Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 26, Num. 3 (2019) 65-74

RESEARCH ARTICLE

Models to evaluate service Provisioning over Cloud Computing Environments - A Blockchain-As-A-Service case study

Modelos para o provimento de Serviços em Ambientes de Computação em Nuvem - Um estudo de caso aplicado a Blockchain como Serviço

Carlos Melo[1]*, Jamilson Dantas[2], Paulo Pereira[3], Ronierison Maciel[4], Paulo Maciel[5]

Abstract: The strictness of the Service Level Agreements (SLAs) is mainly due to a set of constraints related to performance and dependability attributes, such as availability. This paper shows that system's availability values may be improved by deploying services over a private environment, which may obtain better availability values with improved management, security, and control. However, how much a company needs to afford to keep this improved availability? As an additional activity, this paper compares the obtained availability values with the infrastructure deployment expenses and establishes a cost × benefit relationship. As for the system's evaluation technique, we choose modeling; while for the service used to demonstrate the models' feasibility, the blockchain-as-a-service was the selected one. This paper proposes and evaluate four different infrastructures hosting blockchains: (i) baseline; (ii) double redundant; (iii) triple redundant, and (iv) hyper-converged. The obtained results pointed out that the hyper-converged architecture had an advantage over a full triple redundant environment regarding availability and deployment cost.

Keywords: Availability — Blockchain-as-a-Service — Hyper-converged — Virtualization

Resumo: O rigor nos Acordos de Nível de Serviço (ANS) deve-se a um conjunto de restrições nos principais atributos de desempenho e dependabilidade de sistemas. Este artigo apresenta a avaliação de um desses atributos: a disponibilidade, bem como, uma relação de custo - benefício para o provimento de serviços nesses ambientes. Escolhemos modelagem como método para avaliação, em virtude de seu baixo custo e alto nível de representação. Avaliamos quatro diferentes arquiteturas para o provimento de blockchains como serviço: (i) baseline, (ii) redundância dupla, (iii) redundância tripla, (iv) hiperconvergente. Os resultados obtidos apontaram que a arquitetura hiperconvergente apresenta vantagem sobre a arquitetura com redundância tripla em todos os nós tanto em disponibilidade quanto em custos.

Palavras-Chave: Disponibilidade — Blockchain como Serviço — Hiperconvergência — Virtualização

[1,3,4,5] *Federal University of Pernambuco, Recife, Pernambuco - Brazil*
[2] *Federal University of Vale of São Francisco, Salgueiro, Pernambuco - Brazil*
****Corresponding author**: casm3@cin.ufpe.br

DOI: http://dx.doi.org/10.22456/2175-2745.91814 • **Received:** 12/04/2019 • **Accepted:** 02/08/2019

1. Introduction

The virtualization technology provided to datacenters and the possibility to host and manage any service over the Internet is what now is known as the cloud computing paradigm. As a result, the cloud became an exciting research environment to refine, test, and develop new technologies, mainly because its fast adaptation and development. These features enable the processing, network and storage virtualization into a single rack, or even into a single machine [1]. The datacenter that follows this full virtualization technology is called software-defined data center (SDDC) [1] and replaces the older, scattered and disorganized silo-based architectures.

There are two main roads to reach SDDC: convergence and hyper-convergence [1]. In both cases, the environments converge, but in the second one, computing and storage services are hosted into a single node, while the first one has different hardware for each service. The IDC[1] predicted a growth of 60% of the hyper-converged market until 2019, with 3.9 billion dollars on sales [1]. However, what is the benefit of a hyper-converged infrastructure over a converged one? The answer to this question is one of the answers that this paper

[1] IDC: https://goo.gl/NA7E6p

provides.

In this paper, we analyzed a set of environments through availability models. These models are artifacts that may help companies and stakeholders to predict and mitigate the impact of some issues over an SLA fulfillment [2]. Also, these models can be deployed hierarchically to improve system's availability through redundancy mechanisms.

As a mean to demonstrate the proposed models' feasibility, we evaluate four different architectures with two case studies: (1) hosting a service; and (2) cost-benefit evaluation. The blockchain-as-a-service was the chosen service due to the increasing adoption of the blockchain paradigm, which is a result of the cryptocurrencies valorization and their security improvement mechanisms that may change the future of computing [3], while the cost-benefit evaluation is a survey analysis that may help stakeholders to choose to stay in a public cloud computing environment or deploy a private one.

This paper is an extension of [**??**] and [**??**], which has as the primary objective the availability and deployment cost evaluation of four cloud computing environments managed and distributed in different ways. The proposed architectures shared between them the service they are hosting: blockchain-as-a-service.

The key contributions of this work are the following: (1) behavioral models for availability evaluation of four different architectures; (2) System's availability and deployment cost evaluation; and (3) establishment of the cost-benefit relationship for the proposed environments. The paper is organized as follows. Section 2 presents the works that underlie this paper. Already Section 3 describes system's availability, blockchain-as-a-service, and behavioral models. In Section 5, we present the modeling methodology adopted by this paper. The Section 6 shows the availability models that represents the proposed environments. In Section 7, we evaluate the model's feasibility through some case studies. Finally, the Section 8 shows the final remarks about the obtained results and the next steps.

