Received: 9 June 2020 | Revised: 15 November 2020 | Accepted: 15 December 2020

DOI: 10.1002/spe.2953

RESEARCH ARTICLE

WILEY

Availability modeling in redundant OpenStack private clouds

Mahsa Faraji Shoyari[1] | **Ehsan Ataie**[2] | **Reza Entezari-Maleki**[3,4] | **Ali Movaghar**[1]

[1]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

[2]Department of Technology and Engineering, University of Mazandaran, Babolsar, Iran

[3]School of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

[4]School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

Correspondence
Ehsan Ataie, Department of Technology and Engineering, University of Mazandaran, Babolsar, Iran.
Email: ataie@umz.ac.ir

Abstract

In cloud computing services, high availability is one of the quality of service requirements which is necessary to maintain customer confidence. High availability systems can be built by applying redundant nodes and multiple clusters in order to cope with software and hardware failures. Due to cloud computing complexity, dependability analysis of the cloud may require combining state-based and nonstate-based modeling techniques. This article proposes a hierarchical model combining reliability block diagrams and continuous time Markov chains to evaluate the availability of OpenStack private clouds, by considering different scenarios. The steady-state availability, downtime, and cost are used as measures to compare different scenarios studied in the article. The heterogeneous workloads are considered in the proposed models by varying the number of CPUs requested by each customer. Both hardware and software failure rates of OpenStack components used in the model are collected via setting up a real OpenStack environment applying redundancy techniques. Results obtained from the proposed models emphasize the positive impact of redundancy on availability and downtime. Considering the tradeoff between availability and cost, system providers can choose an appropriate scenario for a specific purpose.

KEYWORDS

availability modeling, cloud computing, continuous time Markov chains, hierarchical models, reliability block diagrams

1 | INTRODUCTION

Cloud computing includes different resources of one or more data centers, such as storage, network and compute, shared through virtualization technology.[1] Infrastructure as a service (IaaS) is an important service model of the cloud computing paradigm in which providers allocate virtual computing resources as virtual machines (VMs) on physical machines (PMs) to customers over the internet.[2-4] According to Gartner,[5] the market for public cloud will reach over $306.9 billion at the end of 2021 and $364 billion at the end of 2022. Gartner expects the hyperscale providers, such as Amazon AWS, Google Cloud Platform, Alibaba, and Microsoft Azure, to account for 70% of the market by the end of 2021.[6] Large organization and government agencies usually like to develop their own cloud data centers for satisfying their needs. By contrast, small organizations usually rent cloud resources from cloud providers to solve their own problems.[7]

The reason to choose IaaS cloud services by costumers is their agility, scalability, business growth, cost benefits, and innovation. Despite the opportunities and benefits, the challenges and risks are still outnumbered.[8-10] As an important challenge, short downtime of IaaS cloud services has negative effects on different subjects ranging from economic losses to impact on human life.[11] It is reported that the data center components outage can cost the providers between \$8000 and \$16,000 per minute.[12] According to the results of the report conducted by Ponemon Institute, the average cost of a data center outage has steadily increased from \$505,502 in 2010 to \$740,357 in 2016, indicating a 38% increase.[13] According to a report by Information Technology Intelligence Consulting in 2018, an hour of downtime on average costs data center operators \$260,000, while a 5-min outage costs \$2600.[14] Since IaaS cloud service providers use data centers to house cloud services and VMs, it is essential to ensure the availability of services. The availability of services is important and cloud services should be accessible from anywhere and at anytime.[15-17] The availability may be affected by software and hardware failures.[18] A wide range of fault tolerance approaches exists for coping with software and hardware failures. One of the paramount approaches to ensure high availability is applying the redundancy technique.[17]

In this article, we use redundant nodes and multiple clusters for each subsystem of OpenStack[19] to identify the most important node in OpenStack architecture for improving the system availability. In order to analyze and compare the availability and downtime of nonredundant and redundant architectures of OpenStack clouds, hierarchical heterogeneous models as hybrid models, with the combination of reliability block diagrams (RBDs)[20] and continuous time Markov chains (CTMCs),[21] are presented in this article. Moreover, the cost of each architecture based on the acquisition cost of the computing resources is computed which helps to simultaneously analyze the availability and the cost to achieve that amount of availability. Jointly analyzing availability and cost, an organization can choose the best suitable architecture according to its needs. In cloud data centers, the capacity of PMs and the type of resources requested by each user are usually different. If the capacity of PMs is over, the user request will be rejected. Thus, not only the availability of individual cloud components but also the capacity of PMs affects the availability of the cloud data center. Hence, the heterogeneity of VMs is important in real IaaS clouds and it should be considered in a comprehensive modeling of such systems. The models proposed in this article take into account the number of CPUs among other attributes, such as memory and hard disk, which may differ in various customer jobs. Since failure/repair behavior of hardware and software is important and it can directly affect the availability, we use a monitoring tool to measure the probability of OpenStack components working correctly and the repair rate of each OpenStack component. These parameters are used in the proposed models.

The main contribution of this article is to propose a hierarchical model for IaaS clouds, based on OpenStack, that considers the details of such systems and also the relationship between their components. The hierarchical model combines state space and nonstate space modeling techniques in synergy to model the interconnections and the internals of OpenStack cloud components. In order to attain an accurate evaluation of the proposed model, we measure, in a real environment, the probability of correctly working OpenStack components and the repair rate of each OpenStack component using a monitoring tool. Moreover, we consider the heterogeneous workloads that have a direct effect on the availability of cloud data centers. Furthermore, different scenarios are introduced and analyzed to identify the important subsystems in OpenStack architecture with respect to availability.

The remainder of this article is organized as follows. A literature review is provided in Section 2. Section 3 describes the nonredundant and redundant architectures of the OpenStack studied in this article. Section 4 introduces the availability models proposed to evaluate different nonredundant and redundant scenarios. Section 5 presents the numerical results obtained by solving the proposed analytical models. Finally, Section 6 draws the final remarks and outlines some future research directions.

2 | RELATED WORK

Quality of service (QoS) of cloud computing systems is considered as an important factor in three-layered architectural requirements, namely, provider, enterprise and user requirements. High availability is one of the challenging aspects of QoS in cloud services.[22] The most common method to provide high availability is redundancy in hardware, software, and network components.[23] Availability analysis methods are classified into three main categories including model-based, measurement-based, and simulation-based analysis.[24] However, most of the articles studied in this section are in the model-based category. In the following, we divide related state-of-the-art into two groups. The first group consists of studies that combine two or more modeling techniques, and the second one includes those that use only one technique for modeling.

2.1 | Hybrid methods

In this section, we present related studies that use a combination of modeling methods such as nonstate space and state space techniques. Dantas et al.[25] have used RBDs and Markov reward models to propose a hierarchical heterogeneous model for cloud systems, which applies the redundancy technique. The proposed hierarchical heterogeneous model was solved, with closed-form equations, to evaluate the availability in nonredundant and redundant Eucalyptus architectures. Differential sensitivity analyses were applied to identify bottlenecks and determine which components are critical to improve availability. However, the models analyzed in Reference 25 consider neither the details of the relationships between the components of Eucalyptus nor the failures occur during the Cloud controller working process. In addition, the effect of different workloads on the availability was not addressed in Reference 25. The same authors[17] have extended their work by considering redundant components and multiple clusters not only for improving the availability but also for increasing the computational capacity. The drawback of such improvement in operational cost was also analyzed. Models with a separate number of clusters were evaluated based on metrics for the capacity-oriented availability and the steady-state availability. The authors also considered the cost of architectures, implemented with Eucalyptus, and compared them with a similar infrastructure rented from a public cloud provider. Like the work presented in Reference 25, the details of the interaction between cloud components and the impact of the failure of each component on the delivery of services were not considered in Reference 17.

Torres et al.[26] have proposed a hierarchical model which uses CTMCs, RBDs, and stochastic petri nets (SPNs) to evaluate availability and performance-related metrics for private cloud storage services. The result of this study shows that the application of redundancy technique reduces the probability of downtime and increases the availability leading to increased system utilization. Furthermore, the performance of the proposed model was validated by implementing the real cloud storage service. In contrast to our work, heterogeneous requests were not considered in Reference 26.

Wei et al.[20] have used hierarchical models combining RBDs and generalized stochastic petri nets for evaluating reliability and availability of virtual data centers (VDCs) in cloud computing. They analyzed the impact of consolidation ratio and workload on dependability attributes and derived some useful rules for designing more dependable VDCs. They stated that the consolidation ratio is an important parameter for designing more dependable VDCs. The detailed process and interactions of each cloud subsystem were not considered in Reference 20, and the proposed analytical models are specific to modeling VDCs.

Matos et al.[27] have used composite modeling techniques to evaluate the availability of mobile cloud computing. They represented the mobile cloud architecture using RBDs and applied CTMCs for describing details of each component. Afterward, they provided a sensitivity analysis for mobile cloud availability evaluation, and investigated the impact of each component. The results of the sensitivity analysis showed which components in the mobile cloud architecture are more important in increasing availability. However, the detailed internal process of each cloud subsystem was not considered.

Matos et al.[28] have also proposed a hierarchical modeling approach for availability evaluation of redundant Eucalyptus private clouds by combining RBDs and Markov chains. They performed differential sensitivity analysis on models to identify the most important component of the system with respect to the availability. The results showed that the failure and repair rates of Cloud Manager component have the highest impact on the steady-state availability, and therefore, using the warm-standby technique on this component can improve the availability. Even though the approach proposed in Reference 28 is interesting, their work does not consider the detailed process and problems that may occur in each component of a private cloud which have a direct impact on availability and downtime.

Applying hybrid method, modeling of complex systems, for example, cloud infrastructures, becomes easy, and state space explosion problem can be handled. Compared with the approaches mentioned in this section, we consider a more detailed hybrid model in this article that comprises both state space and nonstate space techniques. Each interaction of components in the cloud environment and the failure of each of them has a significant impact on the system's availability. In our proposed hierarchical approach, we consider the detailed internal process, including interactions between cloud subsystem and the related problems during service delivery.

2.2 | Single methods

In this section, some related state-of-the-art that use one modeling technique are introduced. Ataie et al.[29] have proposed analytical models based on stochastic reward nets (SRNs) to model and evaluate performance, availability and power consumption of an IaaS cloud at different levels. The SRN was used, in a hierarchical manner, to propose a monolithic model for an entire IaaS cloud, and then the folding and fixed-point iteration techniques were used to address scalability problem of the monolithic model.[29] Compared with the models presented in this article, VM requests have been assumed to be homogeneous in Reference 29.

Liu et al.[30] have presented an analytical approach to investigate the effective factors on IaaS cloud availability such as repair policy. In order to increase the availability of PMs, the redundancy technique was used. The authors applied monolithic and scalable SRNs with different repair policies and architectures to evaluate cloud availability. Results reported in Reference 30 show that different repair policies provide almost the same level of availability but with different costs. In contrast to the approach proposed in Reference 30, in our presented model, different types of service failures are investigated.

Jammal et al.[31] have proposed an SPN model to evaluate the availability of cloud applications deployments. They have proposed the cloud scoring system to select the optimal high availability-aware deployment in terms of energy and operational expenditure. The authors considered different stochastic failures, deterministic repairs, functionality constraints, redundancy, and interdependencies between different application components. Although the cost and energy are important factors in processing user requirements, they should not be the sole studied criteria.[30] Heterogeneous workloads, as a realistic user requirements, should be considered in cloud computing, which was not addressed in Reference 31.

