Availability Models for Hyper-converged Cloud Computing Infrastructures

Carlos Melo*, Jamilson Dantas*, Andre Oliveira*, Danilo Oliveira*, Iure Fé*,
Jean Araujo†‡, Rubens Matos§ and Paulo Maciel*
*Informatics Center, Federal University of Pernambuco, Recife, Brazil
†Academic Unit of Garanhuns, Federal Rural University of Pernambuco, Garanhuns, Brazil
‡NOVA LINCS & DI, FCT, Universidade NOVA de Lisboa, Caparica, Portugal
§Institute of Education, Science, and Technology of Sergipe, Lagarto, Brazil
{casm3, jrd, apbo, dmo4, isf, prmm}@cin.ufpe.br*, jean.teixeira@ufrpe.br†, rubens.matos@gmail.com§

Abstract—**The software-defined data center (SDDC) concept replaces the older, scattered and disorganized silo-based architecture. The SDDC adoption implies the reduction of expenses with staff management and the infrastructure improvement, which becomes easier-to-keep, due mainly to the possibility to virtualize all these computational resources. Also, due to the components standardization, the required physical space to store, provide and maintain these environments have decreased. But a problem still remains, how to keep the system and service provisioning working under a stringent Service Level Agreement (SLA)? High Available (HA) environments have an annual downtime of fewer than five minutes; this is the closest that one can reach from a 100% of availability, usually many layers of redundancy are required to achieve this value. This paper evaluates the hyper-convergence, one of the ways to reach HA in cloud computing environments, but with fewer components than typical architectures. The hyper-convergence on OpenStack cloud computing platform utilizes a distributed storage mechanism to reduce the expenses with storage devices. To the proposed environments evaluation, some behavioral models were created, these models show how advantageous are distributed storage systems over typical ones. The availability results pointed out that the full triple redundancy environment availability values are pretty close to the ones extracted from a hyper-convergent environment even with 33% more components.**

I. Introduction

The need for standardization of data centers' hardware and software components has to lead us to evolution, mainly in the concepts of service provisioning and client-server architectures. What was once disorganized and endowed with structural problems became concise and distributed in racks that consumes less physical space than the so-called silo-based environment.

Today is possible virtualizing beyond processing, but also network and storage in a single rack, or even in a single machine. Data centers that follow this type of architecture are called hyper-convergent. The hypervisor is the source to the *hyper* therm, the International Data Corporation (IDC) predicts the market for the hyper-converged environments will experience a 60% annual growth until 2019 with 3.9 billion dollars on sales [1], [2].

This paper has as the main objective the availability evaluation of a hyper-converged environment through availability models. The service evaluated is the object storage, which is distributed over the OpenStack platform's compute nodes; we compare the hyper-convergence results with the obtained from a baseline environment containing the minimal OpenStack configuration and with a full double, and triple redundant components with external storage devices.

The remainder of this paper is organized as follows. Section III presents an introduction to dependability and behavioral models are explained. The Section II shows the related works and how they differ from ours. Already in Section IV the hyper-convergence in private clouds is presented. Section V shows the availability models which represents the proposed architectures. The Section VI models viability is presented in a study case. Finally on Section VII shows the final remarks about the main results obtained by availability models and cost evaluation, also our future works.

II. Related Works

This Section presents the works that underlie this paper. In [3] the hyper-convergence in computing and storage environments are introduced. Also, they evaluate the performance and the escalability characteristics of these architectures through a prototype implementation and deployment of the IBM General Parallel File System (GPFS).

While in [4] an optimized algorithm is proposed, which follows the hyper-convergence as a mean to maximize the utilization of higher tier (Tier 0) storage. The performance evaluation of Solid State Drives (SSD) over Serial Advanced Technology Attachment (SATA) (Tiers 1, 2, 3...), shows the write and read capacity of these components.

In [5] the proposition of an energy-efficient architecture for SDDC infrastructures is made, the authors evaluate the heat reuse system through simulation. Already in work done by [6] the software-defined network as a mean to minimize the cost and maximize the revenue of data center resources.