2. Related Works

Over the last few years, hyper-converged environments and blockchain-as-a-service provisioning arose, authors have devoted their efforts to study and evaluate performance metrics related to those themes, in this section, we present the works that underlie this paper. In [**??**], we evaluated the availability of four architectures with no service running over it, and we show the structural advantages of hyper-converged environment adoption. Already [**??**] evaluates the blockchain-as-a-service deployment over a virtualized environment managed by docker.

Other authors, such as [4] combined blockchain storage to construct a personal platform based on privacy. An automatic access-control protocol that enables blockchain management was implemented. Through this mechanism, the transactions, in this case, system's instructions, do not need to be trusted or be reevaluated by third parties. Already in [5], an evaluation of the blockchains' use and smart contracts to facilitate the data sharing in an Internet of Things (IoT) envrionment was made. Also, they pointed out the main issues and the main advantages of the use of IoT and blockchain combination. In [6], a framework called Hawk was implemented, aiming to improve the way that smart contracts work in a blockchain environment by enabling a company or an user to apply to the chain without the need to develop the cryptography.

In [7], the authors investigates the availability of functions for an Ethereum blockchain scenario, they pointed out the time lost by these functions and their general impact in the cryptocurrency transactions. Already in work done by [8], the authors evaluate a blockchain environment and establish the reliability gradient through EPTM models and Circuit Unit Importance.

[9] evaluate Mobile Backend-as-a-service through hierarchical modeling in two different scenarios: baseline, without automatic repair mechanism, obtaining a lower availability than the scenario with automatic repair routine; a sensitivity analysis was applied, aiming to identify which component impacts the most on system's availability. In [10] the author introduced hyper-convergence in computing and storage environments. Also, he evaluated the performance and the scalability characteristics of these architectures through a prototype implementation and deployment of the IBM General Parallel File System (GPFS). While in [11] an optimized algorithm is proposed, which follows the hyper-convergence as a mean to maximize the utilization of Tier 0 storage.

In [12], the proposition of an energy-efficient architecture for SDDC infrastructures is made, the authors evaluate the heat reuse system through simulation. Finally, in [13], the authors define the software-defined network (SDN) as a mean to minimize the expenses and maximize the revenue of data center resources.

This paper evaluate the deployment cost and proposes behavioral models for availability evaluation of four different environments hosting blockchain-as-a-service. To the best of our knowledge, this has not been considered so far.

3. Background

This section presents the fundamental concepts about availability evaluation, system's modeling, blockchain-as-a-service, and OpenStack cloud computing environment. These concepts are useful for a good understanding of this paper.

3.1 Availability Evaluation

Dependability is the capability of a system to deliver a set of trustable services that are observed by outside agents, such as users and administrators [14]. Usually, system's dependability evaluation requires the study of at least one of its six attributes [15]; It is the case of system's availability, which evaluates the probability of a system being available now or in a moment in the future. In this paper, we evaluate the steady-state availability, also called the long-run availability, which is the limit of the availability function as time tends to infinity (Equation 1).

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

As a counterpart to the system's availability, we may highlight the system's downtime; which is measured by the relationship between the obtained availability value and the aimed period (hours, minutes, seconds). For example, the system's annual downtime is $8760h - (8760h \times Availability)$.

We can reach availability values by many ways, which go from measurement to modeling. The second method has as virtues the possibility to provide high-level system's view with great flexibility in adapting parameters and achieving results, besides a lower cost to evaluate, and because of this modeling was the chosen evaluation method. We usually classify dependability models in 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 [16, 17, 14, 18]. RBDs and DRBDs are the modeling formalism chosen to represent the behavior and availability of our proposed system, these models enable us to evaluate an environment precisely by establishing a relationship between software and hardware components. Models are easier to evaluate and costs less than experiments.

The DRBD is an extension of the traditional RBDs that considers dependencies on dynamic systems, priority between repairs, and resource sharing [19] [20]. If the modeled system is too big to an SPN or the SPN relations (arcs and transition annotations) are too complicated, the SPN graphical notation loses its visual appeal, so the DRBD becomes a solution.

The DRBD's attributes are the modeling constructs: SDEP (state dependency) block and SPARE block. In this work, we map DRBDs into SPNs, as in Figure 1 that represents an SDEP block.

The SDEP block can 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. The Figure 1, shows an immediate transition called **TRIGGER**, that moves a token from the place $C2\_UP$ to the place $C2\_DOWN$. 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.