Chuob et al.[32] have proposed private clouds for an e-government data center. They evaluated availability of the cloud architecture with a hierarchical availability model in three levels. Cluster and node levels were evaluated with Markov chains and the operational level was evaluated using an informal method. In order to provide more accuracy in evaluation, an experimental environment based on Eucalyptus was developed and the failure and recovery rates of cloud components were computed.

Ghosh et al.[33] have considered a large IaaS cloud where the PMs of the infrastructure understudy were grouped into three pools with different power consumptions and provisioning delays. They proposed two different models based on SRNs for availability analysis. The first model was proposed in a monolithic way that is not proper for evaluating availability of large-scale systems while the second model consists of interacting submodels facilitating modeling a large system. They showed that the interacting submodels are a better solution for modeling large-scale systems. Unlike our approach presented herein, the authors in Reference 33 have not used real experiments to measure input parameters of their proposed models.

Wang et al.[34] have proposed a thermal-aware VM consolidation mechanism that considers both host power consumption and temperature to obtain reasonable trade-offs between energy-saving and SLA violations. The variability in temperature is considered as a migration criterion for outage avoidance. The temperature of each host is obtained from the Markov model to predict future CPU usages of physical hosts and VMs to reduce the number of migrations needed in the long run.

It is not effective to use monolithic models for representing complex systems such as clouds. In contrast to the approaches mentioned above, we propose a more detailed heterogeneous hierarchical model in this article. Furthermore, the probability of OpenStack components working correctly and the repair rate of different OpenStack components are measured experimentally to provide more accurate results. The heterogeneity of user requests is also considered in the proposed approach to model a more realistic IaaS cloud system.

3 | SYSTEM ARCHITECTURE

OpenStack is a free and open source software for creating public and private IaaS clouds that manage large pools of resources such as compute, storage, and networking through data centers. The platform consists of several individual projects and components, each of them performs a specific function within the cloud ecosystem.[35] In general, OpenStack has many components, six of them are mandatory for providing VMs. The six core components are Keystone, Glance, Nova, Neutron, Cinder, and Horizon, briefly described in the following.

- Keystone is an identity component. It acts as an authentication system across the cloud. It is the first component that is installed by OpenStack, and other OpenStack components use Keystone among their operations.
- Glance is an image component, which manages and maintains the image of VMs in OpenStack and uploads OpenStack compatible images.

- Nova is a compute component and one of the most important parts of OpenStack. It manages and controls the process of creating OpenStack VM instances, such as the size and location of an instance.
- Neutron is a networking component and one of the complicated components of OpenStack. It manages and controls the networking aspects of instances, such as virtual networks, subnets, and router.
- Cinder is a block storage component. It virtualizes pools of block storage devices, and thereby, users can request and consume those resources without requiring any knowledge of where or on what type of device that storage is actually deployed.
- Horizon is a dashboard component. It manages all other OpenStack components through self-service API, and helps the user to work with OpenStack services via a web interface.

OpenStack components use a database and message broker in order to share the status of cloud components and communicate with each other. MySQL, as a database, and RabbitMQ, as a message broker, are used by OpenStack components. They are also considered as basic components, besides other components of OpenStack.

Figure 1 shows an OpenStack architecture in a nonredundant and basic environment. This architecture has three main nodes, including the Controller node, Compute node, and Storage node. All components depicted in Figure 1 should work properly to create a successful VM. The Controller node consists of Horizon, Nova, Glance, Neutron, Keystone, RabbitMQ, and Database components. The Compute node consists of Nova, Neutron, and KVM components. The Storage node consists of Cinder component. Figure 2 shows an OpenStack architecture in a redundant environment. This architecture has one or more Controller, Compute, and Storage node.

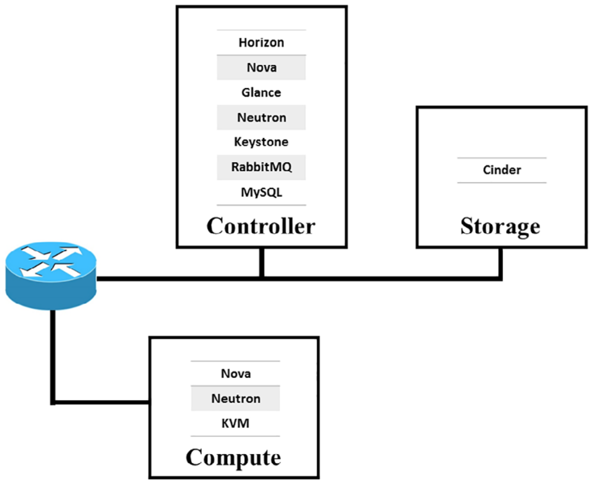

FIGURE 1 The nonredundant OpenStack architecture [Color figure can be viewed at wileyonlinelibrary.com]

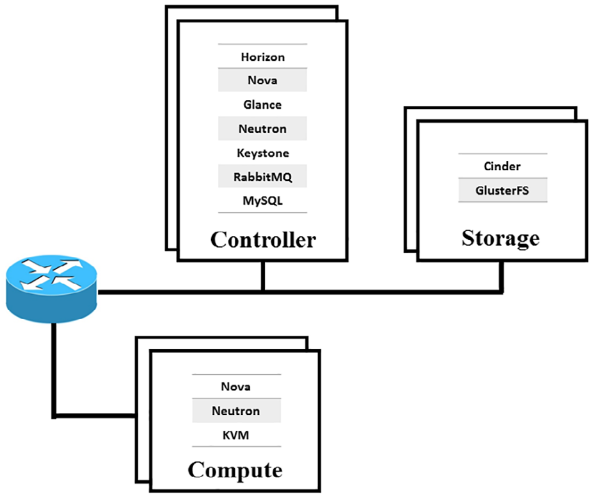

FIGURE 2 The redundant OpenStack architecture [Color figure can be viewed at wileyonlinelibrary.com]

4 | PROPOSED MODELS

Nonstate space and state space models can be hierarchically combined to cope with the complexity of representing large-scale systems and the problem of state space explosion. Modeling distributed systems with hierarchical hybrid models, better representation of dependencies among components of such systems can be achieved. Due to the modular architecture of OpenStack and the dependency of its nodes and components, RBDs are applied in this article to represent the first level of OpenStack architecture, including the relation between the Controller, Compute, and Storage nodes. RBDs are used in this level to analyze the steady-state availability of the architecture. For describing the relationships among OpenStack components and modeling the redundancy mechanism, CTMCs are applied. These CTMCs represent the second level of OpenStack architecture, including the internal relations among the components of the Controller, Compute, and Storage nodes. Thus, a hierarchical heterogeneous model, composed of RBDs and CTMCs, are adopted herein. We summarize the nomenclature used throughout this article in Tables 1 and 2. The nomenclature of the states of the CTMCs modeled in this article is represented in Table 1. Table 2 shows the nomenclature of the blocks modeled in the RBDs introduced in this study.

4.1 | Scenarios

In this work, nine different scenarios are introduced, and the proposed hierarchical hybrid model is applied to analyze them. While Scenario 1 models non-high availability architecture of OpenStack, Scenarios 2–9 model the high availability architectures of OpenStack by applying the redundancy technique. In Scenario 2–7, to determine the critical node of the

TABLE 1 Nomenclature of continuous time Markov chains

Symbol	Meaning
H	Horizon State—The Controller node is running
K	Keystone State—Keystone service is running
N	Nova State—Nova service is running
G	Glance State—Glance service is running
NT	Neutron State—Neutron service is running
F	The Controller node failed
D	The Controller node failure is detected
P	The Controller node failure is not covered, additional recovery step is started
m	The maximum CPUs inside a PM
MC	The maximum number of CPUs requested by a customer $MC \leq m$
(i,j)	"i" is the number of consumed CPUs of the host PM and "j" is the number of VMs running on those CPUs
FU	The first Controller node failed and the second Controller node is running
DU	The failure of the first Controller node is detected
H'	The second Controller node is responsible for receiving a new request
K'	Keystone State—Keystone service is running
N'	Nova State—Nova service is running
G'	Glance State—Glance service is running
NT'	Neutron State—Neutron service is running
$H'2$	The failure of first Controller node is not covered; additional recovery step is started
FF	The second Controller node failed before repairing the first Controller node
DD	The failure of both Controller nodes is detected
PP	The failure of both Controller nodes is not covered; additional recovery step is started

TABLE 2 Nomenclature of reliability block diagrams

Symbol	Meaning
Controller	The OpenStack Controller node
Compute	The OpenStack Compute node
Storage	The Openstack Storage node
HW	The Hardware component
OS	The Operating System component
MySQL	The Database component
Keystone	The Identity component
RabbitMQ	The Message Broker component
Nova	The Computing component
Horizon	The Dashboard component
Glance	The Image component
Neutron	The Network component
KVM	The Hypervisor component
Cinder	The Storage component
GlusterFS	The Software Network File system component

whole system with respect to the availability, only the redundancy is applied to different nodes of OpenStack (Controller, Compute, Storage), while in Scenarios 8 and 9, the effect of clustering technique on availability, in addition to redundancy, is investigated.

4.1.1 | Scenario 1

The OpenStack private cloud architecture is represented by an RBD model in Figure 3. This scenario has three main nodes: The Controller, Compute, and Storage. Since each node is composed of specific components that are internally connected, the RBD of Figure 3 is then expanded into several submodels. The availability of this model is obtained from Equation (1) which is the standard equation for series-parallel RBDs.[21]

$$A_{\text{Scenario}_1} = A_{\text{Controller}} \times A_{\text{Compute}} \times A_{\text{Storage}}, \tag{1}$$

where $A_{nd,\ nd\in\{\text{Controller, Compute, Storage}\}}$ is the availability of each node in the equation of $A_{\text{Scenario}_i}, 1 \leqslant i \leqslant 9$ and it is computed by analyzing the respective Controller, Compute, and Storage submodels.