The main differences between the work done here and the related ones are the proposition and evaluation of combinatorial models for availability analysis of traditional and hyper-convergent cloud computing infrastructures, based on the OpenStack platform.

III. Background

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

A. Dependability Evaluation

Dependability is the ability of a computer system to provide services that can be both: justifiably and trusted [7]. This definition considers the user's perception about the system 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 [8].

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 in a moment in the future. Steady state availability, also called the long-run availability, is the limit of the availability function as time tends to infinity, which 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 find through MTTR = $MTTF \times (\frac{UA}{A})$, where UA is the acronym for the system Unavailability, reached through $1-A$. Equation 1 presents how the system availability was calculated in this paper.

$$A = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

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

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 components.
- State-space models represent the system behavior (failures and repair activities) by its states and event occurrence expressed as rates or distribution functions. These models allow representing more complex relations between components of the system than combinatorial models do.

Reliability Block Diagram (RBD) and Fault Tree (FT) are among the most prominent combinatorial modeling formalism, whereas Stochastic Petri Nets (SPN), Continuous Time Markov Chain (CTMC), and Stochastic Automata Networks are widely used state-space modeling formalism [9], [10], [11], [12].

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

1) Petri Nets (PN): were proposed by Carl Adam Petri in 1962 [13]. The main components of a Petri Net are the places, transitions, arcs and tokens. The places are the reachable states of the system, transitions are the events realized by the system, the arcs connects a place to a transition or vice-versa while the tokens showed the available or unavailable resources to the system or the place [14].

2) RBDs: enables the analysis of system's availability and reliability features, also, with them It is possible to identify which is/are the most critical system's component(s) [15]. They can describe the interaction between system's components through blocks and lines. Each block may represent a single component or sub-components that belong to a larger system.

However, despite the high abstraction level, with RBDs, the priority between system's component is not possible to be modeled [10] and to model more complex systems, a most potent modeling tool is required, such as Dynamic Reliability Block Diagram (DRBD) and Petri Nets.

3) DRBDs: are an extension from the traditional RBDs that considers dependencies on dynamic systems, priority between repairs, and resource sharing [16]. Besides if the modeled system is too big to be shown in SPN or the SPN relations (arcs and transition annotations) are too complex so that its graphical notation loses its visual appeal, the DRBD becomes an alternative. DRBD's main attributes are the modeling constructs: SDEP (state dependency) block and SPARE (spare component) block. In this work, DRBDs are mapped into SPNs, as in Figure 1 that represents the SDEP block behavior.

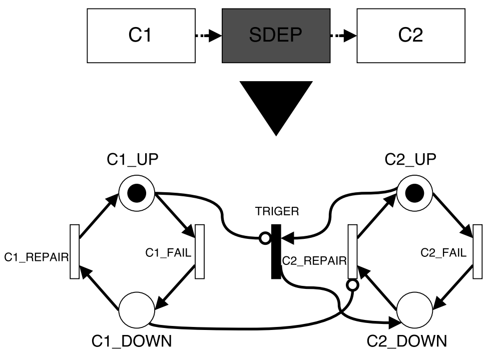

Fig. 1. SDEP block conversion to SPN

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. Which can be seen in Figure 1.

B. Sensitivity Analysis

To determine the most influential factors on interest metrics Sensitivity analysis techniques are helpful tools [17]. 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 2 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, so 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 $S_\theta(Y(\theta))$ [18].

$$S_\theta(Y) = \frac{max\{Y(\theta)\} - min\{Y(\theta)\}}{max\{Y(\theta)\}} \tag{2}$$

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 be found.

IV. 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 [1] based ones (disorganized and nonstandard) are the possibility to provide services with high availability, less components and at 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, and where each silo represents purpose-built hardware with It's own specialized software and need to be managed by dedicated specialists [1].

On silo-based non-redundant architectures, a lot of point of failures are present, since each component is a point of failure, which means that if any of them enters into a failure state all the provided service stops working. That's why redundant converged and hyper-convergent environments aroused, achieving better availability and reliability results through architectures with no single point of failure.

