Models for Hyper-converged Cloud Computing Infrastructures Planning

Carlos Melo*, Jamilson Dantas, Paulo Maciel, and Danilo Oliveira

Informatics Center, Federal University of Pernambuco,
Recife, PE, Brazil
E-mail: {casm3,jrd,prmm,dmo4}@cin.ufpe.br
*Corresponding author

Jean Araujo

Academic Unit of Garanhuns, Federal Rural University of Pernambuco,
Garanhuns, PE, Brazil
E-mail: jean.teixeira@ufrpe.br

Rubens Matos

Federal Institute of Education Science and Technology of Sergipe,
Lagarto, SE, Brazil
E-mail: rubens.junior@ifs.edu.br

Iure Fé

Brazilian Army,
Picos, PI, Brazil
E-mail: iuresf@gmail.com

Abstract: The Data Center concept has evolved, mainly due to the need to reduce expenses with the required physical space to store, provide and maintain large computational infrastructures. The software-defined data center (SDDC) is a result of this evolution. Through SDDC, any service can be hosted by virtualizing more reliable and easier-to-keep hardware resources. Nowadays, many services and resources can be provided in a single rack, or even a single machine, with similar availability, considering the deployment costs of silo-based environments. One of the ways to apply the SDDC into a data center is through hyper-convergence. Among the main contributions of this paper, are the behavioral models developed for availability and capacity-oriented availability evaluation of silo-based, converged and hyper-converged cloud computing infrastructures. The obtained results may help stakeholders to select between converged and hyper-converged environments, due to their similar availability but with the particularity of having lower deployment costs.

Keywords: Hyper-convergence; Dependability Models; Dynamical Reliability Block Diagrams; SDDC; DRBD; Virtualization; Capacity-oriented Availability; Deployment Cost; Redundancy; Cloud Computing; OpenStack.

Reference to this paper should be made as follows: Melo et al. (2019) 'Models for Hyper-converged Cloud Computing Infrastructures Planning', *International Journal of Grid and Utility Computing*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes:
Carlos Melo received his B.S. degree in computer science from the Academic Unity of Garanhuns and the M.Sc. degree in computer science from Federal University of Pernambuco in 2016, and is currently a PhD. candidate in computer science at the same university. He is the author of scientific papers in international journals and conferences on cloud computing, synchronization systems, blockchain and analytical modeling.

Jamilson Dantas received his BS degree in Information Systems from the Seven of September Faculty, Brazil, in 2009, and received his MSc and PhD degree in Computer

Science from Federal University - UFPE, Brazil in 2013, where he is currently a PhD student. His research interests include performance and dependability evaluation, Markov chains, Petri nets, and other formal models for analysis and simulation of computer and communication systems. He is working on research projects encompassing video streaming service, cloud computing systems and network traffic modeling.

Jean Araujo received his B.S. degree in Information Systems from the Seven of September Faculty, Brazil, in 2008. He specializes in network security computer by Gama Filho University. He earned his M.Sc. in computer science from the Informatics Center of the Federal University of Pernambuco in 2012, and is currently a PhD. student in computer science at the same university. He is the author of scientific papers in international journals and conferences on software aging and rejuvenation, cloud computing and analytical modeling. Among the various areas, stand out performance and dependability evaluation, computational modeling and distributed systems. Prof. Araujo is currently assistant professor by the Federal Rural University of Pernambuco, and a member of IEEE and Brazilian Computer Society.

Paulo Maciel received the degree in electronic engineering in 1987 and the M.Sc. and Ph.D. degrees in electronic engineering and computer science from the Federal University of Pernambuco, Recife, Brazil, respectively. He was a faculty member with the Department of Electrical Engineering, Pernambuco University, Recife, Brazil, from 1989 to 2003. Since 2001, he has been a member of the Informatics Center, Federal University of Pernambuco (UFPE), where he is currently an Associate Professor. In 2011, during his sabbatical from UFPE, he stayed with the Department of Electrical and Computer Engineering, Edmund T. Pratt School of Engineering, Duke University, Durham, NC, USA, as a visiting professor. His current research interests include performance and dependability evaluation, Petri nets and formal models, encompassing manufacturing, embedded, computational, and communication systems as well as power consumption analysis.

Rubens Matos received the B.S. degree in computer science from Federal University of Sergipe, Brazil, in 2009. He received his M.Sc. degree in computer science from Federal University of Pernambuco UFPE, Brazil in 2011, and the Ph.D. degree at the same university in 2016. He is currently a professor at the Federal Institute of Education, Science, and Technology of Sergipe. His research interests include performance and dependability evaluation, Markov chains, Petri nets, and other formal models for analysis and simulation of computer and communication systems. He has worked on research projects encompassing telemedicine systems, network traffic modeling, distributed storage mechanisms, and cloud computing systems.Danilo Mendonça Oliveira

is a Ph.D. candidate at the Federal University of Pernambuco (Brazil). He received his bachelor degree in Computer Science from Federal University of Sergipe (Brazil) and his master degree from Federal University of Pernambuco. His research interests are cloud computing, mobile computing, data science, and performance and dependability evaluation with simulation and analytical modeling.

Iure Fé received the B.S. degree in Computer Science from the Federal University of Piauí (UFPI), Brazil, in 2010, and received his M.Sc. degree in Computer Science from Federal University of Pernambuco – UFPE, Brazil in 2017. He is also a system analyst in the 3rd Batalhão de Engenharia de Construção (3BEC), Brazil. His research interests lie mostly in performance and dependability evaluation of computer systems through stochastic models.

1 Introduction

Software-Defined Data Center (SDDC) is characterized by the virtualization of network, storage and other computational resources [28]. One for of reaching the SDDC concept is through hyper-convergence, by using the hypervisor as a mean to deploy compute and storage on the same node in a computational infrastructure [11]. The International Data Corporation (IDC) predicts that the market of hyper-converged systems will experience a 60% annual growth until 2019, with 3.9 billion dollars on sales [11].

Some studies have been carried out in the hyper-converged and SDDC fields, such as the one proposed by Azagury *et al.* [4], that investigates the performance of hyper-convergence in computing and storage resources. Besides, Abhijith *et al.* [26] proposed an algorithm to maximize the utilization of Tier 0 storage in hyper-converged environments. In turn, Taniguchi *et al.*[25] evaluated an energy-efficient architecture for SDDC infrastructures, while Zhang *et al.*[28] evaluated the software defined network (SDN) as a mean to reach the SDDC concept.

This paper aims to design behavioral, combinatorial and hierarchical models to evaluate the availability and capacity-oriented availability (COA) of private cloud computing infrastructures based on the OpenStack platform in order to show stakeholders the possibility to reduce expenses with physical space, maintenance teams and provisioning complexity, while improving their system availability and the metrics associated. Furthermore, this paper promotes the Red Hat distributed file system, known as Ceph [23]; a tool to reach hyper-convergence on the OpenStack cloud computing platform, i.e, to deploy storage and computing services on the same node.