The Controller node is refined by the CTMC shown in Figure 4. This CTMC is proposed to show the interdependency between the Controller internal components to process a new IaaS request for instantiating a new VM. The Controller node is composed of Horizon (H), Keystone (K), Nova (N), Glance (G), and Neutron (NT) components. As it is noted in Section 3, MySQL and RabbitMQ are used by other OpenStack major components to share the current status of the cloud and to communicate with each other, respectively. Therefore, they are not modeled separately; but we use the probability of these components working properly in transitions of some CTMC submodels. The CTMC submodel has eight states represented by H, K, N, G, NT, F, D, and P. We assume that the job arrival follows a Poisson process, which means that the jobs interarrival times are exponentially distributed with rate λ_a. State H denotes that the Controller works properly, and all components (Horizon, Keystone, Nova, Glance, Neutron, Database, RabbitMQ) are healthy. In all the submodels

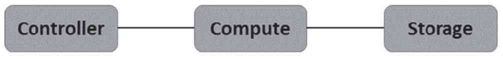

FIGURE 3 The reliability block diagram model of the OpenStack architecture used for Scenarios 1, 2, and 3

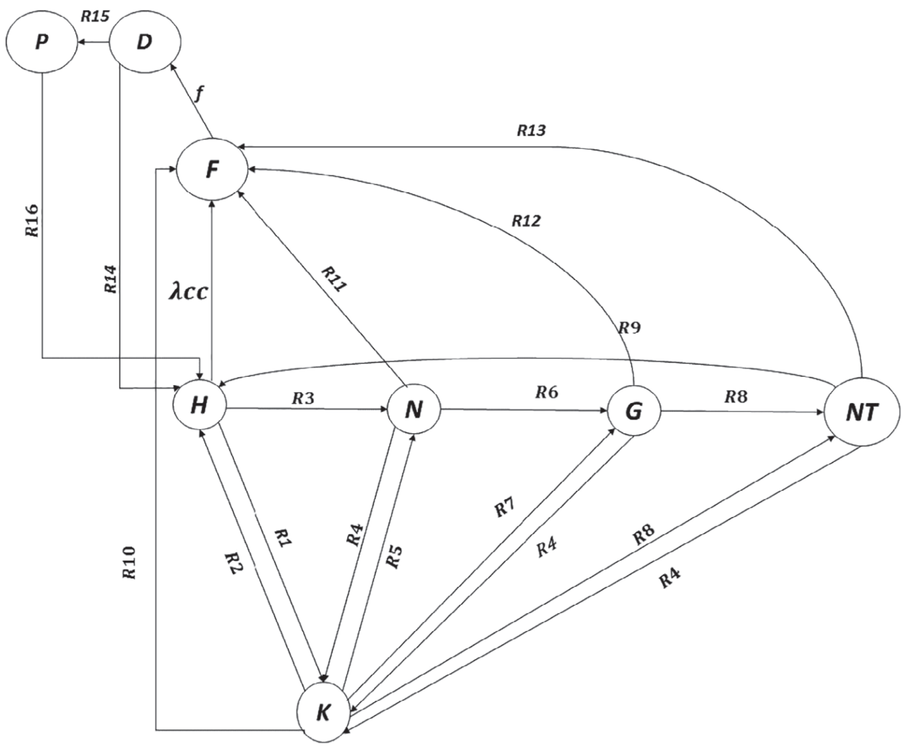

FIGURE 4 The continuous time Markov chain model proposed as the submodel for the Controller node

presented in this article, we use p_i to denote the probability of component i working correctly. For example, p_{HW} represents the probability of the Hardware working correctly.

The Controller checks every OpenStack component step by step, and satisfies any requirement which is needed to launch a new VM. If a new request arrives, and Hardware, Operating System (OS), and Keystone components work properly, the Controller first goes to the state K with rate R_1. This rate is computed by Equation (2).

$$R_1 = \lambda_a \cdot p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot p_K. \tag{2}$$

In state K, authentication of user and all OpenStack components is performed. If Hardware, OS, Database, and Horizon work properly, then the Controller moves to state H. The rate of transition from state K to state H, called R_2, is defined in Equation (3).

$$R_2 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_H, \tag{3}$$

where $1/\delta_{\mathrm{DB}}$ is the mean search time in the database. The Controller moves from state H to state N when Hardware, OS, and Nova are working properly with rate R_3 represented in Equation (4).

$$R_3 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot p_N, \tag{4}$$

where N denotes the Nova component in OpenStack. The state K can be reached from state N with rate R_4 provided that the Hardware, OS, and Keystone are healthy. The rate R_4 is computed by Equation (5).

$$R_4 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot p_K. \tag{5}$$

Similarly, state N is reachable from state K with rate R_5, defined in Equation (6), provided that the Hardware, OS, Database, and Nova are working properly.

$$R_5 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_N. \tag{6}$$

The rate of the transition connecting state N to state G which is responsible for managing VM's image, is R_6, which is represented in Equation (7) provided that the Hardware, OS, Database, RabbitMQ and Glance components, and Compute node are reachable.

$$R_6 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_Q \cdot \delta_C \cdot p_{\mathrm{Compute}} \cdot p_G, \tag{7}$$

where $1/\delta_C$ is the mean search time for finding free space to schedule a VM in the Compute node. The rate of the transition from state G to state K is R_4 represented in Equation (5). The Controller returns back to state G, with rate R_7, when the Hardware, OS, Database, and Glance components work correctly. The rate R_7 is computed by Equation (8).

$$R_7 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_G. \tag{8}$$

When the system decides to create the network of the VM and the Hardware, OS, Database, and Neutron are in the healthy condition, the Controller goes to state NT, that is responsible for VM's network. The rate of this transition is R_8 and it is defined in Equation (9).

$$R_8 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_{\mathrm{NT}}. \tag{9}$$

In state NT, Controller moves to state K with rate R_4 represented in Equation (5) and returns back to state NT with rate R_8 represented in Equation (9). When VM instantiation task completes, and Database, RabbitMQ, Hardware, OS, and Horizon components are healthy, the system returns back to state H with rate R_9 which is defined in Equation (10).

$$R_9 = p_{\mathrm{HW}} \cdot p_{\mathrm{OS}} \cdot \delta_{\mathrm{DB}} \cdot p_{\mathrm{DB}} \cdot p_Q \cdot p_H. \tag{10}$$

The failure of one component moves the Controller to state F. State F is reached from state H with rate λ_{cc}. This rate is calculated from RBD of Figure 5. When the system is in operational state, the failure of Hardware, OS, Horizon, Keystone, Nova, Glance, Neutron, Database, or RabbitMQ leads to the failure of the whole system. The state F is reached from state K due to the failure of one or more of the Hardware, OS, Glance, Database, Neutron, Horizon, Keystone, and Nova. The rate of this transition is R_{10} computed by Equation (11).

$$R_{10} = (1 - p_{\mathrm{HW}}) + (1 - p_{\mathrm{OS}}) + (1 - p_G) + \delta_{\mathrm{DB}} \cdot (1 - p_{\mathrm{DB}}) + (1 - p_{\mathrm{NT}}) + (1 - p_H) + (1 - p_K) + (1 - p_N). \tag{11}$$

A failure occurs, with rate R_{11} represented in Equation (12), leading to the Controller to move from state N to state F.

$$R_{11} = (1 - p_{\mathrm{HW}}) + (1 - p_{\mathrm{OS}}) + (1 - p_Q) + \delta_{\mathrm{DB}} \cdot (1 - p_{\mathrm{DB}}) + (1 - p_G) + (1 - p_K) + \delta_C \cdot (1 - p_{\mathrm{Compute}}). \tag{12}$$

Similarly, the Controller moves from state G to state F with rate R_{12} computed by Equation (13),

$$R_{12} = (1 - p_{\mathrm{HW}}) + (1 - p_{\mathrm{OS}}) + (1 - p_G) + \delta_{\mathrm{DB}} \cdot (1 - p_{\mathrm{DB}}) + (1 - p_{\mathrm{NT}}) + (1 - p_K) \tag{13}$$

and the system moves from state NT to state F with rate R_{13} represented in Equation (14).

$$R_{13} = (1 - p_{\mathrm{HW}}) + (1 - p_{\mathrm{OS}}) + (1 - p_Q) + \delta_{\mathrm{DB}} \cdot (1 - p_{\mathrm{DB}}) + (1 - p_{\mathrm{NT}}) + (1 - p_H) + (1 - p_K). \tag{14}$$

The rate of transition from state F to state D is f, where $1/f$ is the mean failure detection time. The state D shows the situation that the failure is detected.

The repair process is accomplished by transition from state D to state H with rate R_{14}. The rate R_{14} is computed by Equation (15).

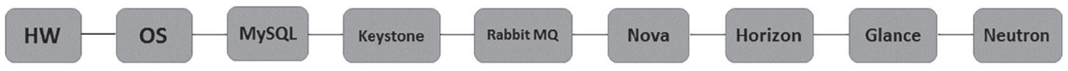

FIGURE 5 The reliability block diagram model proposed for the Controller

$$R_{14} = c \cdot \mu_1 + c \cdot \mu_{\mathrm{HW}} + c \cdot \mu_{\mathrm{OS}} + c \cdot \mu_H + c \cdot \mu_K + c \cdot \mu_N + c \cdot \mu_G + c \cdot \mu_{\mathrm{NT}} + c \cdot \mu_{\mathrm{DB}} + c \cdot \mu_{\mathrm{Q}}, \tag{15}$$

where c is the coverage factor. Parameter μ_1 is the first repair rate of the Controller which is calculated from Figure 5, and μ_x represents the repair rate of any component x of the OpenStack architecture. If a simple service restart does not solve the problem, the system moves from state D to state P with rate R_{15} represented in Equation (16).

$$R_{15} = (1-c) \cdot \mu_1 + (1-c) \cdot \mu_{\mathrm{HW}} + (1-c) \cdot \mu_{\mathrm{OS}} + (1-c) \cdot \mu_H + (1-c) \cdot \mu_K + (1-c) \cdot \mu_N + (1-c) \cdot \mu_G + (1-c) \cdot \mu_{\mathrm{NT}} + (1-c) \cdot \mu_{\mathrm{DB}} + (1-c) \cdot \mu_{\mathrm{Q}}. \tag{16}$$

In state P, the Controller has failed and since the problem has not been fixed yet, second repair is needed. The rate of transition from state P to state H is R_{16}. It is computed by Equation (17).

$$R_{16} = \mu_2 + \mu_{2\mathrm{HW}} + \mu_{2\mathrm{OS}} + \mu_{2K} + \mu_{2N} + \mu_{2G} + \mu_{2\mathrm{NT}} + \mu_{2H} + \mu_{2\mathrm{DB}} + \mu_{2Q}, \tag{17}$$

where μ_2 is the second repair rate of the Controller and μ_{2x} is assumed to be the second repair rate of any component x of the OpenStack architecture. The system is unavailable in states F, D, and P.

The second component of the RBD of Figure 3, for which a CTMC is presented, is Compute. The CTMC of Compute node, responsible for managing the placement of VMs, is depicted in Figure 6. It should be noted that IaaS providers receive heterogeneous requests from customers with various resource demands. However, in some research work, the type of customers' request is considered to be homogeneous[33,36,37] which is not realistic. In this article, we consider heterogeneous VM requests in which the number of requested CPUs are different.[17,38,39] In addition to the type of requests, the time usage of cloud resources is different from a customer to another, so the time to use and release a cloud resource is important to be modeled. If there are no available resources in any PM to be allocated to a customer request in OpenStack cloud, Compute nodes will become unavailable.

Apart from the lack of resources, the availability of the Compute nodes is another important factor affecting the capability of a cloud cluster in delivering its services.[34] An outage of each OpenStack component in Compute nodes will lead to termination of VMs running on them and will cause the cloud cluster to be unavailable. We represent the maximum number of CPUs inside a PM with m. We also assume that the number of CPUs requested by a customer is between 1 and MC, where $MC \leq m$.

