008.

[15] I. Habib, "Virtualization with kVM," *Linux Journal,* article 8, no. 166, 2008.

[16] D. Cerbelaud, S. Garg, and J. Huylebroeck, "Opening the Clouds: Qualitative Overview of the State-of-the-Art Open Source Vm-Based Cloud Management Platforms," *Proc. 10th ACM/IFIP Int'l Conf. Middleware,* pp. 1-8, 2009.

[17] P.T. Endo, G.E. Gonçalves, J. Kelner, and D. Sadok, "A Survey on Open-Source Cloud Computing Solutions," *Proc. Eighth Workshop Cloud and Grid Applications,* pp. 3-16, 2010.

[18] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing,* vol. 13, no. 5, pp. 14-22, Oct. 2009.

[19] N. Khan, A. Noraziah, E.I. Ismail, and M.M. Deris, "Cloud Computing: Analysis of Various Platforms," *J. Entrepreneurship and Innovation,* vol. 3, no. 2, pp. 51-59, 2012.

[20] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proc. IEEE,* vol. 77, no. 4, pp. 541-580, Apr. 1989.

[21] *Lectures on Petri Nets I: Basic Models,* W. Reisig and G. Rozenberg, eds., Springer-Verlag, 1998.

[22] S.K. Garg, S. Versteeg, and R. Buyya, "A Framework for Ranking of Cloud Computing Services," *Future Generation Computer Systems,* vol. 29, no. 4, pp. 1012-1023, 2013.

[23] L. Moura and N. Bjrner, "Satisfiability Modulo Theories: An Appetizer," *Formal Methods: Foundations and Applications* , pp. 23-36, Springer, 2009.

[24] B. Javadi, R. Thulasiram, and R. Buyya, "Characterizing Spot Price Dynamics in Public Cloud Environments," *Future Generation Computer Systems,* vol. 29, no. 4, pp. 988-999, 2013.

[25] *OpenStack,* http://www.openstack.org/downloads/openstack-overview-datasheet.pdf, 2013.

[26] M. Frade and J.S. Pinto, "Verification Conditions for Source-Level Imperative Programs," Technical Report DI-CCTC-08-01, Univ. of Minho, 2008.

[27] SMT-Lib http://smtlib.cs.uiowa.edu/, 2013.

[28] S.U.R. Malik, S.K. Srinivasan, S.U. Khan, and L. Wang, "A Methodology for OSPF Routing Protocol Verification," *Proc. 12th Int'l Conf. Scalable Computing and. Comm. (ScalCom '12),* Dec. 2012.

[29] G. von Laszewski, J. Diaz, F. Wang, and G.C. Fox, "Comparison of Multiple Cloud Frameworks," *Proc. IEEE Conf. Cloud Computing,* pp. 734-741, 2012.

[30] L. de Moura and N. Bjorner, "Z3: An Efficient SMT Solver," *Proc. Theory and practice of software, 14th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS '08),* 2008.

[31] L. Cordeiro, B. Fischer, and J. Marques-Silva, "SMT-Based Bounded Model Checking for Embedded ANSI-C Software," *Proc. IEEE/ACM Int'l Conf. Automated Software Eng. (ASE '09),* pp. 137-148, 2009.

[32] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *Computer,* vol. 38, no. 5, pp. 39-47, May 2005.

[33] Y. Li and O. Boucelma, "A CPN Provenance Model of Workflow: Towards Diagnosis in the Cloud," *Proc. Conf. Advances in Databases and Information Systems,* pp. 55-64, 2011.

[34] P. Sempolinski and D. Thain, "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus," *Proc. IEEE Second Int'l Conf. Cloud Computing Technology and Science (CloudCom '10),* pp. 417-426, 2010.