The obtained results between traditional models and the hyper-converged model are compared in order to demonstrate the feasibility of adopting this trend. The costs of acquiring the necessary equipment for the provision of these infrastructures were also raised. Thus indicating a relation between reduction of expenses and the availability of each proposed and evaluated computational environment. The results obtained showed that hyper-convergent environments have availability values close to the obtained for full triple redundant environments, but with a better cost-benefit relationship.

The remainder of this paper is organized as follows: Section 2 provides the related works and our main contributions. In Section 3, an introduction to dependability and behavioral models are provided. In turn, Section 4 presents the hyper-convergence over private cloud computing environments. Section 5 shows the methodology adopted by this paper, while section 6 shows the availability models which represent the proposed architectures. Section 7 evaluates the feasibility of the models through two case studies. Finally, Section 8 shows the final remarks about the main results obtained by availability models and an evaluation of deployment costs, as well as future works.

2 Related Works

Despite the evident growth in the adoption of the hyper-convergent infrastructures, the amount of academic work that has been carried out in the hyper-convergence field is limited, which does not apply to the SDDC area. The works that underlie this paper are presented in this section.

In [4], the hyper-convergence in computing and storage environments are introduced, with the performance and scalability characteristics of these architectures also being evaluated through a prototype implementation and deployment of the IBM General Parallel File System (GPFS). [26] proposed an optimized algorithm, which follows hyper-convergence, aiming to maximize the utilization of higher tier (Tier 0) storage, such as Solid State Drives (SSD) over Serial Advanced Technology Attachment (SATA) (Tiers 1, 2, 3...), with the write and read capacity being the performance metrics evaluated on both works.

In turn, the study carried out by [13], Markov chains with rewards modeling formalism were proposed to evaluate steady-state availability and capacity-oriented availability (COA) of servers. As for [19], Stochastic Petri Net and analytical models are proposed to evaluate the number of virtual machines that a node on the Eucalyptus Cloud Computing platform can provide, as well as the environmental degradation due to the occurrence of failures.

The work developed by [25] the authors proposed an energy-efficient architecture for SDDC infrastructures. The authors evaluated through simulation a heat reuse system that utilizes high-temperature exhaust heat from servers in conditioning humidity and air temperature to reduce the total power consumption in 27%. In turn, [28] proposes and evaluates the software defined network as a mean to reach the SDDC concept, aiming to minimize costs and maximize revenue of data center resources.

In [18], availability models based on Reliability Block Diagrams (RBD) and Continuous Time Markov Chains (CTMC) were proposed, evaluating a synchronization server infrastructure deployed in a cloud computing environment managed by the Eucalyptus platform. The environment was described and evaluated, aiming to provide the best deployment cost based on the system's annual downtime.

The main differences between the work carried out here and the ones presented above are our main contributions and objectives, which are listed below:

- Proposition and evaluation of combinatorial models for availability analysis of traditional and hyper-convergent cloud computing infrastructures, based on the OpenStack platform;

- Cost survey and analysis of its relationship with the availability of traditional and hyper-convergent cloud computing infrastructures in private environments;
- Application of sensitivity analysis techniques to detect the components with the highest impact on the availability of a baseline architecture managed by the OpenStack platform;

3 Background

This section describes the fundamental concepts about system modelling, dependability evaluation, which are necessary for the complete understanding of this paper.

3.1 Dependability Evaluation

Dependability is the ability of a computer system to provide services that can be both justifiably and trusted [3]. This definition considers the user's perception about the system's behavior. Evaluating the dependability of a computer system is usually the result of evaluating at least one of its five attributes: reliability, availability, maintainability, integrity, and safety [2].

In this paper, the system's availability is the main dependability attribute evaluated, which means that our focus is on the probability of a system being available now or at a certain time in the future.

Instantaneous availability is the probability that the system is operational at t. That is, $A(t) = P\{Z(t)\} = E\{Z(t)\}, t \geq 0$, where $Z(t) = 1$ when the system is operational and $Z(t) = 0$, when it is not operational. Steady-state availability, also called the long-run availability, is the limit of the availability function as time tends to infinity (Equation 1).

$$A = \lim_{t \to \infty} A(t), t \geq 0 \tag{1}$$

System availability can also be represented by the ratio between the Mean Time To Failure (MTTF), which is defined by MTTF $= \int_0^\infty R(t)\partial t$ and Mean Time To Repair (MTTR), which can be found through MTTR $= MTTF \times (\frac{UA}{A})$, where UA is the acronym for the system Unavailability, reached through $1 - A$. Equation 2 presents how the system availability was calculated in this paper.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2}$$

The time period when the system is not available is called downtime. It is usually given in units of time, as in hours per year: Downtime $= UA \times 8760h$. On the other hand, the system's annual uptime, which represents the time period when the system has been available, therefore $Uptime = 8760h - Downtime$.

The mean available capacity is another important measure. It may be expressed by the capacity-oriented availability (COA) [12] [16]. This metric allows to estimate how much service the system is capable of delivering considering failure states. COA may be calculated with Markov Reward Models (MRMs), by assigning a reward to each model state, corresponding to the system's capacity in that condition. Another form of COA evaluation is to consider the steady-state probability in each state of the system and the amount of capacity for each state. In a private cloud system, COA may be associated with the number of virtual machines supported by computing nodes, representing the number of virtual machines on the system (see Equation 3).

$$COA = \frac{\sum \forall s \pi_s \times nr(s)}{TNR}, \tag{3}$$

where $nr(s)$ is the number of operational resources in state s and TNR is the total number of resources of the infrastructure.

Dependability attribute values can be reached in many ways, from measurement to modeling. The latter was chosen, with this choice being made thanks to the main virtue provided by modeling: to provide a high-level system view with great flexibility in adapting parameters and achieving results.

Models for dependability evaluation may be broadly classified into combinatorial (or non-state space) and state-space models.

- Combinatorial models capture conditions that make a system fail (or to be working) regarding structural relationships between the system's components.
- State-space models represent the system's behavior (failures and repair activities) by its states and event occurrence expressed as rates or distribution functions. These models allow to represent more complex relations between components of the system than combinatorial models do.

State-space models support representing system's resource constraints, as well as the impact of a failure of a component in the behavior of other components. On the other hand, combinatorial models usually demand less computation power to be evaluated.

Reliability Block Diagram (RBD) and Fault Tree (FT) are among the most prominent combinatorial modeling formalisms, whereas Stochastic Petri Nets (SPN), Continuous Time Markov Chain (CTMC) and Stochastic Automata Networks are state-space modeling formalisms [15, 24, 22, 10].