In the submodel of Compute node represented in Figure 6, the job arrival rate, λ, is defined by Equation (18),

$$\lambda = \frac{\lambda_a}{n}, \tag{18}$$

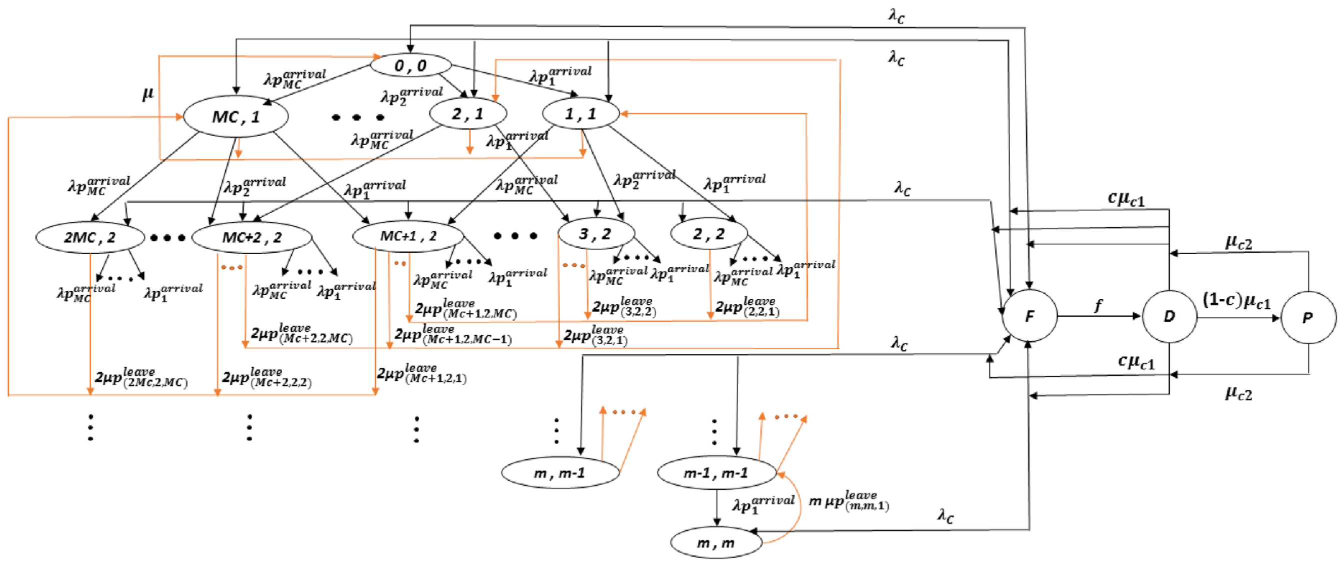
0 , 0
MC , 1
2 , 1
1 , 1
2MC , 2
MC+2 , 2
MC+1 , 2
3 , 2
2 , 2
m , m-1
m-1 , m-1
m , m
F
D
P
μ
λ_c
$c\mu_{c1}$
μ_{c2}
f
$(1\text{-}c)\mu_{c1}$
$\lambda p_{MC}^{arrival}$
$\lambda p_2^{arrival}$
$\lambda p_1^{arrival}$
$2\mu p_{(2MC,2,MC)}^{leave}$
$2\mu p_{(MC+2,2,2)}^{leave}$
$2\mu p_{(MC+1,2,1)}^{leave}$
$2\mu p_{(MC+2,2,MC)}^{leave}$
$2\mu p_{(MC+1,2,MC)}^{leave}$
$2\mu p_{(MC+1,2,MC-1)}^{leave}$
$2\mu p_{(3,2,2)}^{leave}$
$2\mu p_{(3,2,1)}^{leave}$
$2\mu p_{(2,2,1)}^{leave}$
$m\,\mu p_{(m,m,1)}^{leave}$

FIGURE 6 The continuous time Markov chain proposed as the submodel for the Compute node [Color figure can be viewed at wileyonlinelibrary.com]

where n is the number of Compute nodes, and λ is the rate of job arrivals to the Controller. In Scenario 1, n is set to 1. Each state in the CTMC of Figure 6 is described by tuple (i, j), where i is the number of consumed CPUs of the host PM and j is the number of VMs running on those CPUs. The p_k^{arrival} is the probability of requesting k CPUs by the customer, which is assumed to follow uniform distribution.[38]

$$p_1^{\text{arrival}} = p_2^{\text{arrival}} = p_3^{\text{arrival}} = \ldots = p_{\text{MC}}^{\text{arrival}} = \frac{1}{\text{MC}}. \tag{19}$$

In state (0,0), there is no VMs on the PM, hence, all the CPUs are available. If a request arrives at the Controller, the Controller checks the availability of Compute nodes and sends the request to them. Thus, Compute node moves from state (0,0) to state $(j, 1)$, $1 \leq j \leq MC$. With a new request, state $(j + i, 2)$ is reached by state $(j, 1)$, $1 \leq i \leq MC$. All states can be reached via transitions until all resources end up. If all CPUs of a PM are allocated, any new request will be rejected until some resources are released. In state (m, m), all resources are consumed and a newly arriving request is rejected. We assume Compute node is not available in this state. The system moves from state (m, m) to state $(m - 1, m - 1)$ if one VM releases an occupied CPU with rate $(m).(\mu).p_{m.m.1}^{\text{leave}}$, where $1/\mu$ is the mean service time and $p_{m.m.1}^{\text{leave}}$ is the probability that a job releases a CPU when m VMs are using m CPUs. The method for computing $p_{cc.v.crel}^{\text{leave}}$ is derived from Chang et al.,[38] where parameter cc is the number of CPUs consumed, v is the number of VMs on the host PM, and $crel$ is the number of CPUs to be released. The system also moves from state (m, m) to state $(m - j, m - 1)$, $1 \leq j \leq MC$.

In each state, the system may fail and move to state F with rate λ_c. The λ_c is calculated from the RBD model represented in Figure 7 demonstrating that the system is unavailable if one of the components shown in the RBD fails. When failure is detected, the model moves from state F to state D. The rate of this movement is f where $1/f$ represents the mean failure detection time. If failure is fixed with restart, the system returns back to the previous state with rate $c\mu_{C1}$, where c is the coverage factor and μ_{C1} is repair rate of the Compute node calculated from the RBD model of Figure 7. If a simple service restart does not solve the problem, the Compute node goes from sate D to P with rate $(1 - c)\mu_{C1}$, and after second repair, system returns to the previous state with rate μ_{C2}. Compute node is unavailable in states (m, j), $1 \leq j \leq MC$, because all resources are consumed. It is also unavailable in states F, D, and P.

The third block of the RBD of Figure 3 is the Storage node for which a RBD is presented. The RBD shown in Figure 8 illustrates that the Storage node is composed of Hardware, OS, and Cinder components. The Storage node will be available if each component represented in the RBD is available.

4.1.2 | Scenario 2

The architecture considered in Scenario 2 contains three main nodes, Controller, Compute, and Storage. The RBD of this scenario is the same as Figure 3. Each of the Controller, Compute, and Storage nodes is expanded into submodels to describe the detailed process and the interaction between their internal components. Afterward, the availabilities and downtimes of the nodes are computed. The availability of the architecture considered in this Scenario is expressed by Equation (20).

$$A_{\text{Scenario}_2} = A_{\text{Controller}} \times A_{\text{Compute}} \times A_{\text{Storage}}. \tag{20}$$

In this scenario, in order to show the effect of redundancy, we use two servers for the Controller node, one server for the Compute node, and one server for the Storage node. The existence of two servers working as the Controller, increases the availability. Though the CTMC of the Controller in this scenario is similar to the one presented for Scenario 1, there are some subtle differences. The submodel is represented in Figure 9. We apply the hot-standby replication technique to increase availability of the OpenStack architecture. Therefore, both Controllers are active at the beginning.

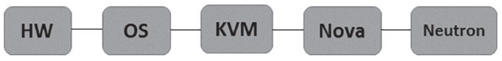

FIGURE 7 The reliability block diagram model proposed for the Compute

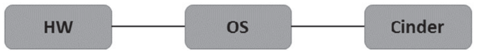

FIGURE 8 The reliability block diagram model proposed as the submodel for Storage node

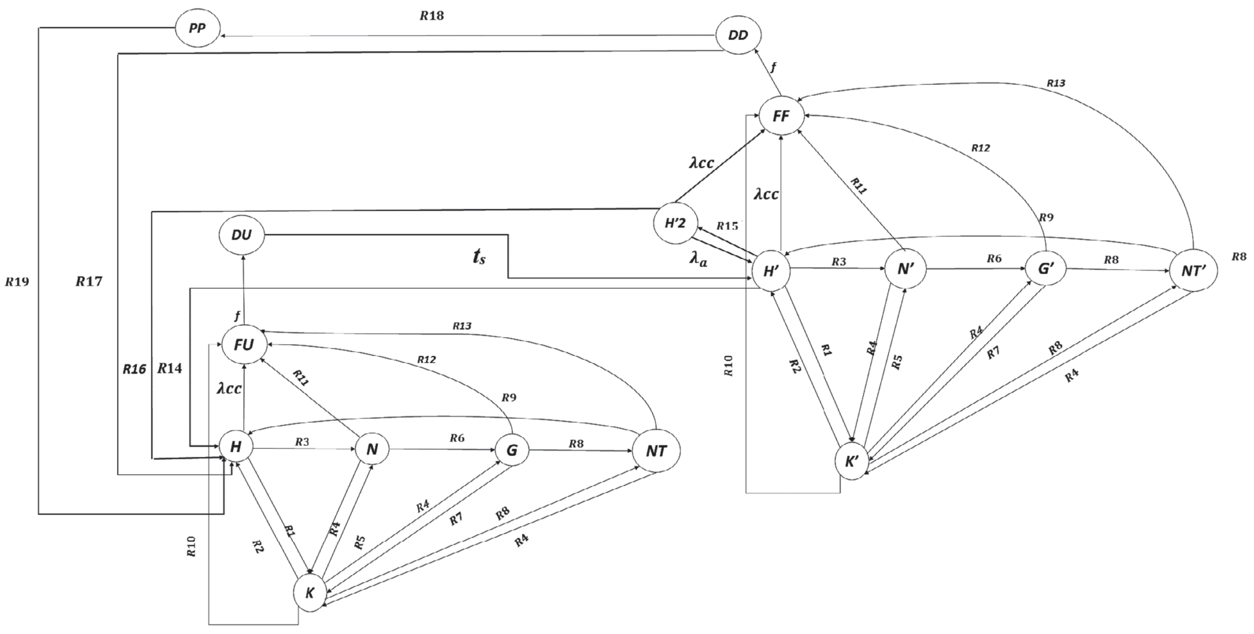

FIGURE 9 The continuous time Markov chain proposed as the submodel for the Controller node with two servers

The state *FU* represents the situation in which the first server is failed, and the second server is working properly. When the first server fails, the other server can provide OpenStack components. In state *DU*, the failure of the first server is detected. The rate of the transition from state *FU* to state *DU* is f, where $1/f$ is the mean failure detection time, and it is equal to pacemaker configuration time in OpenStack. The transition rate from state *DU* to state H' is t_s, where $1/t_s$ is the mean time that requests transfer from the first server to the second one. In state H', the Horizon component of the second server is responsible for receiving new requests.