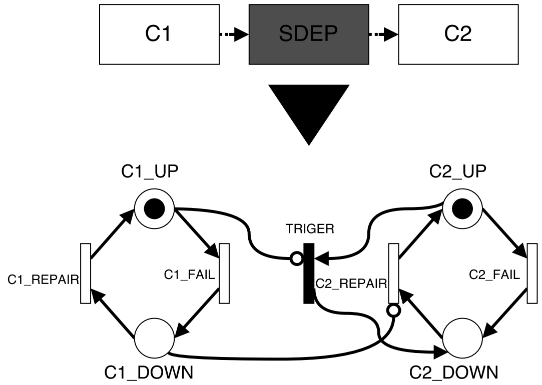

Figure 1. SDEP block conversion to SPN

4. Software Platform Infrastructure

The Hyperledger is an open consortium hosted by Linux Foundation that aims to improve the advances in blockchain technologies[2]. Among the many projects that are hosted by the Hyperledger is the Hyperledger Cello. The Hyperledger Cello is a blockchain module able to provide on-demand blockchain.

Already the OpenStack[3] is the cloud computing platform chosen to provide the service, due mainly to its ability to adapt to plugins for the provision of a hyper-converged infrastructure with Ceph[4].

The OpenStack is an open source platform and a well established environment to test and provide any service through the Internet, one of the main reasons to choose this platform is the Ceph plugin, witch can provide hyper-converged environments, that help to control pools of computing, storage, and networking resources throughout a data center. Below, we describe the main components of the OpenStack platform.

The **Keystone** is the identity service used by OpenStack for security, authentication and high-level authorization for each of the OpenStack components. The **Nova Compute** is the component responsible for providing computing power on OpenStack cloud computing platform. The Nova Compute manages the virtualization service (Docker) to virtual resources provisioning. Already the **Neutron** (Networking) implements OpenStack's "network as a service" model, and it is an OpenStack core component, which means that it must be present in all infrastructures. The broker called **RabbitMQ** implements the Advanced Message Queuing Protocol (AMQP). The **Network Time Protocol** (NTP) keeps time between servers synchronous and the **Database** store services information. All these impacts on the virtual machines (VM) provision.

For the storage service, we may have two different organizations, the first one known as typical contains the OpenStack

[2] Hyperledger Cello: https://goo.gl/7baUzo
[3] OpenStack: https://goo.gl/jNnUe2
[4] Red Hat Ceph: https://goo.gl/vjXW15

Glance, which is the image service that provides image upload and data assets. **Swift** is the Object Store project that offers cloud storage software for data store and retrieves. Already the second organization, characterized by the hyper-convergence environment has the Ceph storage system, which reduces the need for an external storage device or machine, by providing the storage service into the compute node.

The Figure 2 presents how we organize the OpenStack main components of traditional and hyper-converged architectures: Controller, Compute node and Storage.

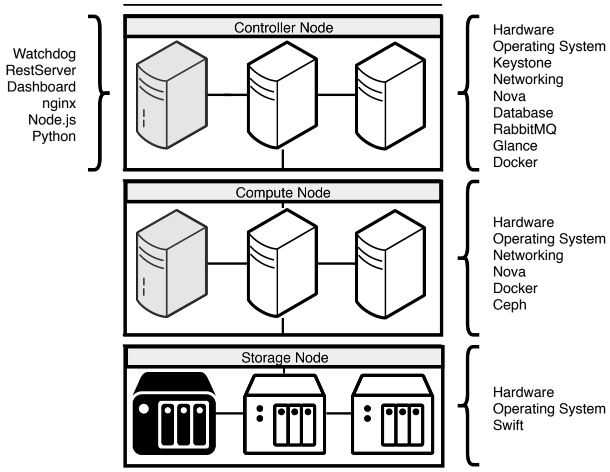

Figure 2. OpenStack Service Layout

The grey and black components represent the minimum amount of resources that must be entirely working to accomplish the service provisioning in traditional architecture. In this paper, we evaluate four architectures; the **baseline architecture** has only one of each of these components, while the **double redundant** has two of each resource, the **triple redundant** has three of each; finally, the hyper-converged environment does not have a storage node, both compute and storage in the same node.

With the Red Hat Ceph and the storage service virtualization approach, it is possible to obtain similar availability values with up to 33% fewer components than a triple full redundant architecture [**??**]. At a hyper-converged environment, the storage services are deployed on the compute nodes, which means that we do not need an external storage device anymore.

Already to host the Hyperledger Cello and manage the blockchains we need the following components to be working: the **Watchdog**, which is the responsible for monitors the blockchain's service and the system's status. The **RestServer** that makes the environment provision, orchestration, and task management. The **Dashboard** provides the environment management for system's administrators. The **nginx** is a reverse proxy used by the Hyperledger Cello to web performance improvement. While the **Node.js** is a lightweight JavaScript runtime used by Cello to improve provisioning. Python runs over the host and provides the means to execute Watchdog, RestServer, and the Dashboard into Controller Node.

Next, we present the followed modeling methodology that enables a person or company to reproduce and improve the proposed models to obtain better availability values aiming the SLA compliance.

5. Modeling Methodology

Survey all the hardware and software resources available and needed to deploy the proposed environments is the first step to accomplish the availability evaluation process. Generally speaking, after listing the required resources and their relationship with the system's deployment, we can create a model that represents the environment's behavior. The Figure 3 shows an organization chart that summarizes the strategy used in this paper.