Reliability Block Diagrams, Dynamical Reliability Block Diagram (DRBD) and an extension from Petri Nets, called Stochastic Petri Nets (SPN), are the modeling formalisms chosen to represent the behavior and availability of our proposed system.

DRBDs are an extension from the traditional RBDs that consider dependencies on dynamic systems, priority

between repairs and resource sharing [8] [27]. Besides, if the modeled system is too big to be shown in SPN, or if the SPN relations (arcs and transition annotations) are too complex so that their graphical notation loses its visual appeal, DRBD becomes an alternative. The main attributes of DRBDs are the SDEP (state dependency) block and SPARE (spare component) block modeling constructs.

RBDs can be analytically evaluated by applying series, parallel and other structural compositions to its blocks. Unfortunately, this approach cannot be applied to DRBDs. In order to solve a DRBD, it is necessary to map it into models that can be solved analytically, numerically or through simulation. However, the main benefit of using the DRBD is its clean and intuitive graphical notation, which allows a hierarchical representation of a large system with complex relations between its components. In this work, DRBDs are mapped into SPNs, as shown in Figure 1, which represents a SDEP block.

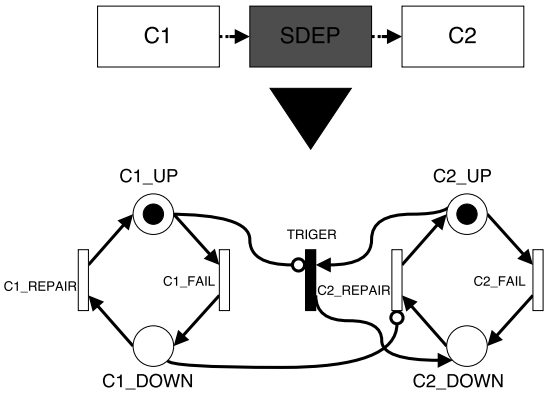

Figure 1 SDEP block conversion to SPN

The SDEP block is able to connect a source block to one or more target blocks. If this block enters into failure state, then all target blocks will fail as well. This can be seen in Figure 1, where an immediate transition called **TRIGGER** moves a token from place $C2\_UP$ to place $C2\_DOWN$ place. The inhibitor arc connected to the **TRIGGER** transition and the $C1\_UP$ place grant that this transition only fires if the source block fails, which leads to the failure of all connected target components. Each SDEP block is converted into the SPN showed, meaning that, for big and complex systems with many SDEP blocks, an equivalent SPN would be even bigger and harder to understand.

3.2 Sensitivity Analysis

To determine the most influential factors on interest metrics Sensitivity analysis techniques are considered helpful tools [9]. In this paper, the percentage difference sensitivity analysis technique is employed, characterized by a sensitivity index known as $S_\theta(Y)$, which indicates the impact of a given measure Y with respect to a parameter θ.

Equation 4 shows how the percentage difference index is calculated for a metric $Y(\theta)$, where $max\{Y(\theta)\}$ and $min\{Y(\theta)\}$ are the maximum and minimum output values, respectively, computed when varying the parameter θ over the range of its n possible values of interest. If $Y(\theta)$ is known to vary monotonically, only the extreme values of θ (i.e., θ_1 and θ_n) may be used to compute $max\{Y(\theta)\}$; $min\{Y(\theta)\}$, and subsequently $S_\theta(Y(\theta))$ [17].

$$S_\theta(Y) = \frac{max\{Y(\theta)\} - min\{Y(\theta)\}}{max\{Y(\theta)\}} \quad (4)$$

The $S_\theta(Y)$ computation is done while the values of the model parameters are fixed, which is carried out for each input parameter until all parameters have been analyzed and the one with the most significant impact is found.

4 Hyper-convergence in Private Clouds

The main advantages of hyper-convergent infrastructures over converged (same type and size of devices distributed on the same rack) and silo [11] based ones (disorganized and nonstandard) are the possibility to provide services with high availability, less components and lower costs, which is possible by virtualizing storage resources with the same hypervisor that provides Virtual Machines (VM).

The silo-based architecture is the traditional infrastructure model, which relies on proprietary hardware forming separated silos, with each silo representing a purpose-built hardware with its own specialized software, that need to be managed by dedicated specialists [11].

On silo-based non-redundant architectures, several points of failures are presented, as each component is a point of failure, which means that if any of them enters into a failure state all services provided stop working. This is the main reason why redundant converged and hyper-convergent environments aroused, achieving better availability and reliability results through architectures with no single point of failure.

4.1 OpenStack Platform

The OpenStack 15th version (Ocata) [21] is the cloud computing platform chosen for this work, due to its ability to adapt to plugins for the provision of hyper-converged infrastructures with Ceph [1]. The OpenStack controls pools of compute, storage and networking resources throughout a data center [21].

[21] describes the main components of OpenStack, with the **Keystone** being the identity service used by OpenStack for security, authentication and high-level authorization for each of the OpenStack components; while the **Nova Compute** is the component responsible to provide computing power on the OpenStack cloud

computing environment. Nova manages the **hypervisor** to provide virtual machines. In turn, **Neutron** (Networking) implements OpenStack's "network as a service" model and it as an OpenStack core component, which means that it must be present in all infrastructures. The OpenStack **Glance** is the image service that provides image upload and data assets. The **Swift** is the Object Store project that offers cloud storage software for data store and retrieve. The broker, called **RabbitMQ**, implements the Advanced Message Queuing Protocol (AMQP), while the **Network Time Protocol** (NTP) keeps time between servers synchronous and the **Database** stores services information. All of these impact on the virtual machines (VM) provision, with the OpenStack default hypervisor being the Kernel-based Virtual Machine (KVM). Figure 2 presents how the OpenStack main components on a traditional architecture are organized: Controller, Compute node and Storage.

The Ceph storage system can auto-scale to the exabyte level and beyond. This reduces the need for an external storage device or machine, by providing the storage service into the compute node [21]. This new organization means a change in the OpenStack software stack, as presented in Figure 3, which shows how the service is layout into a hyper-convergent OpenStack architecture.

With the Red Hat Ceph and the storage service virtualization approach, It can obtain similar availability values, with up to 33% fewer components. At a hyper-convergent environment, the storage services are deployed on the compute nodes; thus, an external storage device is no longer needed.

At a hyper-convergent cloud computing environment, the compute nodes host the storage resources, which means that if a compute node enters into a failure state, then the object data is still available to other virtual machines. The steps followed to create, reach and evaluate our proposed architectures and models are described in Section 5, while Section 6 describes the proposed availability models.