A. OpenStack Platform

OpenStack [19] is the cloud computing platform chosen for this work, due to its ability to adapt to plugins for the provision of a hyper-convergent infrastructures with Ceph [20]. The OpenStack controls pools of compute, storage, and networking resources throughout a data center [19]. It's main components are listed below.

- Keystone is the identity service used by OpenStack for security, authentication and high-level authorization for each of the OpenStack components [19];
- Nova is the component responsible to provide computing power on OpenStack cloud computing environment [19]. Nova manages the hypervisor to provide virtual machines;
- Neutron (Networking) implements OpenStack's "network as a service" model and Its an OpenStack core component, which means that it must be present in all infrastructures [19];
- Glance is the OpenStack image service [19] that provides a mean to users upload and discover data assets;
- Swift is the OpenStack Object Store project that offers cloud storage software for users store and retrieve personal data [19].
- RabbitMQ is the broker software that implements the Advanced Message Queuing Protocol (AMQP). Its main function is to coordinate operations and status information among services running [19].
- Database: most of the OpenStack services uses an SQL database to store their information. The default OpenStack SQL database is the MariaDB [19].
- Hypervisor is the software responsible to provide hardware virtualization and provide virtual machines. The OpenStack default hypervisor is the Kernel-based Virtual Machine (KVM) [19].

The Figure 2 presents how the OpenStack main components on a traditional architecture are organized: Controller, Compute node and Storage.

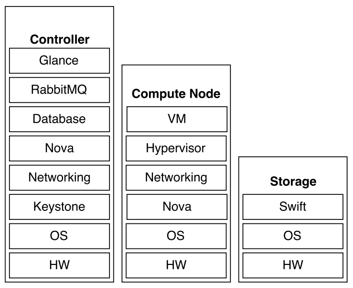

Fig. 2. OpenStack service Layout

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 [19]. This new organization means a change in the OpenStack software stack, as presented in Figure 3, which shows how the service layout into a Hyper-convergent OpenStack architecture.

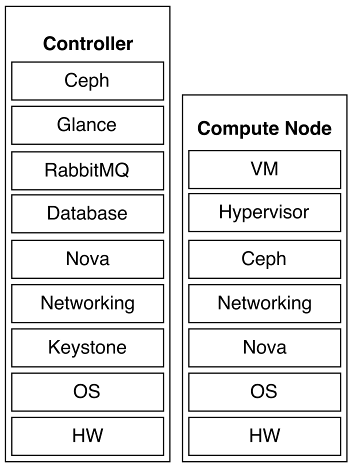

Fig. 3. OpenStack Service Layout with Hyper-convergence

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 [19]. This new organization means a change in the OpenStack software stack, as presented in Figure 3, which shows how we can deploy the services and applications into a Hyper-convergent OpenStack architecture.

V. Availability Models

Dynamic Reliability Block Diagrams (DRBDs) were used to represent the relationship between the hardware and the OpenStack cloud computing components.

A. Baseline Architecture

A DRBD represents OpenStack controller, with some dependencies as shown in Figure 4. The controller machine as shown in Figure 2 contains a 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 if the hardware element enters into a failure state all the 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. Dynamical Reliability Block Diagrams works with this concept: Priority and Shared resources. The SDEP block establishes that one thing must be repaired first then other, in the previous example the operating system only can 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 no one can manage the platform anymore since Neutron, Nova and Glance manager 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, to manage the OpenStack normally, all these components must be working.

For the compute node, another DRBD is proposed and presented in Figure 5. This DRBD consists of hardware (HW), operating system (OS), Nova Compute, Neutron networking service, hypervisor and Virtual Machine.