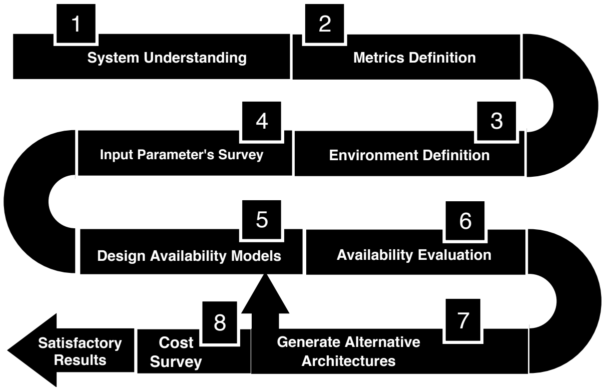

Figure 3. System's Evaluation Methodology

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

2. **Metric Definition**: by knowing how the system works, we can choose the metrics of interest to evaluate; in our case, the system's availability;

3. **Environment Definition**: based on the obtained information from the system understanding step, we defined an architecture containing the minimum necessary components to blockchain-as-a-service delivery, which is called Baseline Architecture;

4. **Input Parameter Survey**: through the baseline scenario we can survey the mean time to failure (MTTF) and mean time to repair (MTTR) of the components and applications inherent to the proposed infrastructure;

5. **Design Availability Models**: in this step it is made the combinatorial and state-based models proposition for availability evaluation based on the architectures behavior and component survey;

6. **Availability Evaluation**: with the availability models that represent the architectures behavior, we can evalu-

ate the system's availability and obtain the values that can be used to propose better alternative architectures;

7. **Generate Alternative Architecture**: from the obtained availability evaluation results from Baseline Architecture we've created three other architectures and classify them as double redundant, converged and hyper-converged; these architectures propositions aim to improve the baseline architecture obtained values, each one has a particularity that is going to be explained later in this paper.

8. **Cost Evaluation**: based in the main components acquisition values in USD, and the proposed architectures alignments, the deployment costs for each environment is surveyed, this value is compared with system's downtime, and provide a cost-benefit relationship evaluation.

The satisfactory results derive from the obtained availability values, and the deployment costs evaluation, which must be consistent with similar parameters of a private and real cloud computing environment.

6. Proposed Models

This section presents the behavioral models and describes the proposed architectures and how the main components are related to each other. The baseline architecture contains the minimum requirements for service deployment: Controller, Compute, and Storage.

The second environment has double redundancy, meaning two controllers, two compute nodes and two storage devices. The third architecture has triple redundancy, which implies into three of each component. Already for the fourth and last environment, we propose an OpenStack Hyper-converged architecture; this architecture unifies the compute node and the storage device into a single machine to reduce expenses.

Dynamic Reliability Block Diagrams (DRBDs) and Reliability Block Diagrams (RBD) were used to represent the relationship between the controller, compute and storage nodes, as well as their hardware and software components of the four proposed environments.

6.1 Baseline Architecture

The OpenStack controller's DRBD model has some dependencies. The controller machine presented in Figure 4, contains HW, an OS, Nova manager, OpenStack subcomponent, Python, NodeJS, nginx, Docker, Dashboard, RestServer, and Watchdog. All these components must be working for service provisioning. The availability function that describes this model is the result of the probability of HW ∧ OS ∧ MongoDB ∧ Keystone ∧ Python ∧ NodeJS ∧ nginx ∧ Docker ∧ Dashboard ∧ RestServer ∧ NTP ∧ RabbitMQ ∧ Nova ∧ Glance ∧ Neutron being operational.

The Network (Neutron), Nova, and Docker are Keystone dependant components, while the NTP server and RabbitMQ broker are vital resources for the environmental management. The Figure 5 shows the compute node DRBD model. The component availability is equal to the probability of HW ∧ OS ∧ Neutron ∧ Nova ∧ Docker being operational.

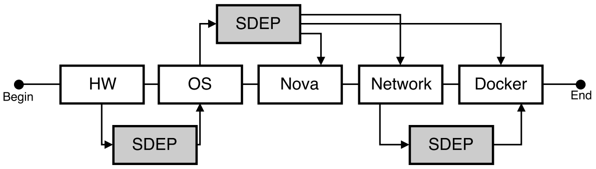

Figure 5. OpenStack Compute Node DRBD model

The last component is the object storage device (OSD), represented by the DRBD in Figure 6, and contains HW, OS and the Swift object storage. The component's availability is equal to the probability of HW ∧ OS ∧ Swift being into an UP state.

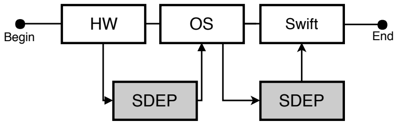

Figure 6. OpenStack Storage DRBD model

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 for service provisioning. All the three blocks must be operational in the OM: Controller ∧ Compute ∧ Storage.