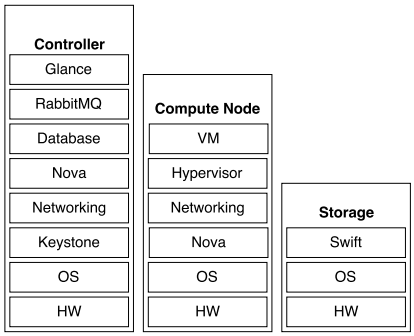

Figure 2 OpenStack service Layout

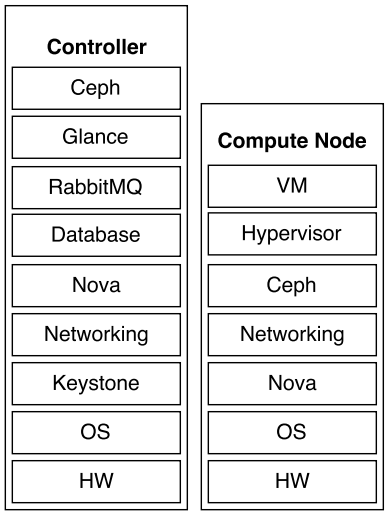

Figure 3 OpenStack Service Layout with Hyper-convergence

5 Modeling Methodology

The first step to evaluate the availability of our cloud computing environment is to survey all the hardware and software resources available and needed to deploy a baseline architecture, as presented in Section 4. The considered resources must be relevant to the availability evaluation.

After listing the most relevant hardware and software resources to deploy the baseline environment, it is possible to make the model's proposition that represents the environment's behavior. Figure 4 shows an organization chart that summarizes the strategy used in this paper.

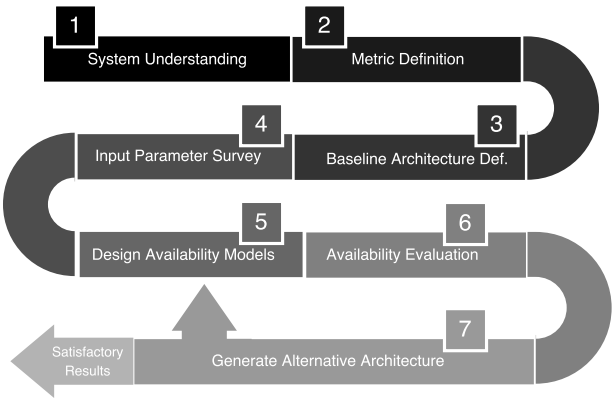

Figure 4 Availability Evaluation Methodology

1. **System Understanding**, characterized by the identification of system components, applications, functionalities and acquisition costs for each required equipment;

2. **Metric Definition**, by knowing how the system works, we can choose the metrics of interest to evaluate;

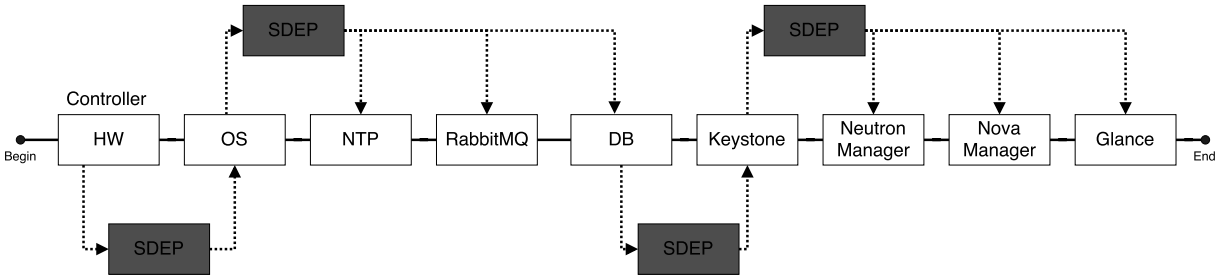

Figure 5 DRBD for Controller

3. **Baseline Architecture**, an architecture containing the minimum number of components required for the deployment of the OpenStack platform, which includes a separate controller, compute and storage;
4. **Input Parameter Survey**, survey of mean time to failure (MTTF) and mean time to repair (MTTR), the components and applications inherent to the proposed infrastructure;
5. **Design Availability Models**, proposition of combinatorial and state-based models for availability evaluation based on the architecture' behavior;
6. **Availability Evaluation**, with the availability and capacity-oriented availability models that represent the architectures, we can evaluate our metrics of interest;
7. **Generate Alternative Architecture**, based on the values obtained through a sensitivity analysis, it is possible to use techniques, such as redundancy, to improve the system's availability values.

6 Dependability Models

This section describes the models proposed to evaluate the system's availability. Dynamic Reliability Block Diagrams (RBD) were used to represent the proposed architectures. Behavioral models present the relationship between the OpenStack cloud computing software and hardware components.

6.1 Baseline Architecture

The OpenStack controller is represented by a DRBD with some dependencies, as shown in Figure 5. The controller machine shown in Figure 2 contains hardware, an operating system, Nova manager, Glance, Neutron, Keystone Identity Manager, Network Time Protocol (NTP) and RabbitMQ.

The hardware (HW) component is the base for each machine in the cloud computing environment; all the software components run over it and, if the hardware element enters into a failure state, the entire software will fall as well. To repair the machine that had a failure into the hardware component, we must start the repair routine by It, after the hardware repair the operating system (OS) becomes the next component to be repaired, after that, all the other software running over It. DRBDs work with the concepts of Priority and Shared resources. The SDEP block establishes that one thing must be repaired befored another. In the previous example, the operating system can only be fixed if the hardware has already been restored, and so on.

The controller machine DRBD have as dependencies, besides hardware and operating system, the Database (DB). If this component fails, the Keystone Identity service goes down and the platform cannot be managed anymore, since Neutron, Nova and Glance managers will fall as well. The controller machine attribution is the platform management (PM). The operational mode (OM) that describes the PM is expressed by the relationship between HW $\wedge$ OS $\wedge$ NTP $\wedge$ RabbitMQ $\wedge$ DB $\wedge$ Keystone $\wedge$ Neutron $\wedge$ Nova $\wedge$ Glance. In order to manage the OpenStack normally, all these components must be working.

For the compute node, another DRBD is proposed and presented in Figure **??**. This DRBD consists of hardware (HW), operating system (OS), Nova compute, Neutron networking service, hypervisor and Virtual Machine.

This DRBD describes the dependencies between node components. As previously stated, if the hardware enters into a failure state, then all software components go down as well. As for the operating system, it can take with all services running at the same time: Neutron, Nova, Hypervisor and Virtual Machine. If the hypervisor fails, the Virtual Machine goes down. The same happens for Neutron, which is the network provider; meaning that the virtual machine is not going to be reachable anymore. If the Nova compute is the one to fail, the hypervisor, which is controlled by Nova, will not be able to launch any more virtualized resources. The OM describes this DRBD, expressed by the relationship between HW $\wedge$ OS $\wedge$ Neutron $\wedge$ Nova $\wedge$ Hypervisor. If all these components are available, the virtual machines can be provided.