If in state H' the failure of the first server is detected, the system moves to state H with rate R_{14} represented in Equation (15). If failure is not covered and it requires second repair, the system moves to state $H'2$ with rate R_{15} represented in Equation (16). Afterward, the system goes to state H with rate R_{16} represented in Equation (17). If a request arrives to the system when it is in state $H'2$, the system moves to state H' with the rate λ_a. If the second server fails before repairing the first server, the model moves to state *FF* with rate λ_{cc}. From the state *FF*, the model moves to state *DD* with rate f, and then returns back to state H with rate R_{17} represented in Equation (21).

$$R_{17} = 2c \cdot \mu_1 + 2c \cdot \mu_{\mathrm{HW}} + 2c \cdot \mu_{\mathrm{OS}} + 2c \cdot \mu_H + 2c \cdot \mu_K + 2c \cdot \mu_N + 2c \cdot \mu_G + 2c \cdot \mu_{\mathrm{NT}} + 2c \cdot \mu_{\mathrm{DB}} + 2c \cdot \mu_Q. \tag{21}$$

Finally, if the system requires second repair, it moves from state *DD* to state *PP* with rate R_{18} represented in Equation (22).

$$\begin{aligned} R_{18} = {} & 2(1-c) \cdot \mu_1 + 2(1-c) \cdot \mu_{\mathrm{HW}} + 2(1-c) \cdot \mu_{\mathrm{OS}} + 2(1-c) \cdot \mu_H + 2(1-c) \cdot \mu_K + 2(1-c) \cdot \mu_N + 2(1-c) \cdot \mu_G \\ & + 2(1-c) \cdot \mu_{\mathrm{NT}} + 2(1-c) \cdot \mu_{\mathrm{DB}} + 2(1-c) \cdot \mu_Q \end{aligned} \tag{22}$$

and then returns back to state H with rate R_{19} represented in Equation (23).

$$R_{19} = 2 \cdot \mu_2 + 2 \cdot \mu_{2\mathrm{HW}} + 2 \cdot \mu_{2\mathrm{OS}} + 2 \cdot \mu_{2H} + 2 \cdot \mu_{2K} + 2 \cdot \mu_{2N} + 2 \cdot \mu_{2G} + 2 \cdot \mu_{2\mathrm{NT}} + 2 \cdot \mu_{2\mathrm{DB}} + 2 \cdot \mu_{2Q}. \tag{23}$$

The submodels of the Compute and Storage nodes, in Scenario 2, are the same as those of Scenario 1.

4.1.3 | Scenario 3

The RBD of Scenario 3 is similar to Figure 3. The availability of this model is computed by Equation (24).

$$A_{\mathrm{Scenario}_3} = A_{\mathrm{Controller}} \times A_{\mathrm{Compute}} \times A_{\mathrm{Storage}}. \tag{24}$$

In this scenario, everything is similar to Scenario 2, except the number of the servers that act as the Controller node. Due to the paramount importance of the Controller's role, the availability of the Controller needs to be high enough. According to Reference 40, the minimum number of the servers as the Controller node is suggested to be three for achieving high availability. The CTMC of the Controller node in this scenario is also similar to that in Scenario 2, but if the second Controller server fails, the system receives requests by the third one. If all three Controller servers fail, the Controller node becomes down and the system gets unavailable. The Compute and the Storage submodels of Scenario 3 are the same as those represented in Scenario 2. Thus, the system will be available if the Compute/Storage node and at least one of the Controller servers run.

4.1.4 | Scenario 4

In order to evaluate the effect of Compute node on the availability, in this scenario, we consider three Compute nodes, two servers as the Controller node, and one Storage node. The RBD model of this Scenario is depicted in Figure 10. Since each Compute node operates individually and has no connection to other Compute nodes, it is separately represented by an RBD, while detailed interactions of each Compute node are modeled by a CTMC. The availability of this model is expressed by Equation (25).

$$A_{\text{Scenario}_4} = A_{\text{Controller}} \times (1 - (1 - A_{\text{Compute}}))^3 \times A_{\text{Storage}}. \tag{25}$$

The Controller and the Storage submodels are the same as those represented in Scenario 2. The CTMC of each Compute node submodel is similar to the model presented in Scenario 1. In this scenario, we use three parallel nodes as the Compute. Therefore, the system continues to operate if at least one of the Controller servers and one of the Compute nodes are available and the Storage node runs.

4.1.5 | Scenario 5

This scenario is similar to Scenario 4. However, in accordance with the OpenStack best practices,[40] the Controller node is assumed to consist of three servers. This scenario is presented because we want to compare the availabilities and downtimes of Scenarios 4 and 5 based on the best practices of OpenStack documents. The submodel of Controller node can be described by the CTMC model of Scenario 3. The availability of this model is expressed by Equation (26).

$$A_{\text{Scenario}_5} = A_{\text{Controller}} \times (1 - (1 - A_{\text{Compute}})^3) \times A_{\text{Storage}}. \tag{26}$$

4.1.6 | Scenario 6

The Storage node is one of the most important parts of the system.[41] Thus, in Scenario 6, we apply the hot-standby technique to the Storage node in order to increase the availability. Herein, we use GlusterFS,[42] as a software technique, to increase availability of Storage nodes. We consider two nodes as the Storage, connected by GlusterFS, where all data is replicated in both nodes. The RBD of Scenario 6 is depicted in Figure 11. The reason that the RBDs of the Storage nodes have been shown separately is that each of them can do its own task independently. The availability of this model is computed by Equation (27).

$$A_{\text{Scenario}_6} = A_{\text{Controller}} \times (1 - (1 - A_{\text{Compute}})^3) \times (1 - (1 - A_{\text{Storage}})^2). \tag{27}$$

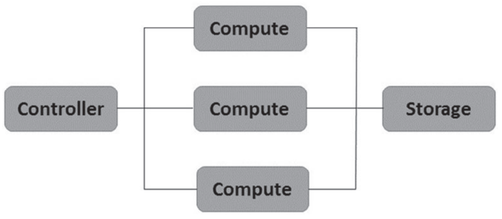

FIGURE 10 The reliability block diagram model proposed for Scenarios 4 and 5

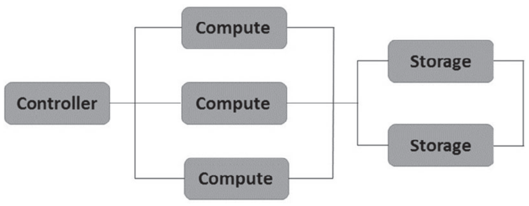

FIGURE 11 The reliability block diagram model proposed for Scenarios 6 and 7

Both the Controller and the Compute submodels are the same as those represented in Scenario 4. Each Storage node is expanded by the RBD model shown in Figure 12.

4.1.7 | Scenario 7

This scenario is similar to Scenario 6, but the number of the servers as the Controller node in accordance with the OpenStack high availability architecture is set to three. The availability of this model is obtained by Equation (28).

$$A_{\text{Scenario}_7} = A_{\text{Controller}} \times (1 - (1 - A_{\text{Compute}})^3) \times (1 - (1 - A_{\text{Storage}})^2). \tag{28}$$

4.1.8 | Scenario 8

In order to analyze the effect of an additional cluster to the availability,[28] Figure 13 represents the architecture of a highly available system consisting of two clusters. If one cluster fails, the requests are sent to the other one. The availability of this model is computed by Equation (29).

$$A_{\text{Scenario}_8} = A_{\text{Controller}} \times (1 - ((1 - (1 - A_{\text{Compute}})^3) \times (1 - (1 - A_{\text{Storage}})^2))^2). \tag{29}$$

Each block in this scenario is expanded by its corresponding submodel. The CTMCs of the Controller and each Compute node are similar to those presented in Scenario 2. Furthermore, the RBD of the Storage node is similar to the one presented in Scenario 6.

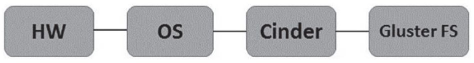

FIGURE 12 The reliability block diagram model proposed as the submodel for high available Storage node

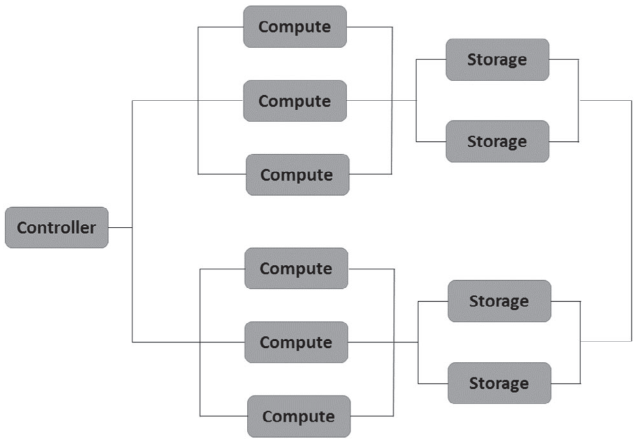

FIGURE 13 The reliability block diagram model proposed for Scenarios 8 and 9

4.1.9 | Scenario 9

This scenario studies the same clusters as described in Scenario 8. However, the number of the servers as the Controller node is assumed to be three in each cluster. The availability of this model is obtained by Equation (30).

$$A_{\text{Scenario}_9} = A_{\text{Controller}} \times (1 - ((1 - (1 - A_{\text{Compute}})^3) \times (1 - (1 - A_{\text{Storage}})^2))^2). \tag{30}$$

4.2 | Algorithm for dynamically generation of the hybrid model

In this section, a dynamic method for modeling cloud infrastructure[11,43-45] is presented. The presented method applies an algorithm to use a hierarchical modeling approach proposed in Section 4.1. As mentioned in Section 4.1, Scenario 1 does not exploit any redundancy technique, and it is presented just as a baseline for evaluating availability of later scenarios that use a redundancy technique. Scenarios 2–3, 4–5, and 6–7 are presented to evaluate the effect of redundancy in Controller, Compute, and Storage nodes on the availability, respectively. Besides, for evaluating the impact of clustering technique on availability, Scenarios 8 and 9 are proposed. However, a cloud administrator may implement the OpenStack environment differently from our scenarios by changing the number of each node type. Moreover, although the recommended configuration for setting up an OpenStack architecture on a site includes three Controller and two or more Compute and Storage nodes,[40] the Compute and Storage nodes can be increased on demand. Therefore, we need a solution to dynamically model OpenStack architecture on each site, and then calculate availability. The process of generating the hierarchical hybrid models is illustrated in Algorithm 1. The algorithm has some inputs, including (a) the information of each node, for example, node name, IP address, and password credentials, (b) Controller node's CTMCs, and (c) Storage node's RBDs introduced in Section 4.1. The CTMC of a Compute node changes based on the number of CPUs in each Compute node and should be created for each Compute node separately. The first step of the proposed algorithm is detecting the type/role of each node, that is, Controller, Compute, and Storage. Using the information of each node as a valid input of the OpenStack APIs, our algorithm obtains the role of each node and saves it into a file. OpenStack provides an API that allows administrators to write softwares and scripts for managing their own cloud nodes. The loop structure in lines 2–13 is responsible for counting the number of different node types. In lines 5 and 6, the algorithm counts the number of Controller nodes by reading the file prepared in the previous step. In lines 7 and 8, the algorithm determines the number of Compute nodes; besides, a script runs in line 9 to count the number of CPUs in each PM that plays the role of Compute node. The number of CPUs is essential to create the CTMCs of the Compute node in the next steps. In lines 10 and 11, the number of Storage nodes is counted.

After determining the number of instances of each node type, the process of building the models is started. The empty list is created, as shown in line 14. Afterward, the algorithm checks the number of Controller nodes by defining a conditional structure in lines 15–27. If the cloud infrastructure has a single Controller node, the algorithm creates one RBD and assigns its proper CTMC, based on the CTMC models of the Controller node proposed in Section 4.1. The algorithm then adds the Controller node's block to the list generated in line 14. Otherwise, according to the number of Controller nodes, the *else*statement in either line 19 or 23 is executed. The next step is the creation of the RBDs of the Compute nodes. In line 28, the empty list of the RBD model of Compute nodes is created. For each Compute node, parallel RBDs are generated, as shown in line 29. After generating the RBD model of each Compute node, the *while* structure in lines 31–36 is exploited to assign submodels of parallel RBDs by reading each line of the file prepared in line 9. In this loop, the algorithm creates CTMCs of each Compute node. Afterward, the generated CTMC is assigned to each Compute node. All parallel Compute node's blocks are added to the blocks list in line 37. According to the number of Storage nodes, the algorithm calls a function to generate parallel RBDs in line 39 and saves the output to a list. For each Storage node's block, the submodels described in Section 4.1 are assigned, as shown in line 41. In line 43, the Storage node's blocks are added to the blocks list. The algorithm generates the final model in line 44.