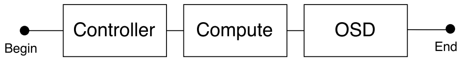

Figure 7. Baseline Architecture RBD

6.2 Double Redundant Architecture

In the baseline scenario, if any of the components fail the service will fail as well. To reduce the unavailability period we propose a model with double redundant in Figure 8. The logic function that describes the operational mode (OM) of the double redundant environment is expressed by: (Controller 1 ∨ Controller 2) ∧ (Compute 1 ∨ Compute 2) ∧ (OSD 1 ∧ OSD 2).

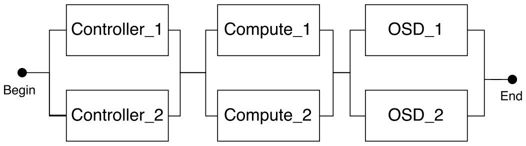

Figure 8. RBD for Double Redundant Architecture

In this second RBD, all components previously presented have an active-active redundancy, sharing all the work 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, because two components imply in a 100% of the resources

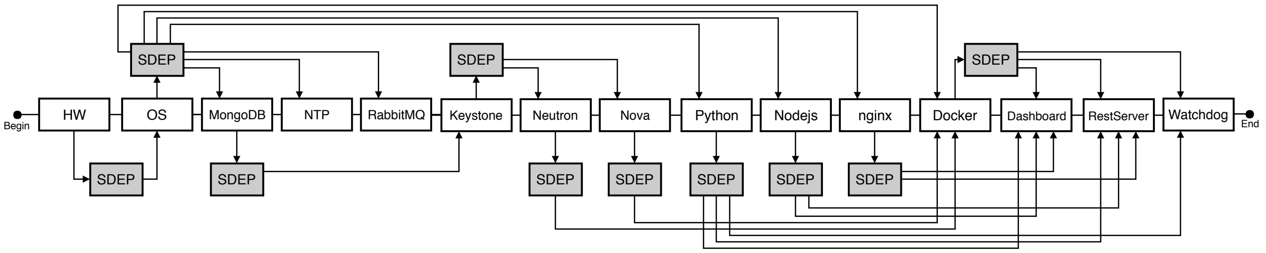

Figure 4. OpenStack Default Controller Node Model

available and a half of it (one machine)imply in at least 50% of the systems entire capacity, if two of the same component stop to working then the entire system becomes unavailable.

6.3 OpenStack Triple Redundant Architecture

The OpenStack converged architecture has nine physical machines, with triple active-active redundancy, this architecture should have no single point of failure. The Figure 9 presents the RBD that represents this environment. 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).

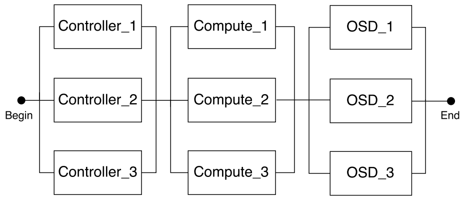

Figure 9. OpenStack Triple Redundant Model

A triple redundant mode must have at least one controller, one compute node, and one storage device available or operational to accomplish service provisioning, as well as the maintenance of the virtual machines, images, and user data.

6.4 Hyper-converged Architecture

To turn an OpenStack cloud computing environment into a hyper-converged one, we must provide a way to maintain the 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 in Figure 10. The component's availability is measured by the probability of HW ∧ OS ∧ MongoDB ∧ Keystone ∧ Python ∧ NodeJS ∧ nginx ∧ Docker ∧ Dashboard ∧ RestServer ∧ Ceph_Monitor ∧ NTP ∧ RabbitMQ ∧ Nova ∧ Glance ∧ Neutron being into an UP state.

The Ceph monitor is responsible for providing control over the Ceph on compute nodes, and analyzing where virtual machine's object data is going and where it is if a compute node fails. It main dependency is the HW, as any of applications already presented. The Figure 11 shows the DRBD for the compute node, now containing Ceph Object Storage. The component availability is equal to the probability of HW ∧ OS ∧ Neutron ∧ Nova ∧ Docker ∧ Ceph being operational.

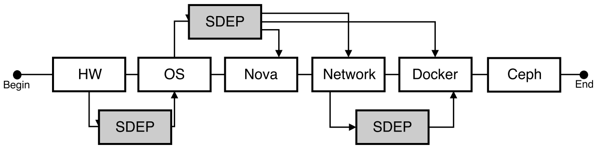

Figure 11. Hyper-converged Compute/OSD

The Docker maintains the Ceph object storage. For the hyper-converged environment, we propose another RBD in Figure 12. The logic function that describes the OM of the hyper-convergent environment is expressed by: (Controller_Monitor 1 ∨ Controller_Monitor 2 ∨ Controller_Monitor 3) ∧ (Compute_OSD 1 ∨ Compute_OSD 2 ∨ Compute_OSD 3).