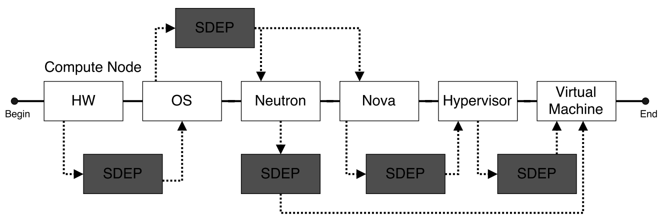

Fig. 5. DRBD for Compute node

This DRBD describes the dependencies between node's components, as previously presented if the hardware enters into a failure state then all software components goes down as well, as for the operating system it can take with it all services running at the time: Neutron, Nova, Hypervisor and Virtual Machine. If the hypervisor fails, the Virtual Machine goes down, the same for the 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 more virtualized resources. The OM that 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 on Figure 6, and contains hardware (HW), operating system (OS) and the Swift object storage. The OM for this DRBD represented by the relationship between HW $\wedge$ OS $\wedge$ Swift, all these components must be available to provide the storage service.

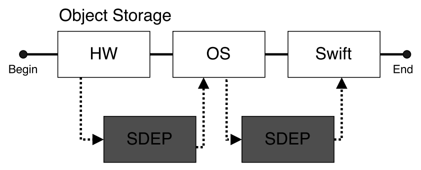

Fig. 6. DRBD for storage

By combining the three DRBDs that represents the baseline architecture we have the baseline RBD with three blocks, each one is a component subsystem as in Figure 7. All the three blocks must be operational in the OM: Controller $\wedge$ Compute $\wedge$ Storage.

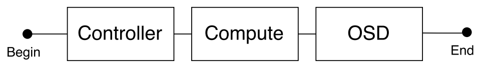

Fig. 7. RBD for baseline architecture

B. Double Redundant Architecture

In the baseline scenario, if any of the cloud components fails the system will not be available anymore, also, 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 in Figure 8.

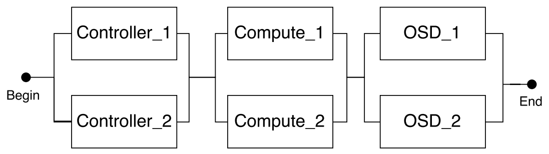

Fig. 8. RBD for double redundancy architecture

In this second RBD, all components previously presented have an active-active redundancy, sharing all the work and

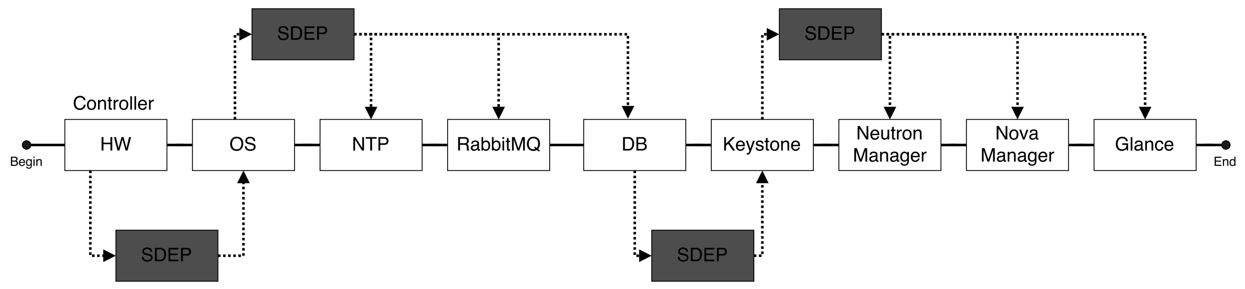

Fig. 4. DRBD for Controller

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, if two of the same component stop to working then the entire system becomes unavailable. The logic function that describes the OM of the double redundant environment is expressed by: (Controller 1 $\vee$ Controller 2) $\wedge$ (Compute 1 $\vee$ Compute 2) $\wedge$ (OSD 1 $\wedge$ OSD 2).

C. High Availability Models for triple redundancy environment

A high available environment has an annual downtime fewer than five minutes. Figure 9 presents the RBD that represents this environment with this availability.