5 | NUMERICAL RESULTS

Herein, the models of the nine scenarios introduced in Section 4, are analyzed and compared based on the availability, downtime, and cost. Furthermore, in order to show the effect of job arrival and service rates on the availability and downtime measures, the analytical model proposed for Scenario 4, as an example, is solved with different job arrival and service rates.

Algorithm 1. Algorithm for generating Hybrid Models Dynamically

```
    Input: Node information, CTMCs of Controller nodes, RBDs of Storage nodes
    Output: Hybrid Availability Model of the Cloud System
 1: NodeInfoFile(hostname,role) ← OpenStackAPI
 2: while NodeInfoFile.hasNext() do
 3:     String Line ← NodeInfoFile.nextline()
 4:     String Fields ← Line.split(}},″ )
 5:     if Fields [1] == Controller then
 6:         Countctrl ++
 7:     else if Fields [1] == Compute then
 8:         Countcmp ++
 9:         append.NumOfCPUFile ← Fields[0].CPUInfo.sh
10:     else /* Fields [1] == Storage */
11:         Countstr ++
12:     end if
13: end while
14: blocks-list ← list
15: if Countctrl == 1 then
16:     Controller-node-block ← GenerateSeriesRBD ()
17:     AssignSubModel(Controller-node-block, CTMC-OneController)
18:     blocks-list.add (Controller-node-block)
19: else if Countctrl == 2 then
20:     Controller-node-block ← GenerateSeriesRBD ()
21:     AssignSubModel(Controller-node-block, CTMC-TwoController)
22:     blocks-list.add (Controller-node-block)
23: else /* Countctrl == 3 */
24:     Controller-node-block ← GenerateSeriesRBD ()
25:     AssignSubModel(Controller-node-block, CTMC-ThreeController)
26:     blocks-list.add (Controller-node-block)
27: end if
28: Compute-node-blocks-list ← list
29: Compute-node-blocks-list ← GenerateParallelRBD (Countcmp)
30: i=0
31: while NumOfCPUFile.hasNext() do
32:     String CPU ← NumOfCPUFile.nextline()
33:     CTMC-Compute ← GenerateCTMCCompute(CPU)
34:     AssignSubModel(Compute-node-blocks-list.get(i), CTMC-Compute)
35:     i ++
36: end while
37: blocks-list.add (Compute-node-blocks-list)
38: Storage-node-blocks-list ← list
39: Storage-node-blocks-list ← GenerateParallelRBD (Countstr)
40: for j ≤ Countstr do
41:     AssignSubModel(Storage-node-blocks-list.get(j), RBD-Storage)
42: end for
43: blocks-list.add (Storage-node-blocks-list)
44: Final-Model ← GenerateSeriesRBD (blocks-list)
```

Based on a recent cloud data center workload analysis, more than 80% of VM requests submitted by a customer need one or two CPU(s).[41] Thus, we assume $m = 4$, and $MC = 2$. The $p^{leave}_{(cc.v.crel)}$ does not always follow uniform distribution, so we use the assumption given in References 38,46.

The accurate calculation of failure and repair rates is a vital part of availability evaluation. In order to find more accurate failure and repair rates, a real testbed infrastructure for experimental evaluation of redundant architecture with seven servers is conducted. The OpenStack Newton is installed and configured on CentOS 7.0 in our experiments. The configuration of the servers is provided in Table 3. Two servers for the Controller node, two servers for the Storage node, and three servers for the Compute node are allocated. We focus on VM creation as the most critical task of an IaaS.[47] Creation and Management of a VM involve most core OpenStack services and their components. In order to increase the computation capacity, we use the CPU allocation ratio 16:1 to allocate 16 virtual cores per physical core. Similarly, the Compute nodes use the RAM allocation ratio 1.5:1 to assign VMs to a PM as long as the total amount of RAM associated with the VMs is less than 1.5 times the amount of RAM available on the PM.

We select Rally[48] as a benchmarking tool for OpenStack to generate real workload and concurrent requests. We allocate the smallest OpenStack flavor (vCPU, memory, storage) to each VM with the specification of 512 MB of RAM, one vCPU, and 1 GB of Disk in our experiments. A sustainable rate of concurrent requests is established, and 1 to 200 concurrent creation requests are handled with that flavor. The target concurrency for this benchmark is 50, 100, 150, and 200.

We ran concurrent VM creation requests twice a day (peak and nonpeak times of network usage in our lab) for 12 months. In OpenStack, different components generate logs during their execution in the respective log files. A large volume of log records are produced and stored in the system. An example of the log messages is represented in Table 4. Log messages have a specific pattern. The first column in Table 4, named *ServiceName*, demonstrates the logs related to the specified OpenStack component. The *timestamp*, *TypeOfMessage*, and the *MessageText* are shown in the second, third, and fourth column of Table 4, respectively. The system logs were continuously monitored for 12 months, using Nagios[49] as the monitoring tool. We developed a Nagios plugin that runs every 30 s to process the log records and extract information that we need as input parameters of the nine scenarios. The type of OpenStack's messages are categorized as follows: Debug, Info, Warning, Error, and Critical. In the Error and Critical situation, the component is out of service. In a healthy condition, and after returning from Error/Critical status to a normal functioning situation, the logs indicate an Info message. We create the Nagios plugin to select Error and Critical records for each OpenStack component and then extract important information, for example, *timestamp* till the component gets ready. Afterward, we count the number of failures in each component. Given the extracted data, the mean time between failures (MTBF), mean time to repair (MTTR), and the probability of component working correctly during 12 months is calculated based on the formula explained in References 17,50,51. The failure and repair rates are the reverse of MTBF and MTTR, respectively.

TABLE 3 Server description

Model	Components	Description	Considered cost
HP ProLiant DL80 Gen9	Hard Disk	490 GB	10,000$
	Memory	100 GB	
	CPU	Intel® Xeon® E5-2600 v3 @ 3.00 GHz, 14 cores	

TABLE 4 Example of OpenStack logs

Service name	Timestamp	Type of message	Message text
Nova	2020-08-25 19:24:19.407 34492	ERROR	nova.servicegroup.drivers.dbDBConnectionError: (pymysql.err.OperationalError) (2003, "Can't connect to MySQL server on '192.168.104.110')"
Glance	2020-08-25 10:02:11.467 9122	INFO	eventlet.wsgi.server[req-1a5aca78-33cf-4a43-b5b5-53c28ac386975a2601 21efc54d569a73472a66c1e37c 81687c05018241e9ac02acbc4aefb203 - default default] 192.168.104.110 - - [25/Aug/2020 10:02:11] "GET /v2/images?limit=1000sort-key=created-atsort-dir=desc HTTP/1.1" 200 18059 0.141111)
Neutron	2020-08-25 18:39:05.975 20601	ERROR	ERRORneutron.services.metering.agents.metering-agentMessaging Timeout: Timed out waiting for a reply to message ID 745bddaa3faf4594959214cc825069d9

The input parameters of models collected by the monitoring tool are described in Tables 5,6,7,8, and 9. The probability of Hardware and OS being healthy and the repair rate of them are derived from References 17 and 27. Furthermore, time parameters defined in Table 5 are derived from OpenStack High Availability configurations.[40] The Controller failure rate is derived from Figure 5, and it is 0.001550 as shown in Table 5. We consider the Controller failure rate as a sample compared with other cloud Controller's failure rates reported in References 17,25,27,28 in order to verify the parameters

TABLE 5 Input parameters of the continuous time Markov chain modeling for the Controller node

Parameter	Value or Range
Job arrival rate (λ_a)	0.1 to 5 - step 0.5 (job/h)
Probability of Hardware working correctly (P_{HW})	0.13
Probability of OS working correctly (P_{OS})	0.0023
Probability of Keystone working correctly (P_K)	0.0763
Probability of Horizon working correctly (P_H)	0.0763
Probability of Nova working correctly (P_N)	0.0497
Probability of Glance working correctly (P_G)	0.1224
Probability of Neutron working correctly (P_{NT})	0.0128
Probability of Database working correctly (P_{DB})	0.0862
Probability of RabbitMQ working correctly (P_Q)	0.1224
Mean search time in Database ($1/\delta_{\text{DB}}$)	1/3600 (h)
Mean search time to choose free Compute server ($1/\delta_C$)	1/3600 (h)
Probability of Compute node working correctly (P_{Compute})	Computed from model in Figure 6
Controller failure rate (λ_{cc})	0.001550 (h^{-1})
Mean failure detection time ($1/f$)	1/360 (h)
Mean time that requests transfer from the first server to the second one ($1/t_s$)	1/3600 (h)
Coverage factor (c)	0.95
Hardware first repair rate -covered case- (μ_{HW})	0.600 (h^{-1})
OS first repair rate -covered case- (μ_{OS})	4 (h^{-1})
Horizon first repair rate -covered case- (μ_H)	60 (h^{-1})
Keystone first repair rate -covered case- (μ_K)	60 (h^{-1})
Nova first repair rate -covered case- (μ_N)	10 (h^{-1})
Glance first repair rate -covered case- (μ_G)	20 (h^{-1})
Neutron first repair rate -covered case- (μ_{NT})	20 (h^{-1})
Database first repair rate -covered case- (μ_{DB})	30 (h^{-1})
RabbitMQ first repair rate -covered case- (μ_Q)	60 (h^{-1})
Controller first repair rate -covered case- (μ_1)	4.6511 (h^{-1})
Controller second repair rate -uncovered case- (μ_2)	2.3255 (h^{-1})
Horizon second repair rate -uncovered case- (μ_{2H})	30 (h^{-1})
Keystone second repair rate -uncovered case- (μ_{2K})	30 (h^{-1})
Nova second repair rate -uncovered case- (μ_{2N})	5 (h^{-1})
Glance second repair rate -uncovered case- (μ_{2G})	10 (h^{-1})
Neutron second repair rate -uncovered case- ($\mu_{2\text{NT}}$)	10 (h^{-1})
Hardware second repair rate -uncovered case- ($\mu_{2\text{HW}}$)	0.300 (h^{-1})
OS second repair rate -uncovered case- ($\mu_{2\text{OS}}$)	2 (h^{-1})
Database second repair rate -uncovered case- ($\mu_{2\text{DB}}$)	15 (h^{-1})
RabbitMQ second repair rate -uncovered case- (μ_{2Q})	30 (h^{-1})

TABLE 6 Input parameters of the reliability block diagram modeling for the Controller node

Parameter	Value
Hardware failure rate (λ_{HW})	0.0001141 (h^{-1})
OS failure rate (λ_{OS})	0.0003456 (h^{-1})
Keystone failure rate (λ_K)	0.000146 (h^{-1})
Database failure rate (λ_{DB})	0.000139 (h^{-1})
RabbitMQ failure rate (λ_Q)	0.0001198 (h^{-1})
Neutron failure rate (λ_{NT})	0.0001484 (h^{-1})
Nova failure rate (λ_N)	0.0001712 (h^{-1})
Glance failure rate (λ_G)	0.0001198 (h^{-1})
Horizon failure rate (λ_H)	0.000146 (h^{-1})

TABLE 7 Input parameters of the continuous time Markov chain modeling for the Compute node