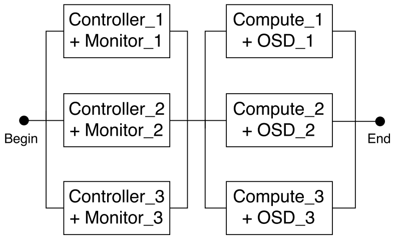

Figure 12. RBD for Hyper-converged Environment

With six blocks, three controllers/monitors and three computing/storage nodes with no single point of failure, and with user's data safe, self-healing mechanism, and 33% fewer components than a full triple redundant converged environment.

7. Case Study

This section provides two case studies that demonstrate how feasible the proposed models are: (1) Availability and (2) Cost-Benefit evaluation of four different architectures. The first thing to accomplish our tasks is to obtain the system's input values to perform the availability evaluation. Those values can be seen in Table 1, they were extracted from [21] [9] [22].

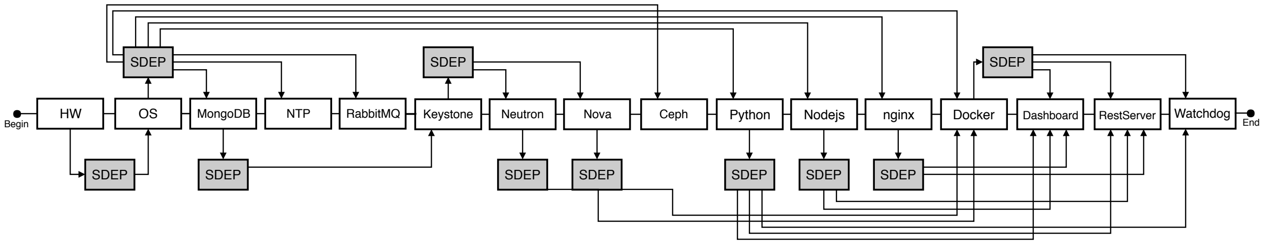

Figure 10. Hyper-converged Controller/Monitor

Table 1. Resources Input values

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
MongoDB	1440 h	20 min
Python, NodeJS, nginx Dashboard, RestServer, Nova, Neutron, Keystone, RabbitMQ, NTP, Glance, Watchdog, Swift, Ceph	788.4 h	1 h
Docker	2990 h	1 h

7.1 Case Study I - Availability Evaluation

The first case study is the evaluation of proposed models' availability. The baseline architecture was the first one to be evaluated, 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 2 shows the availability results for each sub-model and for the cloud RBD model.

Table 2. Baseline Architecture Results

Model	Availability (%)	Downtime (h)
Controller	96.7285	286.58
Compute	99.4458	48.54
Storage	99.7936	18.08
Baseline	95.9987	350.51

The obtained results point out the availability of 96.72% for the controller's DRBD, which is the machine with more components, already the compute node had availability of 99.44%, while the storage reached 99.79%. The RBD composed by the three DRBDs representing the proposed environment had availability of 95.99%, meaning an annual downtime of 350.51 hours or more than eighteen entire days.

Based on the baseline architecture obtained values, we propose three other architectures. The Table 3 presents the availability results for each of these alternative architectures.

Table 3. Alternative Architectures Results

Model	Availability (%)	Downtime (h)
Double Redundant	99.8932	10.5
Triple Redundant	99.9964	0.31
Hyper-converged	99.9989	0.09
Baseline	95.9987	350.51

The architecture of greater availability is the hyper-converged one. It is worth noting that in this structure we have triple redundancy in Compute/OSD and controller nodes, and the obtained results pointed out an annual downtime of fewer than six minutes. Also, the double and full triple redundant models have lower availability than the hyper-converged one.

The architecture with double redundancy on all components has an annual downtime of 10.5 hours, which is a reasonable result compared to the obtained from the baseline architecture, while the full triple redundant model has almost 20 minutes of annual downtime.

7.2 Case Study II - Cost-Benefit Evaluation

A two stages survey characterizes the deployment cost analysis. At survey's first stage the necessary machines and storage devices to provide a cloud computing environment were defined.

To the survey's second stage, all the acquisition cost for each component individually was analyzed. By consulting Brazilian and North American websites[5], the lowest value of each architecture element in the period from 6 to June 13Th, 2018 was selected.

The server chosen to our environment is the Dell PowerEdge T320 servers, with Intel Xeon E5-2420, six physical cores, and 12 threads, 1 TB of storage and 24 GB RAM[6]. Already the storage mechanism is the Iomega StorCenter Pro ix4-200d 8 TB (4 X 2 TB) Network Attached Storage Server[7].

The third step is characterized by the analysis of the technical specification of each component in search of energy power values, aiming to calculate the energy cost per unit for one year, where these components would be functional 24 hours a day seven days a week. The cost of kilowatt time in Recife, Brazil, in March 2017 was 58 cents of real, converted to 16 cents of a dollar. In possession of these data, we calculated the kWh amount consumed by each device in one year by Equation 2, and multiply the result by the kWh value.

$$kWh = \frac{Power(W) \times No.h/Day \times No.DaysPerYear}{1000} \quad (2)$$

where Power stands for equipment required power, and No. h/Day for Number of hours per day that the equipment is used. Table 4 shows the relationship between components and their acquisition costs, power and annual energy cost.

