Federal Rural University of Pernambuco
Department of Statistics and Informatics
Post Graduation in Applied Informatics

Elton Bezerra Torres

AVAILABILITY AND PERFORMANCE ANALYSIS OF PRIVATE CLOUD STORAGE SERVICES

M.Sc. Dissertation

RECIFE
2017

Elton Bezerra Torres

AVAILABILITY AND PERFORMANCE ANALYSIS OF PRIVATE CLOUD STORAGE SERVICES

A M.Sc. Dissertation presented to the Department of Statistics and Informatics of Federal Rural University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Applied Informatics.

Advisor: *Prof. Dr. Gustavo Rau de Almeida Callou*

RECIFE
2017

Elton Bezerra Torres
Availability and Performance Analysis of Private Cloud Storage Services/ Elton Bezerra Torres. – RECIFE, 2017-
76 p. : il. (algumas color.) ; 30 cm.

Advisor Prof. Dr. Gustavo Rau de Almeida Callou

M.Sc. Dissertation – Universidade Federal Rural de Pernambuco, 2017.

1. Palavra-chave1. 2. Palavra-chave2. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

CDU 02:141:005.7

Dissertação de mestrado apresentada por **Elton Bezerra Torres** ao programa de Pós-Graduação em Informática Aplicada do Departamento de Estatística e Informática da Universidade Federal Rural de Pernambuco, sob o título **Availability and Performance Analysis of Private Cloud Storage Services**, orientada pelo **Prof. Prof. Dr. Gustavo Rau de Almeida Callou** e aprovada pela banca examinadora formada pelos professores:

Prof. Gustavo Rau de Almeida Callou
Departamento de Estatística e Informática/UFRPE

Prof. Ermeson Carneiro de Andrade
Departamento de Estatística e Informática/UFRPE

Prof. Gabriel Alves de Albuquerque Júnior
Departamento de Estatística e Informática/UFRPE

Prof. Eduardo Antonio Guimaraes Tavares
Centro de Informática/UFPE

RECIFE
2017

I dedicate this dissertation to my mother and father.

Acknowledgements

After many difficulties, it is time to finish this step in my academic and professional career. I would not have been able to do this work without the help, companionship and love of many people.

First of all, I want to thank my parents, Nilda and Antonio, without whom I would not have gotten here. I owe so much thanks to the effort of both who believed in me. I hope to return all their benevolence in due course. To my wife Clyvia for the unfailing companionship and patience, who helped me at various times during this journey. I owe her and my daughter Alice apology for the moments I was absent during this masters course.

To my colleagues José Accioly and Hallyson Gustavo and the Professors Ermeson Andrade and Gabriel Alves for the valuable tips for the development of this work and all the other colleagues of the PPGIA.

To my advisor Gustavo Callou for his patience, availability and support.

There is a season for everything, and a time for every event under heaven: A time to be born, and a time to die; a time to plant, and a time to uproot what was planted.

—ECCLESIASTES 3:1

Resumo

A computação em nuvem é um conjunto de novas práticas, tecnologias e conceitos que favorecem serviços de comunicação e armazenamento de dados. Serviços como OneDrive, Google Drive e DropBox aumentam a disponibilidade dos dados e provêem novas facilidades como sincronização e colaboração. Estes serviços requerem alta disponibilidade e características de desempenho, como alta vazão, que são fundamentais para garantir a continuidade dos negócios e a não interrupção dos serviços providos. Este trabalho tem como objetivo avaliar métricas de disponibilidade e desempenho para serviços de armazenamento em nuvem privada. Uma estratégia hierárquica baseada em modelos é proposta para avaliar tanto disponibilidade como o desempenho através da composição de Cadeias de Markov de Tempo Contínuo (CTMC), diagramas de blocos de confiabilidade (RBD) e redes de Petri estocásticas (SPN). Estudos de casos serão apresentados para ilustrar a aplicabilidade dos modelos propostos através de um serviço de armazenamento de arquivos em nuvem hospedado na plataforma Eucalyptus. Também é adotado o índice de importância da disponibilidade para identificar os componentes mais críticos com relação à disponibilidade do sistema. Levando em consideração esse índice, são propostos cenários considerando diferentes níveis de redundância e, consequentemente, o efeito sobre a disponibilidade e desempenho global do sistema é explorado. Os resultados demostram que a estratégia de modelagem proposta pode ser atraente para identificar quais componentes devem contar com redundância para aumentar o desempenho e a disponibilidade do sistema.

Palavras-chave: Computação em Nuvem, CTMC, Desempenho, Disponibilidade, Eucalyptus, RBD, SPN.