Parameter	Value
Job arrival rate in Compute (λ)	λ_a/N
Probability of requesting k CPU (p_k^{arrival})	1/MC
Service rate (μ)	0.1, 0.5, 1 (h^{-1})
Probability of release *crel* CPU ($p_{cc.v.crel}^{\mathrm{leave}}$)	Is obtained from [36]
Failure detection rate (f)	360 (h^{-1})
Coverage factor (c)	0.95
Compute failure rate (λ_C)	0.001209 (h^{-1})
Compute first repair rate -covered case- (μ_{C1})	1.9087 (h^{-1})
Compute second repair rate -uncovered case- (μ_{C2})	0.9543 (h^{-1})

TABLE 8 Input parameters of the reliability block diagram modeling for the Compute node

Parameter	Value
KVM failure rate (λ_{kvm})	0.0003344 (h^{-1})
KVM repair rate (μ_{kvm})	1 (h^{-1})

TABLE 9 Input parameters of the reliability block diagram modeling for the Storage node

Parameter	Value
Cinder failure rate ($\lambda_{\mathrm{cinder}}$)	0.001652 (h^{-1})
Cinder repair rate (μ_{cinder})	12 (h^{-1})
Gluster failure rate ($\lambda_{\mathrm{gluster}}$)	0.0004629 (h^{-1})
Gluster repair rate (μ_{gluster})	1 (h^{-1})

represented in Tables 5,6,7,8, and 9. The value of this parameter was reported 0.002997 and 0.0012 in References 27 and 17,25,28, respectively. However, the values used in References 17,25,27,28 were derived from researches that are not specifically related to cloud infrastructures. Moreover, some of the components involved in the cloud Controller, were not considered in those studies. According to China Mobile's OpenStack production cloud,[47] the *zero* failure rate can be achieved by tuning cloud infrastructure. Most of the tuning actions are performed in our cloud environment before executing Rally Benchmark scripts, in order to decrease the failure probability of components.

We use the SHARPE software package[52] to numerically solve the proposed hierarchical hybrid models. The steady-state availability and the downtime of the nine scenarios are compared in Figures 14 and 15, respectively. The interval availability is the probability that the system is functioning correctly during a given period of time, and the steady-state availability is achieved when this time period is too long. The availability could also be expressed as a percentage of uptime in a given period. On the other hand, the downtime is simply the time the system is out of service. It is usually expressed as the number of minutes the system fails to perform its function in a year.[17]

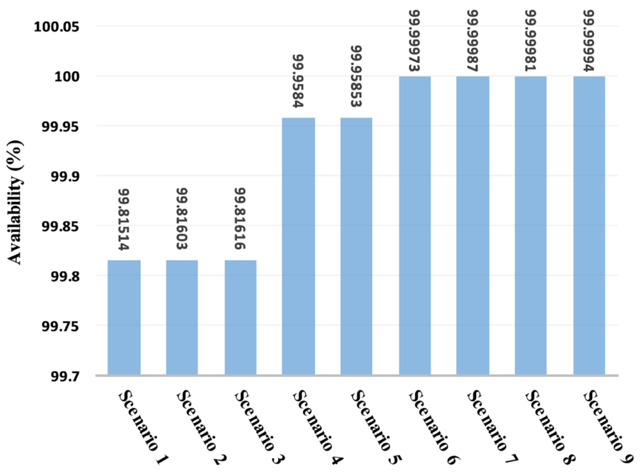

FIGURE 14 The availability of the nine scenarios considered in this article [Color figure can be viewed at wileyonlinelibrary.com]

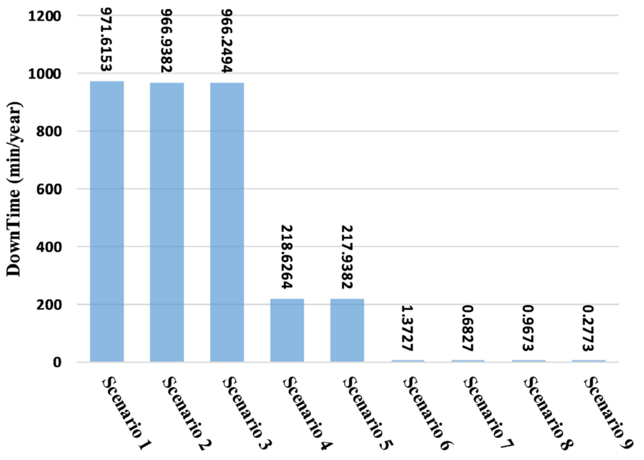

FIGURE 15 The downtime of the nine scenarios considered in this article [Color figure can be viewed at wileyonlinelibrary.com]

The main reason behind the low availability resulted from Scenario 1 is the lack of availability improving policies and the low number of Compute nodes. These servers may not only be unavailable due to failures but also cannot respond customer requests if the Compute node capacity is over. For the evaluation of Scenario 1, the private cloud without applying any redundancy technique, we obtain the value of 99.81514% for the availability and 971.6153 min/year for the downtime. The result of this scenario empathizes the use of an appropriate solution to improve availability. Therefore, in order to identify the most important subsystem of OpenStack that affect the availability, we use redundancy technique on each node. Since, the Controller is a critical part of OpenStack, we apply redundancy to this node in Scenarios 2 and 3.

In Scenario 2, the number of Compute nodes is still low, but adding one server as a Controller node increases the availability and decreases the downtime. The Controller node is an essential part of the OpenStack environment and it is important that the number of servers of the Controller node be more than one. If one of them fails, another one can be replaced. In this scenario, the values of the availability and downtime obtained by solving the proposed models are 99.81603% and 966.9382 min/year, respectively. Although we expect that the redundancy increases availability, we achieved a 0.48% reduction of downtime in comparison to Scenario 1. The results show that the Controller is in a stable condition, but using redundant Controller helps us to maintain the whole system in a good running status.

Based on high available OpenStack best practices, it is better to set the number of servers in the Controller node to three. Analysis shows that in Scenario 3, adding servers to the Controller node, the availability increases and the downtime decreases compared with Scenario 2. The values of the availability and downtime obtained from Scenario 3 are 99.81616% and 966.2494 min/year, respectively. All the above scenarios are proposed to evaluate the effect of the number of servers in the Controller node on the output measures of interest. The results show that there is a slight difference between the outputs of Scenarios 2 and 3.

The impact of adding a server to the system as the Compute node is evaluated in Scenarios 4 and 5. Increasing the number of servers of the Compute node to three, the availability increases and the downtime decreases. According to the results, the availability of Scenario 4 is 99.9584% and the downtime is 218.6264 min/year. By adding new servers to Compute node, the capacity is increased, and consequently, we experience a significant improvement in the availability and downtime compared with Scenario 1. The results show 77.49% reduction in the downtime in contrast to the baseline scenario, Scenario 1. In addition, based on OpenStack recommendation that Controller should include three or greater than three servers, in Scenario 5 we add one more Controller server to Scenario 4. However, not much difference between the results of Scenarios 5 and 4 is observed.

In addition to the significance of Controller and Compute nodes, the availability of Storage node is also important. The analysis of Scenarios 6 and 7 shows that adding a redundant node as a Storage node, the availability and downtime are improved compared with the earlier scenarios. However, there is a slight difference between the results of Scenarios 6 and 7. The values of the availability and downtime obtained by Scenarios 6 and 7 indicate that we gain over 99% improvement in the downtime when the redundancy technique is applied to the system in comparison with Scenario 1.

In Scenarios 8 and 9, using clustering techniques, availability increases and downtime decreases. We experience less than 1-min downtime per a year in these scenarios which shows a significant improvement in contrast to Scenario 1.

In addition to the importance of the availability and customer satisfaction, considering the cost as an important system-oriented parameter is essential. In our analysis, the cost of a scenario is the sum of the total price of servers required to be purchased to set up that scenario. Moreover, the maintenance and operational (M&O) costs are obtained from the Private Cloud TCO Calculator.[53] The calculator has some input fields, including the number of hosts, maximum storage capacity, cloud type, deployment type, and management type. Yearly M&O costs are calculated based on the above-mentioned inputs. The number of hosts in the corresponding calculator input field is proportional to the number of servers used in each scenario, the maximum storage capacity is the total storage employed in each scenario, the cloud type is OpenStack, and the deployment type is custom architecture. In this article, we choose a self-managed option for management. Regarding the number of servers used in each scenario and yearly M&O cost obtained from the calculator, the resulted cost values are represented in Figure 16. According to the results provided in Figure 16, cloud service providers have to determine which scenario satisfies both the customer demands and provider benefits. Therefore, if financial penalties that cloud providers must pay for every second of downtime is high, they have to choose a high available scenario. If the downtime is tolerable, they could choose a high available scenario with lower cost such as Scenario 6.

In addition to the above analysis, we quantify the effect of variation in job arrival and service rates on the availability and downtime of an OpenStack cloud. To this end, the model proposed for Scenario 4, as a sample scenario, is exploited to evaluate the availability and downtime by varying job arrival rate from 0.1 to 5 and the job service rate between 0.1, 0.5 and 1. The effects of variation in job arrival and service rates are shown in Figures 17 and 18, respectively. Changing job

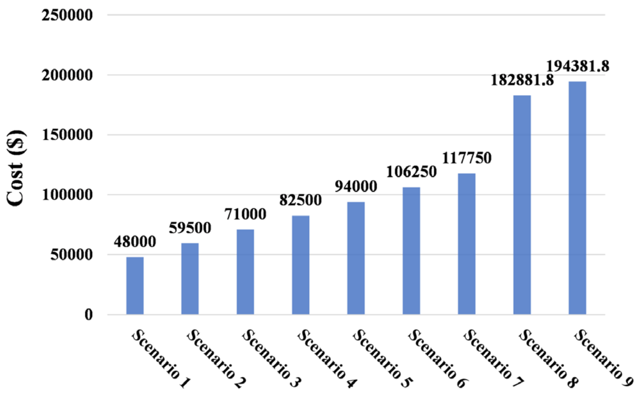

FIGURE 16 The cost of the nine scenarios considered in this article [Color figure can be viewed at wileyonlinelibrary.com]

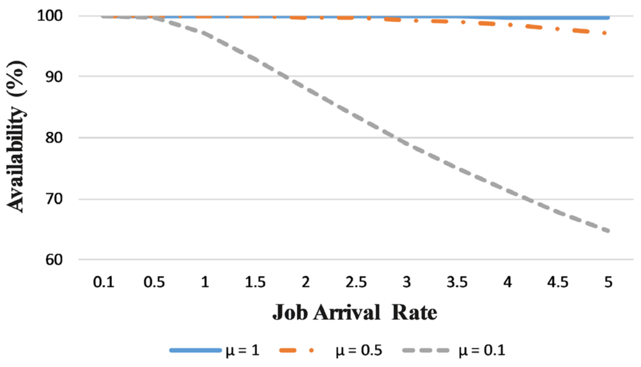

FIGURE 17 The effect of job arrival and service rates on availability [Color figure can be viewed at wileyonlinelibrary.com]

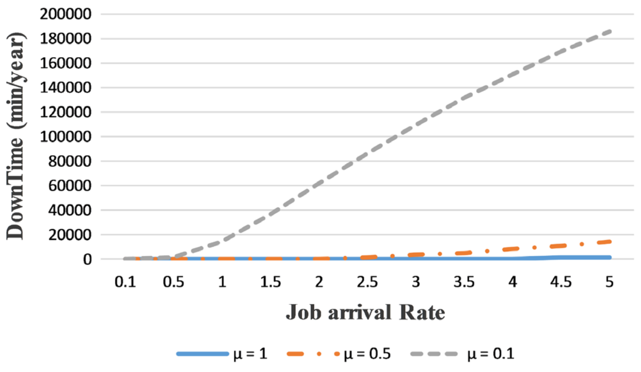

FIGURE 18 The effect of job arrival and service rates on downtime [Color figure can be viewed at wileyonlinelibrary.com]

arrival rate from 0.1 to 5, the availability decreases and the downtime increases. Moreover, by changing the job service rate from 0.1 to 1, cloud resources are released later. Hence, the availability decreases and the downtime increases.