[5] Amazon https://goo.gl/kC8yuj

[6] Dell Website https://goo.gl/XmTbwF

[7] Iomega Website https://goo.gl/f8xF1N

Table 4. Relationship between components, power and cost

Component	Acq. (USD)	Potency (W)	Energ. (USD)
Server	1209.82	100	140.16
Switch	119.00	6.9	9.67
Storage	699.00	63.07	
Air conditioning	490.83	1080	1513.73
Monitor	191.95	70	98.11
Keyboard	5.71	-	-
Mouse	2.55	-	-
Personal Computer	313.27	100	140.16
Rack	416.73	-	-

Table 5. Estimated Cost for each architecture

Architecture	Energ. (USD)	Acq. (USD)	Total (USD)
A1	2.105,06	4.658,58	6.763,74
A2	2.448,45	7.777,32	10.225,77
A3	2.791.84	10.895,96	13.687,8
A4	2.602,63	8.798,62	11.401,59

Table 5 presents the overall acquisition costs for each proposed architecture where the column that is the total value it is the sum of acquisition costs (Acq.) and electricity expenses (Energ.) for each one.

There are some fixed expenses for each architecture; this expenditure corresponds respectively to the switch, air conditioning, monitor, keyboard, mouse, rack and the personal computer needed to mount and access the infrastructures from the administrator side, this expense remains estimated in 1540.04 dollars. Another fixed expense is the energy annual consumption of these components; which is estimated in 1761.67 dollars.

To determine the architectures with the best cost-benefit relationship, we must define what is the benefit we are seeking is. For this work, we long for an ideal value between downtime and deployment and maintenance cost. However, they are different greatness, in order to compare them a normalization process put them into the same interval: 0 and 1. Equation 3 presents the normalization process conducted.

The architecture with the highest cost is the triple redundant converged one, which for its implementation and operation over a year it presents an expense of US$13.687,80 dollars, a value US$6.924,06 dollars higher than the architecture with the lowest cost, which is the baseline.

The double redundant architecture has a deployment cost US$1.175,82 dollars lower than the architecture with hyper-convergence, with an annual downtime 116 times higher than the obtained from the hyper-converged model, as seen in Figure 5.

To determine the architectures with the best cost-benefit relationship, we must define which the benefit we are seeking. For this work, we expect the ideal value between downtime and cost. However, they are different greatness, so we need to normalize and put them in the same interval: 0 and 1. Equation 3 presents the normalization process conducted.

$$NormalizeXY = \frac{NumX - MinNumX}{MaxNumX - MinNumX} \tag{3}$$

where X = Cost, Downtime, Y = Architecture: A1, A2, A3, A4, $MinNumX$ = Minimum value for architectures and $MaxNumX$ = Maximum value for architectures where A1 stands for Baseline, A2 for the double redundant environment, A3 is for the triple redundant scenario, while A4 corresponds to the hyper-converged architecture.

Now in possession of the normalized cost and downtime values, we must relate them somehow, so we use the Euclidean distance, the architecture with less distance from the origin tends to be the best one. The distances calculation may be done by the following equation: $DistanceZ = \sqrt{MNCost^2 + MNDowntime^2}$, where Z is for architecture and MN for the metric, either Normalized Cost or Normalized Downtime. The Ranking for the relationship between cost and downtime appears in Table 6.

Table 6. Architecture Ranking

Rank	Architecture	Euclidean Distance
1st	A2	0.50
2nd	A4	0.58
3rd	A1	0.71
3rd	A3	0.71

The ranking showed that the architecture A2, with double redundancy, has the best relationship between deployment cost and system's downtime. This architecture does not have the higher availability value or the worst price, but thanks to the normalization and distance methods we can define it as the best cost-benefit relationship. The last place goes for architectures A1 and A3, while one has the best cost and the worse downtime baseline; the other one has the inverse relationship triple redundant environment.

The feasibility of the results depends on the administrator needs, for example, a higher downtime implies in a lower deployment cost, as well as a higher deployment cost implies into a lower downtime, which is the case for the triple redundant environment in counterpart to the baseline architecture. Also, we can highlight the two other architectures, because they can be used as an availability midterm and offer a better cost-benefit relationship, these architectures are the hyper-converged and double redundant ones.

It is important to mention that the proposed models and obtained results are generalized through parameterization, which means that if your environment or datacenter has a higher or fewer amount of components the models can be adjusted to fit the company reality.

8. Conclusions and Future Works

This paper proposed and evaluated the availability of blockchain-as-a-service provisioning over four different architectures. Also, the cost-benefit relationship based on the deployment cost and annual downtime was established.

By modeling the proposed 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 artifacts to help cloud computing service providers and stakeholders.