The last component of the baseline architecture is the object storage device (OSD), represented by the DRBD in Figure 7, and containing hardware (HW), operating system (OS) and the Swift object storage. The OM for this DRBD is represented by the relationship between

HW ∧ OS ∧ Swift, and all these components must be available to provide the storage service.

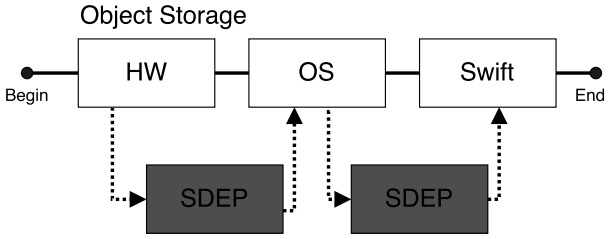

Figure 7 DRBD for storage

By combining the three DRBDs that represents the baseline architecture, we have the baseline RBD with three blocks, with each one being a component subsystem, as shown in Figure 8. All the three blocks must be operational in the OM: Controller ∧ Compute ∧ Storage.

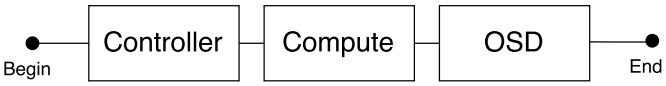

Figure 8 RBD for baseline architecture

6.2 Double Redundant Architecture

In the baseline scenario, if any of the cloud components fail, the system will not be available anymore. Futhermore, any service that was running on the compute node will not be accessible from the outside of the infrastructure. To reduce the unavailability period, we propose a model with double redundancy, represented in Figure 9.

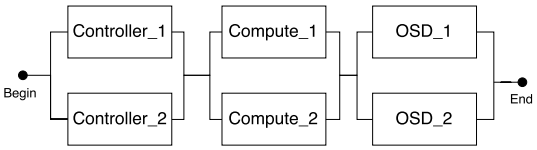

Figure 9 RBD for double redundant architecture

In this second RBD, all components previously presented have an active-active redundancy, sharing all the works and services with each other. If any component enters into a failure state, the other is going to absorb the workload performed by its counterpart with at least a half of the original capacity. In case of two of the same components stop working, then the entire system becomes unavailable. The logic function that describes the OM of the double redundant environment is expressed by: (Controller 1 ∨ Controller 2) ∧ (Compute 1 ∨ Compute 2) ∧ (OSD 1 ∧ OSD 2).

6.3 High Availability Models for a triple redundancy environment

A highly-available environment has an annual downtime of less than six minutes. On OpenStack, in order to reach a similar with default settings, at least nine physical machines, three controllers, three compute nodes and three object storages are needed, with no single point of failure in this setting. Figure 10 presents the RBD that represents this environment.

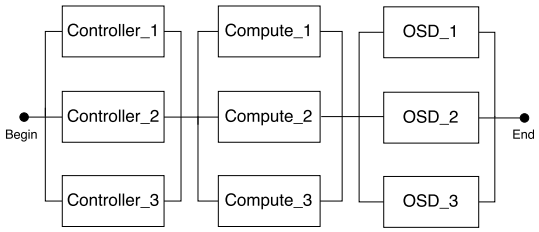

Figure 10 RBD for High Available triple redundancy environment

At least one controller, one compute node and one storage device must be operational to provide the services and maintain the virtual machines, images and user data, with each of the blocks having a DRBD as subsystem. The logic function that describes the OM for the triple redundant environment is expressed by: (Controller 1 ∨ Controller 2 ∨ Controller 3) ∧ (Compute 1 ∨ Compute 2 ∨ Compute 3) ∧ (OSD 1 ∧ OSD 2 ∧ OSD 3).

6.4 High Availability for a Hyper-convergent Architecture

To turn an OpenStack cloud computing environment into a hyper-convergent one, we must provide a way to keep data safe without an external storage device, and with similar availability values to the triple redundancy scenario. The Red Hat Ceph is a tool that accomplishes this. Ceph has a monitor on the controller, as implied by some modifications to the DRBD presented in Figure 5, as shown in Figure 11. This new OM is expressed by HW ∧ OS ∧ Ceph_Monitor ∧ NTP ∧ RabbitMQ ∧ Nova ∧ Glance ∧ Neutron.

The Ceph monitor is responsible for providing control over Ceph on compute nodes, and analyzing where the object data of the virtual machine is going to be stored and where it is if a compute node fails. Its main dependency is the hardware, as well as of all applications already presented. Figure **??** shows the DRBD for compute node, now containing the Ceph Object Storage. The OM that describes this DRBD is HW ∧ OS ∧ Ceph ∧ Neutron ∧ Nova ∧ Hypervisor. If all these components are available, then virtual machines can be provided.

The hypervisor controls the Ceph object storage. For the hyper-convergent environment, we propose another RBD in Figure 13.

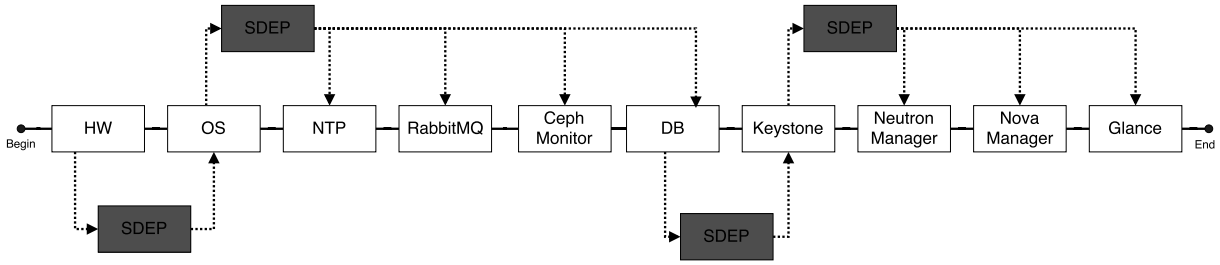

Figure 11 DRBD for Controller with Ceph Monitor

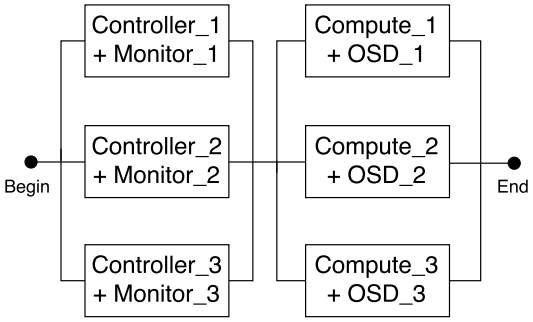

Figure 13 RBD for High Available hyper-convergent architecture

With six blocks, three controllers/monitors and three computing/storage nodes with no single point of failure, we may reach high availability values, close to the ones reached by full triple redundant architectures with nine machines, maintaining user's data safe, a self-healing mechanism and 33% fewer components. The logic function that describes the OM of the hyper-convergent environment is expressed by: (Controller_Monitor 1 $\vee$ Controller_Monitor 2 $\vee$ Controller_Monitor 3) $\wedge$ (Compute_OSD 1 $\vee$ Compute_OSD 2 $\vee$ Compute_OSD 3).