Abstract

Cloud computing is a set of new practices, technologies and concepts that favor communication services and data storage. Services like OneDrive, Google Drive and DropBox increase data availability and provide new features as synchronization and collaboration. These services require high availability and performance characteristics like high throughput, since it is fundamental to guarantee both business continuity and uninterrupted public services. In this research, we aim at evaluating availability and performance-related metrics for private cloud storage services. A hierarchical model-based strategy is proposed to evaluate availability throughput and system utilization by means of the composition of continuous-time Markov chains (CTMC), reliability block diagrams (RBD) and stochastic Petri nets (SPN). Case studies are presented to illustrate the applicability of the proposed models through a cloud storage service hosted in the Eucalyptus platform. We also adopt availability importance index to identify the most critical components in relation to the system availability. Based on such an index, we propose scenarios considering different degrees of redundancy and, accordingly, the effect on overall system availability and performance is explored. The results demonstrate that the proposed strategy can be an attractive approach for identifying which components should be supported with redundancy in order to increase the overall system performance and availability.

Keywords: Availability, Cloud Storage, CTMC, Performance, RBD, SPN.

List of Figures

2.1 Computational paradigms evolution. 24
2.2 Cloud computing deployment models. 26
2.3 Cloud computing service models. 27
2.4 Example of Eucalyptus components. 28
2.5 Performance Evaluation Techniques. 31
2.6 System classifications. 33
2.7 Petri net basic elements. 34
2.8 PN Transition Enabling and Firing. 35
2.9 Example of a SPN Model. 37
2.10 Reliability Block Diagrams. 38
2.11 Example of a RBD k-out-of-n configuration. 39
2.12 Example of an RBD bridge configuration. 40
2.13 Example of a Continuous-time Markov Chains (CTMC) model. 41
2.14 Availability Importance numerical example. 44

3.1 Modeling strategy for availability and performance evaluation. 47
3.2 Adopted architecture. 50

4.1 Model of the Reliability Block Diagrams (RBD) Node Module. 52
4.2 CTMC of the VM Module. 52
4.3 Model of the RBD Storage Module. 54
4.4 Model of the RBD Controller Module. 54
4.5 RBD of the entire system. 54
4.6 SPN model for the cloud storage service. 56

5.1 Throughput graphical comparison 60
5.2 RBD model for the Scenario 2. 63
5.3 RBD model for the Scenario 3. 63
5.4 RBD model for the Scenario 4. 64
5.5 RBD model for the Scenario 5. 65
5.6 Downtime results comparison. 65
5.7 Number of 9s for all scenarios. 66
5.8 POT and NUF results comparison. 68

List of Tables

1.1 Comparison between the proposed work and related works. 21

2.1 MTTF and MTTR of all the components. 44
2.2 Availability importance measures for all the components. 44

4.1 Nomenclature of CTMC states. 53
4.2 Expression for the calculation of SPN measurements. 55
4.3 Transitions attributes. 57

5.1 Parameters used for the experiments. 59
5.2 Results obtained from the measurements and models. 60
5.3 Parameters of the RBD models. 61
5.4 CTMC parameters for the Virtual Machine (VM) component. 61
5.5 Parameters of the RBD representing the entire system. 62
5.6 Summary results for all scenarios. 62
5.7 Availability Importance values of baseline scenario. 62
5.8 Availability Importance values of second scenario. 63
5.9 Availability Importance value of the third scenario. 64
5.10 Availability Importance value of the fourth scenario. 64
5.11 Summary of performance evaluation results. 67

List of Acronyms

CTMC	Continuous-time Markov Chains 16
DTMC	Discrete-time Markov Chains 41
NIST	National Institute of Standards and Technology 22
EBS	Amazon Elastic Block Storage 17
iSCSI	Internet Small Computer System Interface 30
AMI	Amazon Machine Image 29
RBD	Reliability Block Diagrams 16
SPN	Stochastics Petri net 16
PN	Petri net 33
IaaS	Infrastructure as a Service 17
PaaS	Platform as a Service 26
SaaS	Software as a Service 26
PC	Personal Computer 23
CLC	Cloud Controller 28
OSP	Object Storage Provider 28
CC	Cluster Controller 28
SC	Storage Controller 28
NC	Node Controller 28
AI	Availability Importance 19
MTTF	Mean Time To Failure 19
MTTR	Mean Time To Repair 42
OS	Operating System 30
HW	Hardware 51
KVM	Kernel-based Virtual Machine 49
VM	Virtual Machine 20
NFS	Network File System 30
DB	Data Base 49
UEC	Ubuntu Enterprise Cloud 19

PBF Probability of the network buffer to be full 55
POT Probability of timeout occur 55
NUF Probability the users are not attended due to failures 55
SU System utilization 55
ST System throughput 55
CRM Customer Relationship Management 17
EC2 Elastic Compute Cloud 17
EBS Elastic Block Store 17
GSPN Generalized Stochastic Petri Nets 19
IT Information Technology 22
PC Personal computer 23
CPU Central processing unit 23
ERP Enterprise Resource Planning 26
S3 Simple Storage Service 29
SLA Service Level Agreement 29
AWS Amazon Web Services 48
HW Hardware 51