6 | CONCLUSIONS AND FUTURE WORK

This article presented analytical hierarchical hybrid models, by combining RBDs and CTMCs, to evaluate availability, downtime, and cost of nine scenarios defined in OpenStack environment. Input parameters of the proposed models are computed in real OpenStack environment by Nagios as a monitoring tool collecting experimental data for about 1 year. In the proposed model, heterogeneous workload was considered. The results indicated that availability increases by applying the redundancy technique, which adds redundant servers and clusters to the system. Since there is an opposite relation between availability and cost, cloud administrators should provide services for the customer considering these two factors. The comparison made between availability and cost resulted from nine scenarios considered in this article can help cloud providers to make appropriate decisions on the level and type of redundancy they need. In addition, the effect of variations in job arrival and service rates on the availability and downtime was investigated in this article.

Future studies might consider more analysis on the redundancy techniques in geo-distributed cloud environments to achieve a high available system at the lowest price because cloud providers usually owned geo-distributed data centers in the real world. An appropriate architecture of cloud data centers depends on the right networks to secure the continuity of operations. Therefore, business and online services require highly available connectivity between cloud services. The models proposed in this article might also be employed to identify the role of network services in the availability of a geo-distributed cloud.

ACKNOWLEDGMENT

This work was partially supported by the Iran National Science Foundation.

ORCID

Ehsan Ataie https://orcid.org/0000-0002-3424-579X

REFERENCES

1. Calheiros RN, Ranjan R, Beloglazov A, Rose CAFD, Buyya R. CloudSim: a toolkit for modeling and simulation of Cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exper*. 2011;41:23-50.
2. Son J, He T, Buyya R. CloudSimSDN-NFV: modeling and simulation of network function virtualization and service function chaining in edge computing environments. *Softw Pract Exper*. 2019;49:1748-1764.
3. Fakhrolmobasheri S, Ataie E, Movaghar A. Modeling and evaluation of power-aware software rejuvenation in cloud systems. *Algorithms*. 2018;11(10):160.
4. Khorshed M, Ali AS, Wasimi S. A survey on gaps, threat remediation challenges and some thoughts for proactive attack detection in cloud computing. *Futur Gener Comput Syst*. 2012;28:833-851.
5. Gartner Forecasts Worldwide Public Cloud Revenue to Grow 6.3 percent in 2020. https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020. Accessed August 2020.
6. Hyperscale cloud providers shaping the platform market. https://www.forbes.com/sites/peterbendorsamuel/2020/03/02/hyperscale-cloud-providers-shaping-the-platform-marketplace/#19b51661103d. Accessed August 2020.

7. Faiz T, Noor-E-Alam M. Data center supply chain configuration design: a two-stage decision approach. *Socioecon Plann Sci*. 2019;66:119-135.
8. Armbrust M, Fox A, Griffith R, et al. A view of cloud computing. *Commun ACM*. 2010;53:50-58.
9. Jadeja Y, Modi K. Cloud computing-concepts, architecture and challenges. Paper presented at: Proceedings of the International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Kumaracoil, India: IEEE; 2012:877-880.
10. Bauer E, Adams R. *Reliability and Availability of Cloud Computing*. Hoboken, NJ: John Wiley & Sons; 2012.
11. Rosendo D, Gomes D, Santos GL, et al. A methodology to assess the availability of next-generation data centers. *Supercomputing*. 2019;75:1-25.
12. Tan S, Tahoori M, Kim T, Wang S, Sun Z, Kiamehr S. . *Long-term reliability of nanometer VLSI systems: modeling, analysis and optimization*. Switzerland: Springer Nature; 2019.
13. Cost of Data Center Outages. https://www.ponemon.org/blog/2016-cost-of-data-center-outages. Accessed August 2020.
14. DataCenter knowledge, lessons from this year's data center outages: focus on the fundamentals. https://www.datacenterknowledge.com/. Accessed August 2020.
15. Sharmaa Y, Javadia B, Sia W, Sunb D. Reliability and energy efficiency in cloud computing systems: survey and taxonomy. *Netw Comput Appl*. 2016;74:66-85.
16. Nabi M, Toeroe M, Khendek F. Availability in the cloud: state of the art. *Netw Comput Appl*. 2016;60:54-67.
17. Dantas J, Matos R, Araujo J, Maciel P. Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. *Comput Secur*. 2015;97:1121-1140.
18. Gokhale SS, Crigler JR, Farr WH, Wallace DR. System availability analysis considering hardware/software failure severities. Paper presented at: Proceedings of the 29th Annual IEEE/NASA Software Engineering Workshop, Greenbelt, MD: IEEE; 2005:47-56.
19. Open source software for creating private and public cloud. https://openstack.org/. Accessed August 2020.
20. Wei B, Lin C, Kong X. Dependability modeling and analysis for the virtual data center of cloud computing. Paper presented at: Proceedings of the The International Conference on High Performance Computing and Communications (HPCC), Banff, AB; 2011:784-789.
21. Trivedi K, Bobbio A. *Reliability and Availability Engineering: Modeling, Analysis, and Applications*. Cambridge, MA: Cambridge University Press; 2017.
22. Rimal B, Jukan A, Kastaros D, Goelven Y. Architectural requirements for cloud computing systems: an enterprise cloud approach. *Grid Comput*. 2011;9(1):3-26.
23. Maciel P, Trivedi K, Kim D. *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. Pennsylvania, PA: Information Science Reference-Imprint of IGI Publishing; 2011.
24. Harchol-Balter M. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge, MA: Cambridge University Press; 2013.
25. Dantas J, Matos R, Araujo J, Macie P. An availability model for eucalyptus platform: an analysis of warm-standy replication mechanism. Paper presented at: Proceedings of the International Conference on Systems, Man, and Cybernetics (SMC), Seoul, South Korea; 2012:1664-1669.
26. Torres E, Callou G, Andrade E. A hierarchical approach for availability and performance analysis of private cloud storage services. *Comput Secur*. 2018;100(6):621-644.
27. Matos R, Araujo J, Oliveira D, Maciel P, Trivedi K. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simul Model Pract Theory*. 2015;50:151-164.
28. Matos R, Dantas J, Araujo J, Trivedi K, Maciel P. Redundant Eucalyptus private clouds: availability modeling and sensitivity analysis. *Grid Comput*. 2017;15:1-22.
29. Ataie E, Entezari-Maleki R, Rashidi L, Trivedi K, Ardagna D, Movaghar A. Hierarchical stochastic models for performance, availability, and power consumption analysis of IaaS clouds. *IEEE Trans Cloud Comput*. 2019;7(4):1039-1056.
30. Liu B, Chang X, Han Z, Trivedi K, Rodriguez RJ. Model-based sensitivity analysis of IaaS cloud availability. *Futur Gener Comput Syst*. 2018;83:1-13.
31. Jammal M, Kanso A, Heidari P, Shami A. Evaluating high availability-aware deployments using stochastic petri net model and cloud scoring selection tool. *IEEE Trans Serv Comput*. 2017;1.
32. Chuob S, Pokharel M, Park JS. Modeling and analysis of cloud computing availability based on Eucalyptus platform for e-government data center. Paper presented at: Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Seoul, South Korea; 2011:289-296.
33. Ghosh R, Longo F, Naik VK, Trivedi K. Modeling and performance analysis of large scale IaaS clouds. *Futur Gener Comput Syst*. 2013;29:1216-1234.
34. Wang JV, Cheng CT, Tse CK. A thermal-aware VM consolidation mechanism with outage avoidance. *Softw Pract Exper*. 2019;49(5):906-920.
35. Ahmed J, Malik A, Ilyas M, Alowibdi J. Instance launch-time analysis of OpenStack virtualization technologies with control plane network errors. *Comput Secur*. 2019;101:989-1014.
36. Bruneo D. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems. *IEEE Trans Parall Distrib Syst*. 2013;25:560-569.
37. Chang X, Wang B. Muppala kJ, Liu J. modeling active virtual machines on IaaS clouds using an M/G/m/m+K queue. *IEEE Trans Serv Comput*. 2014;9:408-420.

38. Chang X, Xia R, Muppala JK, Trivedi KS, Liu J. Effective modeling approach for IaaS data center performance analysis under heterogeneous workload. *IEEE Trans Cloud Comput*. 2016;6:991-1003.
39. Khazaei H, Misic JV, Misic VB, Mohammadi NB. Modeling the performance of heterogeneous IaaS cloud centers. Paper presented at: Proceedimgs of the 33rd International Conference on Distributed Computing Systems Workshops (ICDCSW). Philadelphia, PA: IEEE; 2013:232-237.
40. OpenStack High Availability Guide. https://docs.openstack.org/ha-guide/. Accessed August 2020.
41. Tan C, Hanafi M, Hijazi A, Lim Y, Hani A. A survey on proof of Retrievability for cloud data integrity and availability: cloud storage state-of-the-art, issues, solutions and future trends. *Netw Comput Appl*. 2018;110:75-86.
42. Gluster, storage for your cloud. https://www.gluster.org/. Accessed August 2020.
43. Santos GL, Rosendo D, Gomes D, et al. A methodology for automating the cloud data center availability assessment. Paper presented at: Proceedings of the International Conference on Advanced Information Networking and Applications; 2019:1011-1023; Springer, New York, NY.
44. Moreira A, Rosendo D, Gomes D, et al. DCAV: a software system to evaluate next-generation cloud data center availability through a friendly graphical interface. *Softw Pract Exper*. 2019;49(11):1573-1599.
45. Rosendo D, Gomes D, Santos GL, et al. Availability analysis of design configurations to compose virtual performance-optimized data center systems in next-generation cloud data centers. *Softw Pract Exper*. 2020;50(6):805-826.
46. Wang B, Chang X, Liu J. Modeling heterogeneous virtual machines on IaaS data centers. *IEEE Commun Lett*. 2015;19:537-540.
47. Analyzing and tuning china mobile's openstack production cloud. https://01.org/sites/default/files/performance_analysis_and_tuning_in_china_mobiles_openstack_production_cloud_2.pdf. Accessed August 2020.
48. What is rally?. https://rally.readthedocs.io/en/3.1.0/. Accessed: August 2020.
49. Nagios-the industry standard in it infrastructure monitoring. https://www.nagios.org/. Accessed August 2020.
50. Birolini A. *Reliability Engineering: Theory and Practice*. Berlin, Germany: Springer Science & Business Media; 2013.
51. Lienig J, Bruemmer H. *Fundamentals of Electronic Systems Design*. New York, NY: Springer; 2017.
52. SHARPE Software package. http://people.ee.duke.edu/~kst/. Accessed August 2020.
53. Private cloud TCO calculator. https://ubuntu.com/tco-calculator. Accessed August 2020.

How to cite this article: Faraji Shoyari M, Ataie E, Entezari-Maleki R, Movaghar A. Availability modeling in redundant OpenStack private clouds. *Softw Pract Exper*. 2021;1–24. https://doi.org/10.1002/spe.2953