6.5 Capacity-oriented Availability Script

The compute node availability model enables us to determine the capacity-oriented availability (COA); i.e., the real amount of available resources delivered to system's users can be determined. For such a purpose, this paper evaluates how many virtual machines our platform can handle and provide, as well as what are our limitations to afford these virtual machines.

A Mercury script [20] extracts the COA values from the proposed DRBDs. This enables the creation of a parameterizable evaluation mechanism, such as the number of virtual machines that a node can host, based on the available hardware resources. The proposed script obtains the COA values through analytical evaluation or simulation, with the latter employed in case of the number of virtual machines is too large to be represented graphically or if the model generates a large state space. The proposed script is presented as follows.

```
//n defines the amount of virtual machines
n = 4;
//create the DRBD model
DRBD Model{
    //define the DRBD blocks and their MTTR and MTTF
    block HW( MTTF = 8760.0, MTTR = 1.66);
    block OS( MTTF = 2893.0, MTTR = 0.25);
    block Neutron( MTTF = 788.4, MTTR = 1.0);
    block Nova( MTTF = 788.4, MTTR = 1.0);
    block Hypervisor( MTTF = 788.40, MTTR = 1.0);

    //instantiate the declared amount of VMs
    for i in range(1, n){
        block VM#($i)( MTTF = 2880.0, MTTR = 0.019);
    }

    //define the dependencies between components
    SDEP HW -> OS;
    SDEP OS -> Neutron;
    SDEP OS -> Nova;
    SDEP OS -> Hypervisor;
    SDEP Nova -> Hypervisor;

    for i in range(1, n){
        SDEP OS -> VM#($i);
        SDEP Neutron -> VM#($i);
        SDEP Hypervisor -> VM#($i);
    }

    //define blocks organization
    parallel VMs( blockName = "VM$i", from = 1, to = 4 );
    series s0(HW, OS, Neutron, Nova, Hypervisor, VMs );

    //compute node's operational mode (OM)
    top s0;

    //VMs start as available
    reward r(
         true -> summation( "VM$i_up", 1, n ) / 4
    );

    //metric definition
    metric coa = stationaryReward( rewardRate = r);
    metric coaSim = variableMetric(
        variable = Model.r,
        numberOfSamples = 100,
        batchSize = ( 365 * 24 * 5 ),
        type = "stationary"
    );}
main{
    //coa calculates the COA analytically
    //coaSim calculates the COA through simulation
    coa = solve(Model, coa);
    coaSim = solve(Model, coaSim);
}
```

The previous script was created in Mercury Tool Script Language [20]; **Line 2** defines n as the number of virtual machines hosted by each node. In turn, the **Line 4** marks the beginning of the proposed DRBD model, which resembles an XML file. **Lines 6 - 10** describes each block presented in Figure **??**, it is important to mention that each block has a MTTF and a MTTR parameter, both are parameterizable.

A *for* loop starts at **line 13**, then it adds the MTTF and MTTR for each declared VM. In turn, **lines 18 - 22** defines the state dependencies (SDEP) between each block in the DRBD model, already the second *for* loop, started in **Line 24** works in the dependencies of the virtual machine. **Line 31** put the virtual machines in a **parallel** way, meaning that at least one of them must be operational in order to accomplish the service provisioning. The **line 32** order the blocks in **series** in order to provide the model of Figure **??**.

The **top s0** builds the model while the **reward r** present in **Lines 38 - 40** defines the initial state of each VM (Operational). The evaluated metrics are defined in **Lines 43 - 49**. The main function in **line 50** executes the model evaluation and solves the COA analytically and through simulation.

7 Case Study

This section provides three case studies that demonstrate the feasibility of the proposed models: Availability Evaluation, Capacity-oriented Availability Evaluation and Deployment Cost Evaluation. First, in order to evaluate our models, a list of input values is needed. Those values used for the components of the OpenStack cloud computing platform and other software applications, such as failure and repair rate of hypervisor and Virtual Machines are extracted from information of hardware manufactures and from the works carried out by [7, 6, 5, 13]. The input values for the proposed models can be seen in Table 1. It is important to point out that all the proposed models and their respective metrics were evaluated and extracted from the Mercury Tool [14].

Table 1 Input values for Cloud computing infrastructure components

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
DB	1440 h	20 min
Nova, Neutron, Keystone, RabbitMQ, NTP, Glance, Swift and Ceph	788.4 h	1 h
Hypervisor	2990 h	1 h
Virtual Machine	2880 h	69 s

7.1 Case Study I - Availability Evaluation

The first case study evaluates the steady-state availability of the proposed models based on the input parameters fail and repair rates. The baseline architecture was the first to be evaluated, which enabled to extract the availability values for each DRBD sub-model (Controller, Compute Node and Storage), by inserting each of the input values presented on the corresponding block. Table 2 shows the availability results for each sub-model and for the cloud RBD model.

Table 2 Comparison of availability results for each subsystem model

Model	Availability (%)	Downtime (h/year)
Controller	98.614	121.41
Compute Node	99.355	55.62
Storage	99.797	17.78
Cloud Baseline	97.786	194.03

The results point out to an availability of 97.78% for the cloud baseline RBD, indicative of an annual downtime higher than 194 hours; that is, for the 365 days of the year, the system will be unavailable for eight entire days. However, it is necessary to understand which component was the responsible for this massive downtime. Since each subsystem has its own components, and each of these elements has an impact on the subsystem availability, with a consequent impact on the availability of the entire system, sensitivity analysis can answer this question.

The first step to apply the sensitivity analysis technique is provide the MTTF and MTTR for each component on the platform, as represented in Table 1. The input values for the sensitivity analysis are the times of failures and repairs for each system component. Each parameter varied between +50% and -50% of its original value, with this measure being considered adequate to cover an upper and a lower limit for the variation of architecture availability. Table 3 presents the results of the sensitivity analysis ordered by the sensitivity index rank until the 10th element.