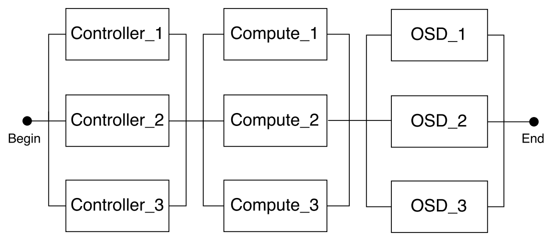

Fig. 9. RBD for High Available triple redundancy environment

A triple redundancy mode is necessary to produce availability values lower than five minutes, 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. Each of the blocks has DRBD as their subsystem. The logic function that describes the OM for the triple redundant environment is expressed by: (Controller 1 $\vee$ Controller 2 $\vee$ Controller 3) $\wedge$ (Compute 1 $\vee$ Compute 2 $\vee$ Compute 3) $\wedge$ (OSD 1 $\wedge$ OSD 2 $\wedge$ OSD 3).

D. High Availability for Hyper-convergent Architecture

To turn an OpenStack cloud computing environment onto a hyper-convergent one, we must provide a way to maintain 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 it, the Ceph has a monitor on the controller, which implies in some modifications to the DRBD presented on Figure 4, as shown in Figure 10. This new OM is expressed by HW $\wedge$ OS $\wedge$ Ceph_Monitor $\wedge$ NTP $\wedge$ RabbitMQ $\wedge$ Nova $\wedge$ Glance $\wedge$ Neutron.

The Ceph monitor is responsible for providing control over the Ceph on compute nodes, and analyze where virtual machine's object data is going and where it is if a compute node fails, it main dependency is the hardware, as any of applications already presented. The Figure 11 shows the DRBD for compute node, now containing Ceph Object Storage. The OM that describes this DRBD is HW $\wedge$ OS $\wedge$ Ceph $\wedge$ Neutron $\wedge$ Nova $\wedge$ Hypervisor, if all these components are available, then virtual machines can be provided.

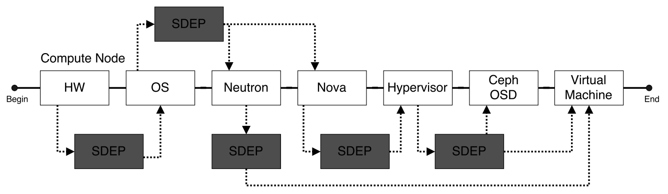

Fig. 11. DRBD for Node with Ceph Object Storage

The hypervisor controls the Ceph object storage. For the hyper-convergent environment, we propose another RBD in Figure 12.

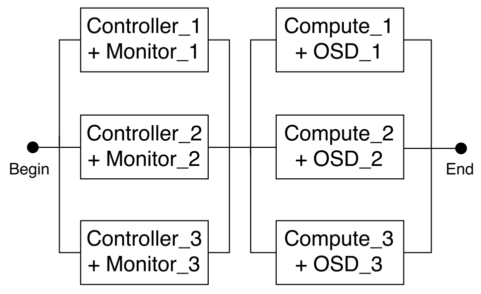

Fig. 12. RBD for High Available hyper-convergent architecture

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, with 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

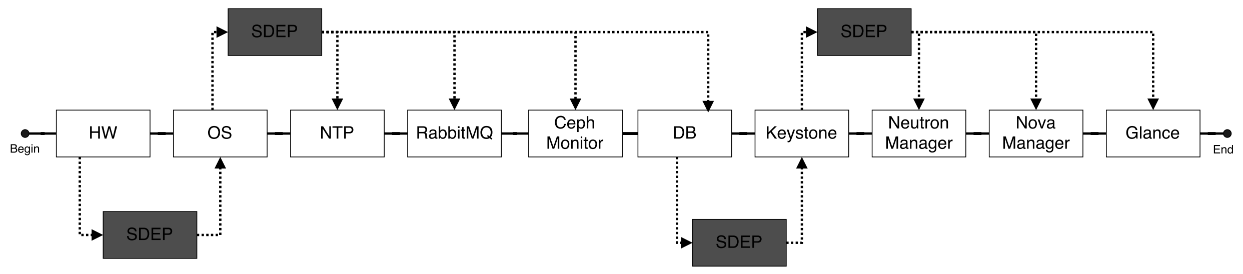

Fig. 10. DRBD for Controller with Ceph Monitor

1 ∨ Controller_Monitor 2 ∨ Controller_Monitor 3) ∧ (Compute_OSD 1 ∨ Compute_OSD 2 ∨ Compute_OSD 3).