[35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," *Proc. IEEE Int'l Symp. Cluster Computing and the Grid,* pp. 124-131, 2009.
[36] *Eucalyptus Systems Home Page,* http://www.eucalyptus.com, 2013.
[37] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy Logic-Based Approaches," *J. Cluster Computing,* vol. 11, pp. 213-227, 2008.
[38] D. Milojicic, I.M. Llorente, and R.S. Montero, "OpenNebula: A Cloud Management Tool," *IEEE Internet Computing,* vol. 15, no. 2, pp. 11-14, Mar./Apr. 2011.
[39] *Open Nebula,* http://www.opennebula.org, 2013.
[40] *Nimbus,* http://www.nimbusproject.org/docs/2.10, 2013.
[41] I. Foster and C. Kesselman, "The Globus Project: A Status Report," *Proc. IEEE Heterogeneous Computing Workshop,* pp. 4-18, 1998.
[42] *A Survey of Open-Source Cloud Infrastructure Using FutureGrid Testbed,* T. Wu, S.N. Baasha, and S.S. Karwa, http://salsahpc.indiana.edu/b649proj/proj7.html, 2013.
[43] P. Campegiani and F.L. Presti, "A General Model for Virtual Machines Resources Allocation in Multi-Tier Distributed Systems," *Proc. Int'l Conf. Autonomic and Autonomous Systems,* pp. 162-167, 2009.
[44] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded Model Checking," *Advances in Computers,* vol. 58, Academic press, 2003.
[45] Y.M. Quemener and T. Jeron, "Model Checking of CL on Infinite Kripke Structures Defined by Simple Grammers," Technical Report RR-2563, 1995.
[46] M. Maidl, "The Common Fragment of CTL and LTL," *Proc. IEEE Symp. Foundations of Computer Science,* 2000.
[47] F. Zhang, J. Chen, H. Chen, and B. Zang, "CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization," *Proc. 23rd ACM Symp. Operating Systems Principles (SOSP '11),* Oct. 2011.
[48] Z. Wang and X. Jiang, "HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity," *Proc. IEEE Symp. Security and Privacy (SP '10),* pp. 380-395, 2010.
[49] F. Sabahi, "Secure Virtualization for Cloud Environment Using Hypervisor-Based Technology," *Int'l J. Machine Learning and Computing,* vol. 2, no. 1, pp. 39-45, Feb. 2012.
[50] S.U.R. Malik, S.K. Srinivasan, and S.U. Khan, "Convergence Time Analysis of Open Shortest Path First Routing Protocol in Internet Scale Networks," *IET Electronics Letters,* vol. 48, no. 19, pp. 1188-1190, 2012.
[51] J. Kolodiej, S.U. Khan, E. Gelenbe, and E.-G. Talbi, "Scalable Optimization in Grid, Cloud, and Intelligent Network Computing," *Concurrency and Computation: Practice and Experience,* vol. 25, no. 12, pp. 1719-1721, 2013.

Saif U. R. Malik received the MS degree from the COMSATS Institute of Information Technology, Islamabad, Pakistan in 2009. He is working towards his PhD degree at North Dakota State University, Fargo, ND. His primary research interests include formal verification, modeling, cyber physical systems, and cloud computing systems. Other areas of research interest include data replication, routing protocols, and HPC.

Samee U. Khan (S'02-M'07-SM'12) received the BS degree from the Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi, Pakistan, and the PhD degree from the University of Texas, Arlington, TX. Currently, he is assistant professor of electrical and computer engineering at the North Dakota State University, Fargo, ND. His research interests include optimization, robustness, and security of: cloud, grid, cluster, and big data computing, social networks, wired and wireless networks, power systems, smart grids, and optical networks. His work has appeared in over 200 publications. He is a fellow of the Institution of Engineering and Technology (IET, formerly IEE), and a fellow of the British Computer Society (BCS). He is a senior member of the IEEE.

Sudarshan K. Srinivasan (M'07) received the BE degree in electrical and electronics engineering from the University of Madras, Chennai, India, in 2001, and the MS and PhD degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2003 and 2007, respectively. He is currently an associate professor at the Department of Electrical and Computer Engineering, North Dakota State University, Fargo. His primary research interests include formal verification. His other research interests include computer architecture and electronic design automation. He is a member of the IEEE.