Table 3 Sensitivity ranking for baseline architecture

Component	Rank	Value
$MTTF_{keystone}$	1st	4.77E-03
$MTTF_{nova}$	2nd	3.40E-03
$MTTF_{db}$	3rd	2.92E-03
$MTTR_{hypervisor}$	4th	2.60E-03
$MTTR_{glance,nova,neutron}$	5th	2.33E-03
$MTTR_{keystone}$	10th	2.11E-03

The most critical component, i.e., the one with more significant impact on the baseline architecture availability, is the Keystone Mean Time to Failure, which is at the top of the rank. The Keystone failure has implications on the failures of the Neutron Manager, Nova Manager and Glance, since these services cannot be accessed, and no virtual machine can be provided. The Nova compute MTTF follows the Keystone MTTF, running on compute node and is responsible for controlling the hypervisor, which provides virtual machines.

The last component of the sensitivity analysis rank is the Virtual Machine MTTF, as the failure of the VM

does not imply in the crash of other components, with a solo repair based on a reboot strategy.

By understanding which components have the greatest impact on the infrastructure availability, redundancies may be applied on them to improve the availability values obtained. A triple active-active redundancy model was proposed. As presented in Figure 10, this RBD contains three controllers, three compute nodes and three storages. In addition, the evaluation of the hyper-converged environment presents similar availability values, with even fewer components.

Since most of the ranked components are on the same interval (E-03), we need to improve all machines to advance on our analysis and evaluate the highly-available models. Table 4 presents the availability results for each architecture.

Table 4 Comparison of availability results for each architecture

Architecture	Availability (%)	Downtime (h/year)
Baseline	97.786	194.03
Double Redundant	99.976	2.07
HA Triple Redundant	99.9997	0.025 (1.6 min)
HA hyper-convergent	99.9995	0.04 (2.4 min)

The architecture of greater availability is the full triple redundant. It is worth noting that in the structure of the full triple redundancy the consumption of physical space is higher, as it has three more components. This environment fit the conditions required to be considered highly available, having an annual downtime of fewer than six minutes. However, the hyper-convergent architecture also falls with in the requirements to be regarded as highly available, having a yearly downtime of 2.4 minutes or 144 seconds.

The hyper-converged model has a lower availability than the full triple redundant model, though it contains a much smaller number of devices to reach high availability; six against nine of the counterpart, with a total of 33 % less components.

In turn, the architecture with the highest downtime and, therefore, with the lowest availability, is the baseline architecture, containing three main components, a controller, a computing node and a storage, having an annual downtime of approximately 194 hours, much higher than what is intended for a highly available environment.

The architecture with double redundancy on all components has an annual downtime close to 2 hours; a reasonable result compared to ones obtained from the baseline architecture, though it does not fit the high available parameter.

7.2 Case Study II - Capacity-oriented Availability Evaluation

The values obtained from the availability models evaluation enabled us to determine the system's COA, by calculating the real amount of VMs that can be delivered by the proposed environments. To accomplish this task, we must define the computing nodes' main settings and how many virtual machines these nodes can handle, based on the default VM flavors.

The compute nodes, as well as controllers and storage devices for the proposed environments, are Dell PowerEdge T320 servers, with Intel Xeon E5-2420, six physical cores, and 12 threads, 1 TB of storage and 24 GB RAM. Another relevant information is that OpenStack default hypervisor (KVM) supports a virtual core per physical core or thread available. Table 5 presents how many VMs each node can handle.

Table 5 Virtual Machine Flavors

Flavors	vCPU	vRAM	VM per Node
m1.tiny	1	512MB	12
m1.small	1	2GB	12
m1.medium	2	4GB	6
m1.large	4	8GB	3
m1.xlarge	8	16GB	1

The node component in the baseline, double redundancy and HA triple redundant environments have higher availability because Ceph distributed storage service is an exclusive component of the hyper-converged architecture. Yet, they have a similar COA value. Table 6 presents the obtained values of COA through analytical evaluation and simulation.

The analytical evaluation of the COA proposed model results in 0.9948 for a node hosting up to six virtual machines; for twelve or more virtual machines, a state explosion occurs, leading us to 2^{18} states. For 12 VMs (Tiny or Small flavors), a simulation (Sim.) showed with 95% of confidence and a standard error of 4.11×10^{-4} resulted in a COA of 0.994676. The total Capacity (COA $\times$ Expected Amount) for each architecture, with an expected amount for each node of 12 virtual machines of small or tiny flavors, can be seen in Table 7.

7.3 Case Study III - Cost evaluation

A three stage survey characterizes the cost analysis. At the first stage of the survey, the necessary components to provide a cloud computing environment were defined, such as servers (variable number for the baseline, double redundant, HA triple redundant and hyper-converged architectures), Switch, rack and air conditioning (fixed amount).

On survey's second stage all the acquisition costs for each component were individually analyzed, by consulting Brazilian and North American websites, the lowest value of each architecture element in the period from 6th to April 13th, 2016 was selected.

The third and last stage of our survey was the technical specification analysis of each environment component, in search of energy power consumption values to calculate the energy cost per unit for one year period. Each architecture would be running for 24 hours a day, seven days a week. Furthermore, the kilowatt-hour

Table 6 COA evaluation

VMs	CoA (Analytical)	CoA (Sim.)	Error	Confidence Interval
1	0.994807	0.994591	6.83×10^{-4}	[0.9932, 0.9959]
3	0.994807	0.994605	6.02×10^{-4}	[0.9934, 0.9958]
6	0.994807	0.994551	4.53×10^{-4}	[0.9936, 0.9954]
12	State Explosion	0.994676	4.11×10^{-4}	[0.9938, 0.9954]

Table 7 Capacity for each architecture

Architecture	Capacity (VMs/node)
Baseline	11.93
Double Redundant	23.87
HA Triple Redundant	35.80
HA hyper-converged	35.80

cost in Recife, Brazil, in March 2016 was 16 cents of American Dollar. With the collected data the amount of consumed kilowatt-hour (kWh) by each component in one year was evaluated by Equation 5.

$$kWh = \frac{Power(W) \times NHD \times NDY}{1000} \quad (5)$$

where NHD is the number of hours per day and NDY is the number of days per year that the equipment was operational. Table **??** shows the relationship between components and their acquisition costs, power and annual energy costs.

Table **??** presents the overall acquisition costs for each proposed architecture, where the column named as the total value is the sum of acquisition and electricity costs for each cloud computing infrastructure component.

The architecture with the highest cost is the HA triple redundant, which, for its implementation and operation over a year presents an expense of 14,699.88 dollars, 8,199.98 dollars greater than the architecture with the lowest cost, the baseline.

It is important to mention that both architectures, double redundant and the hyper-converged one, have similar deployment costs, with both containing six machines in their composition. However, as previously stated, the availability of each one is much different. While the annual downtime of the architecture with double redundancy is close to 2 hours, the yearly downtime of the hyper-converged environment is less than 2 minutes. This result implies that hyper-convergence is of better use than double redundancy.