VI. Case Study

This section provides the case study that demonstrates the feasibility of the proposed models. A list of input values are needed to evaluate our models; Those are in Table I.

TABLE I
INPUT VALUES FOR CLOUD COMPONENTS

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

The models that represent the baseline architecture were the first ones to be evaluated through Mercury Tool, which made possible to extract the availability values for each DRBD sub-model (Controller, Compute Node and Storage) by inserting each presented input values on the corresponding block. Table II shows the availability results for each sub-model and for the cloud RBD model.

TABLE II
COMPONENTS AVAILABILITY RESULTS

Model	Av. (%)	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 pointed out an availability of 97.78% for the cloud baseline RBD, meaning an annual downtime higher than 194 hours, that is, for the 365 days of one year the system will be unavailable for eight entire days! But, what we want to know is which component was the responsible for this massive downtime? Since each subsystem has it's own components, and each one has an impact on the subsystem availability consequently an impact on the availability of the entire system, the answer to this question is reach by sensitivity analysis technique.

The sensitivity analysis technique evaluates the MTTF and MTTR for each component on the platform (see Table I). The input values for sensitivity analysis are chosen based on the rates and times between failures and repairs for each system component. Each parameter varied between +50% and -50% of its original value; this measure was considered adequate to cover an upper and lower limit for the variation of architecture availability. The Table III shows the rank by sensitivity index.

TABLE III
SENSITIVITY RANKING FOR BASELINE

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
$MTTR_{glance}$	5th	2.33E-03
$MTTR_{nova\_manager}$	-	-
$MTTR_{neutron\_manager}$	-	-
$MTTR_{keystone}$	9th	2.11E-03
$MTTF_{hypervisor}$	10th	1.72E-03
$MTTF_{ntp}$	11th	1.69E-03
$MTTF_{rabbittmq}$	-	-
$MTTF_{neutron}$	-	-
$MTTF_{swift}$	-	-
$MTTF_{glance}$	15th	1.68E-03
$MTTF_{nova\_manager}$	-	-
$MTTF_{neutron\_manager}$	-	-
$MTTF_{OS\_Controller}$	18th	1.65E-03
$MTTR_{swift}$	19th	1.53E-03
$MTTR_{nova}$	20th	1.48E-03
$MTTF_{neutron}$	21th	1.32E-03
$MTTR_{rabbitmq,ntp}$	22th	1.22E-03
$MTTR_{OS\_Computer}$	23th	1.16E-03
$MTTF_{HW\_Controller}$	24th	7.66E-04
$MTTF_{HW\_Computer}$	25th	6.10E-04
$MTTF_{OS\_Storage}$	26th	5.75E-04
$MTTF_{HW\_Storage}$	27th	4.25E-04
$MTTR_{db}$	28th	3.66E-04
$MTTR_{OS\_Controller,OS\_Storage}$	29th	1.02E-04

The component with the more significant impact on the baseline architecture availability is the Keystone MTTF, ranked at the top, the failure of the Keystone imply on the failure of Neutron Manager, Nova Manager, and Glance since no one can access these services anymore.

By knowing which components impacts the most on the infrastructure availability, we may apply for redundancies on them to improve the availability values obtained. An active-active redundancy model was proposed, as presented in Figure

8; this RBD contains two controllers, two compute nodes and two storages. Also, the hyper-converged environment evaluation shows similar availability result with fewer items.

Since most of the ranked components are in the same interval (E-03) we need to improve all machines to advance on our analysis and evaluate the high available models. The Table IV shows the comparison of the availability results.