It is important to highlight that the obtained results showed that the hyper-converged architecture has the higher availability and the second best cost-benefit relationship, while the baseline architecture has the higher downtime but the lowest deployment cost. As for future works, we intend to provide performability studies and evaluate the impact of reading workloads on the environment, as well as the application of a sensitivity analysis technique to determine how much does the components MTTF and MTTR impacts in the overall system's availability.

Author contributions

- Carlos Melo: main author.
- Jamilson Dantas: co-author, helped with the modeling formalisms.
- Paulo Pereira: co-author, helped to establish the cost-benefit relationship.
- Ronierison Maciel: co-author, helped with sensitivity analysis.
- Paulo Maciel: co-author, research supervisor, orchestration and proof reading.

References

[1] HAAG, M. *Hyper-Converged Infrastructures for DUMMIES*. [S.l.]: John Wiley & Sons, Inc., 2016.

[2] MATOS, R. et al. Redundant eucalyptus private clouds: Availability modeling and sensitivity analysis. *Journal of Grid Computing*, v. 15, n. 1, p. 1–22, Mar 2017. Disponível em: ⟨https://doi.org/10.1007/s10723-016-9381-z⟩.

[3] GUPTA, M. *Blockchain for DUMMIES*. [S.l.]: John Wiley & Sons, Inc., 2017.

[4] ZYSKIND, G.; NATHAN, O.; PENTLAND, A. . Decentralizing privacy: Using blockchain to protect personal data. In: *2015 IEEE Security and Privacy Workshops*. [S.l.: s.n.], 2015. p. 180–184.

[5] CHRISTIDIS, K.; DEVETSIKIOTIS, M. Blockchains and smart contracts for the internet of things. *IEEE Access*, v. 4, p. 2292–2303, 2016.

[6] KOSBA, A. et al. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: *2016 IEEE Symposium on Security and Privacy (SP)*. [S.l.: s.n.], 2016. p. 839–858.

[7] WEBER, I. et al. On availability for blockchain-based systems. In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. [S.l.: s.n.], 2017. p. 64–73.

[8] XIAO, J. et al. Blockchain architecture reliability-based measurement for circuit unit importance. *IEEE Access*, v. 6, p. 15326–15334, 2018.

[9] COSTA, I. et al. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Journal Quality and Reliability Engineering International*, 2015.

[10] AZAGURY, A. C. et al. Gpfs-based implementation of a hyperconverged system for software defined infrastructure. *IBM Journal of Research and Development*, v. 58, n. 2/3, p. 6:1–6:12, March 2014.

[11] U, A. et al. The efficient use of storage resources in san for storage tiering and caching. In: *2016 2nd International Conference on Computational Intelligence and Networks (CINE)*. [S.l.: s.n.], 2016. p. 118–122.

[12] TANIGUCHI, Y. et al. Tandem equipment arranged architecture with exhaust heat reuse system for software-defined data center infrastructure. *IEEE Transactions on Cloud Computing*, v. 5, n. 2, p. 182–192, April 2017.

[13] ZHANG, Y. et al. 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)*. [S.l.: s.n.], 2016. p. 931–935.

[14] MACIEL, P. et al. Dependability modeling. In: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. [S.l.: s.n.], 2011.

[15] AVIŽIENIS, A. et al. *Fundamental Concepts of Dependability*. University of Newcastle upon Tyne, Computing Science, 2001. (Technical report series). Disponível em: ⟨https://books.google.com.br/books?id=cDkmGwAACAAJ⟩.

[16] MALHOTRA, M.; TRIVEDI, K. Power-hierarchy of dependability-model types. *Reliability, IEEE Transactions on*, v. 43, n. 3, p. 493–502, Sep 1994.

[17] SOFTWARE, I. *Reliability Block Diagram*. 2007. Http://www.reliabilityeducation.com/rbd.pdf. [Online; accessed 26-September-2015].

[18] GARG, S. et al. Analysis of software rejuvenation using markov regenerative stochastic petri net. In: *Proc. In: Sixth International Symposium on Software Reliability Engineering, (ISSRE'95)*. Paderborn: [s.n.], 1995. p. 180–187.

[19] DISTEFANO, S.; XING, L. A new approach to modeling the system reliability: dynamic reliability block diagrams. In: *RAMS '06. Annual Reliability and Maintainability Symposium, 2006.* [S.l.: s.n.], 2006. p. 189–195.

[20] XU, H.; XING, L.; ROBIDOUX, R. Drbd: Dynamic reliability block diagrams for system reliability modelling. *International Journal of Computers and Applications*, v. 31, n. 2, p. 132–141, 2009. Disponível em: ⟨http://www.tandfonline.com/doi/abs/10.1080/1206212X.2009.11441934⟩.

[21] DANTAS, J. et al. Models for dependability analysis of cloud computing architectures for eucalyptus platform. *Inter-*

national Transactions on Systems Science and Applications, v. 8, p. 13–25, 2012.

[22] CAMPOS, E. et al. Stochastic modeling of auto scaling mechanism in private clouds for supporting performance tuning. In: *Proceedings of the IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'15)*. Hong Kong, China: [s.n.], 2015.