8 Future Works and Final Remarks

This paper proposed and evaluated the availability, capacity-oriented availability and the deployment cost of computational environments based on the OpenStack cloud computing platform. By modeling these environments hierarchically, through Dynamic Reliability Block Diagrams (DRBDs) and Reliability Block Diagrams (RBD), we were able to represent the behavior of each proposed architecture and obtain values, as well as artifacts to help the cloud computing stakeholders. With this, both clients and providers can host their services with high availability and lower financial loss, due to breaches in the Service Level Agreements (SLA).

The obtained results show that it is better to provide a hyper-converged environment than a double redundant one, since both architectures have the same number of components, but an annual downtime that differs in more than sixty times. Besides, the hyper-converged environment proves to be an alternative to the traditional highly available triple redundancy on the controller, node, and storage devices, with an annual downtime of less than five minutes as well, but with 33% fewer components and for 72% of the price of the HA triple redundant deployment.

As future works, we intend to provide performability studies and evaluate the impact of reading and writing operations on shared storage devices, as well as determine whether this is in fact the system's bottleneck on service provisioning.

References

[1] Red hat ceph storage architecture and administration. Red Hat. Available in: *http : //gs.statcounter.com/*.

[2] A. Avižienis, J.C. Laprie, B. Randell, and University of Newcastle upon Tyne. Computing Science. *Fundamental Concepts of Dependability*. Technical report series. University of Newcastle upon Tyne, Computing Science, 2001.

[3] Algirdas Avizienis, Jean Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.

[4] A. C. Azagury, R. Haas, D. Hildebrand, S. W. Hunter, T. Neville, S. Oehme, and A. Shaikh. Gpfs-based implementation of a hyperconverged system for software defined infrastructure. *IBM Journal of Research and Development*, 58(2/3):6:1–6:12, March 2014.

[5] M. C. Bezerra, R. Melo, J. Dantas, P. Maciel, and F. Vieira. Availability modeling and analysis of a vod service for eucalyptus platform. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 3779–3784, Oct 2014.

[6] I. Costa, J. Araujo, J. Dantas, E. Campos, F. A. Silva, and P. Maciel. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Journal Quality and Reliability Engineering International*, 2015.

[7] Jamilson Dantas, Rubens Matos, Jean Araujo, and Paulo Maciel. Models for dependability analysis of cloud computing architectures for eucalyptus platform. *International Transactions on Systems Science and Applications*, 8:13–25, 2012.

[8] S. Distefano and Liudong Xing. A new approach to modeling the system reliability: dynamic reliability block diagrams. In *RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, pages 189–195, Jan 2006.

[9] P. M. Frank. *Introduction to Sensitivity Analysis*. Academic, 1978.

[10] Sachin Garg, Puliafto A, Telek M, and Kishor S. Trivedi. Analysis of software rejuvenation using markov regenerative stochastic petri net. In *Proc. In: Sixth International Symposium on Software Reliability Engineering, (ISSRE'95)*, pages 180–187, Paderborn, 1995.

[11] Michael Haag. *Hyper-Converged Infrastructures for DUMMIES*. John Wiley & Sons, Inc., 2016.

[12] DI Heimann, N Mittal, and KS Trivedi. Dependability modeling for computer systems. In *Reliability and Maintainability Symposium, 1991. Proceedings., Annual*, pages 120–128. IEEE, 1991.

[13] D. S. Kim, F. Machida, and K. S. Trivedi. Availability modeling and analysis of a virtualized system. In *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on*, pages 365–371, Nov 2009.

[14] P. Maciel, R. Matos, B. Silva, J. Figueiredo, D. Oliveira, I. F, R. Maciel, and J. Dantas. Mercury: Performance and dependability evaluation of systems with exponential, expolynomial, and general distributions. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 50–57, Jan 2017.

[15] M. Malhotra and K.S. Trivedi. Power-hierarchy of dependability-model types. *Reliability, IEEE Transactions on*, 43(3):493–502, Sep 1994.

[16] RDS Matos, Paulo Romero Martins Maciel, Fumio Machida, Dong Seong Kim, and Kishor S Trivedi. Sensitivity analysis of server virtualized system availability. *Reliability, IEEE Transactions on*, 61(4):994–1006, 2012.

[17] Rubens Matos, Jean Araujo, Danilo Oliveira, Paulo Maciel, and Kishor Trivedi. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Elsevier Journal Simulation Modelling Practice and Theory*, 50:151–164, January 2015.

[18] C. Melo, J. Dantas, J. Araujo, and P. Maciel. Availability models for synchronization server infrastructure. In *Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'16)*, Budapest, Hungary, 2016.

[19] C. Melo, R. Matos, J. Dantas, and P. Maciel. Capacity-oriented availability model for resources estimation on private cloud infrastructure. In *The 22nd IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'17)*, Christchurch, New Zealand, 2017.

[20] Danilo Oliveira, André Brinkmann, and Paulo Maciel. Advanced stochastic petri net modeling with the mercury scripting language. In *ValueTools 2017, 11th EAI International Conference on Performance Evaluation Methodologies and Tools*, December 2017.

[21] OpenStack. OpenStack Cloud Software. `http://www.openstack.org/software//`, 2013. [Online; accessed 19-December-2014].

[22] Rivalino Matias Paulo Maciel, Kishor Trivedi and Dong Kim. Dependability modeling. In *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, 2011.

[23] V. Shankar and R. Lin. Performance study of ceph storage with intel cache acceleration software: Decoupling hadoop mapreduce and hdfs over ceph storage. In *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 10–13, June 2017.

[24] ITEM Software. Reliability block diagram. http://www.reliabilityeducation.com/rbd.pdf, 2007. [Online; accessed 26-September-2015].

[25] Y. Taniguchi, K. Suganuma, T. Deguchi, G. Hasegawa, Y. Nakamura, N. Ukita, N. Aizawa, K. Shibata, K. Matsuda, and M. Matsuoka. Tandem equipment arranged architecture with exhaust heat reuse system for software-defined data center infrastructure. *IEEE Transactions on Cloud Computing*, 5(2):182–192, April 2017.

[26] Abhijith. U, M. Taranisen, A. Kumar, and L. Muddi. The efficient use of storage resources in san for storage tiering and caching. In *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*, pages 118–122, Jan 2016.

[27] H. Xu, L. Xing, and R. Robidoux. Drbd: Dynamic reliability block diagrams for system reliability modelling. *International Journal of Computers and Applications*, 31(2):132–141, 2009.

[28] Yan Zhang, Yongbin Wang, Manjun Zhang, and Bo Lu. Vdc embedding scheme based on virtual nodes combination in software defined data center. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 931–935, Oct 2016.