TABLE IV
COMPARISON OF AVAILABILITY RESULTS

Architecture	Avail. (%)	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-converged	99.9995	0.04 (2.4 min)

The environments of greater availability are the architectures with nine machines, that is, the triple redundant one. However, the hyper-converged architecture also falls within the requirements to be considered high availability, having an annual downtime of 2.4 minutes or 144 seconds.

Already the organization with the highest downtime and therefore the lowest availability is our baseline architecture, containing only three main components, a controller, a computing node, and storage, it has an annual downtime of approximately 194 hours, much higher than Is intended for a high availability environment.

VII. Final Remarks

This paper presented Dynamic Reliability Block Diagrams (DRBDs) to model the behavior of cloud computing environments based on OpenStack. The evaluation results of four different architectures show that is better to provide a hyper-convergent infrastructure than a full double redundant one, even with the same amount of equipment.

The annual downtime of hyper-convergent and double redundant differs of more than sixty times. Also, the hyper-converged environment proves to be an alternative to traditional high available triple redundancy on the controller, node, and storage, with an annual downtime of fewer than five minutes as well, but with 33% fewer components.

The proposed models may help enterprises, and cloud administrators on service delivering. As a future work, we intend to evaluate traditional high available environments and provide a deployment cost evaluation. Also, a performance evaluation of the proposed architectures can help one to prove it is there a bottleneck on service provisioning in this kind of environment.

References

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

[2] I. D. Corporation. Worldwide hyperconverged systems 20152019 forecast. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=253267

[3] A. C. Azagury, R. Haas, D. Hildebrand, S. W. Hunter, T. Neville, S. Oehme, and A. Shaikh, "Gpfs-based implementation of a hyper-converged system for software defined infrastructure," *IBM Journal of Research and Development*, vol. 58, no. 2/3, pp. 6:1–6:12, March 2014.

[4] A. 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)*, Jan 2016, pp. 118–122.

[5] 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*, vol. 5, no. 2, pp. 182–192, April 2017.

[6] Y. Zhang, Y. Wang, M. Zhang, and B. 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)*, Oct 2016, pp. 931–935.

[7] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 11–33, 2004.

[8] A. Avižienis, J. Laprie, B. Randell, and U. of Newcastle upon Tyne. Computing Science, *Fundamental Concepts of Dependability*, ser. Technical report series. University of Newcastle upon Tyne, Computing Science, 2001. [Online]. Available: https://books.google.com.br/books?id=cDkmGwAACAAJ

[9] M. Malhotra and K. Trivedi, "Power-hierarchy of dependability-model types," *Reliability, IEEE Transactions on*, vol. 43, no. 3, pp. 493–502, Sep 1994.

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

[11] R. M. Paulo Maciel, Kishor Trivedi and D. Kim, "Dependability modeling," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, 2011.

[12] S. Garg, P. A, T. M, and K. S. Trivedi, "Analysis of software rejuvenation using markov regenerative stochastic petri net," in *Proc. In: Sixth International Symposium on Software Reliability Engineering, (ISSRE'95)*, Paderborn, 1995, pp. 180–187.

[13] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr 1989.

[14] P. Maciel, R. Lins, and P. Cunha, *Uma Introducao as Redes de Petri e Aplicacoes*. Sociedade Brasileira de Computacao, 1996.

[15] K. S. Trivedi, S. Hunter, S. Garg, and R. Fricks, "Reliability analysis techniques explored through a communication network example," 1996.

[16] S. Distefano and L. Xing, "A new approach to modeling the system reliability: dynamic reliability block diagrams," in *RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, Jan 2006, pp. 189–195.

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

[18] R. Matos, J. Araujo, D. Oliveira, P. Maciel, and K. Trivedi, "Sensitivity analysis of a hierarchical model of mobile cloud computing," *Elsevier Journal Simulation Modelling Practice and Theory*, vol. 50, pp. 151–164, Jan. 2015.

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

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