Contents

1 Introduction **15**

1.1 Motivation and Justification 17
1.2 Objectives 18
1.3 Related works 18
1.4 Structure of the Dissertation 21

2 Background **22**

2.1 Cloud Computing 22
2.1.1 Essential Characteristics 23
2.1.2 Deployment Models 25
2.1.3 Service Models 26
2.2 Eucalyptus Platform 27
2.2.1 Cloud Controller 28
2.2.2 Object Storage Provider 29
2.2.3 Cluster Controller 29
2.2.4 Storage Controller 30
2.2.5 Node Controller 30
2.3 Performance and Dependability evaluation 30
2.3.1 System Classifications 32
2.4 Petri Nets 33
2.4.1 Stochastic Petri Nets 35
2.5 Reliability Block Diagrams 38
2.6 Markov Chains 40
2.7 Availability Importance 42
2.8 Summary 45

3 Methodology and Architecture **46**

3.1 Methodology 46
3.2 Private Storage Cloud Environment 48
3.3 Summary 50

4 Models **51**

4.1 Availability Models 51
4.1.1 Node 51
4.1.2 Virtual Machine 52
4.1.3 Storage 53

4.1.4 Controller 54
4.1.5 Composition of models 54
4.2 Performance Model 55
4.3 Summary 57

5 Case Study **58**
5.1 Case Study I - Validation 58
5.2 Case Study II - System Availability 60
5.3 Case Study III - Performance Evaluation 66
5.4 Summary 69

6 Conclusion and Future Works **70**
6.1 Contributions 71
6.2 Future works 72

References **73**

1 Introduction

For a long time, data has been tied to a single machine which was generally a desktop computer or a mobile device and belonged to a single user. However, what would happen when the machine which stored those data malfunctions and the user did not have a backup? How does one keep these data safe? How does one make these data available whenever they need to be used? How do we share these data among many devices that have been developed and popularized in the last decade, such as smartphones, tablets, smart TVs, among others? How to manage the synchronization of these data, and maintain data updated across all the devices? In order to address the issues raised above, a new cloud computing paradigm called cloud storage services have become incredibly popular and widely adopted over the years.

Cloud computing provides on-demand access to shared computing resources and services, such as network infrastructure, storage, operating systems, and applications. Such resources and mechanisms can be easily acquired and released with minimal management effort (MELL; GRANCE, 2011). These features enable administrators to focus only on the business model, without worrying about infrastructure details. The experience in acquiring public cloud services is often compared to the consumption of public utilities, such as electricity or water so that user just pay for the service without worrying or knowing about the infrastructure that provides it (BAUER; ADAMS; EUSTACE, 2011).

Cloud storage service is a cloud computing service in which data can be stored, edited and retrieved from a remote cloud storage server over the Internet under a utility computing model. This service is built to provide for applications, services and organizations access to offsite storage capacity that can be provisioned instantly, with automatic scaling at run time and globally accessible. Cloud storage service is provided, hosted and managed by cloud storage service providers, such as Microsoft One Drive, Google Drive, DropBox, Amazon Cloud Drive, alongside a range of other suppliers of this nature (CASSERLY, 2016).

This type of storage service has had an exponential growth over recent years. In July 2012, Microsoft alone had stored 4 trillion objects; and in January 2015 this value already exceeded 10 trillion objects (CALDER, 2015). Cloud storage has become a critical component of cloud computing, storing the information used by applications and users. Big data analytics,

data warehouses, Internet of Things, databases, and backup and archive applications all rely on some form of data storage architecture. Cloud storage is typically more reliable, scalable, and secure than traditional on-premises storage systems (AMAZON, 2016a). There are many advantages in using cloud storage services, such as: the cost and easiness to perform backups and data recovery; protection of data hosted in large data center infrastructures; accessibility and synchronization of these data among any device connected to the network. In addition, data sharing and collaborating among users can be improved by the adoption of this paradigm (NIELSEN, 2015). Some disadvantages can also be enumerated when utilizing this kind of service, such as: the need to be connected to the Internet, security, privacy, legislation, service downtime, limited control and flexibility and the ownership of personal or business data hosted at the service suppliers (CLOUDSTORAGE, 2015).

However, even large cloud storage service providers are not failure-free; recent outages at (FIVEASH, 2015) and (BELL, 2015) show that regardless of companies' size or their target market, they are all prone to catastrophic events. Customers also have specific performance (e.g., response time, throughput) and availability requirements when using cloud storage services (XIONG; PERROS, 2009). Therefore, the efficient and accurate assessment of the adopted cloud storage services is essential to find the best one that best suits customer needs and demands.

In addition to public clouds, under which cloud services are provided and accessible over a public network such as the internet, there are also private and hybrid clouds. The private cloud infrastructure are operated for a single organization, whether managed internally or by a third party, and hosted either internally or externally. Private clouds provide a major level of control on software, network, and hardware components and more control of privacy and security. This factors might lead to the choice of private clouds instead of public clouds. The hybrid clouds, on the other hand, combines two or more distinct cloud infrastructures.

The main goal of this dissertation is to propose an approach to evaluate availability and performance-related metrics of a data storage service hosted in a private cloud by means of analytic models. A hierarchical strategy is adopted to evaluate both availability and performance metrics through the composition of Reliability Block Diagrams (RBD), Continuous-time Markov Chains (CTMC) and Stochastics Petri nets (SPNs). RBD and CTMC models are used to evaluate the service availability of the private cloud where multiple scenarios are also considered using multiple levels of redundancy. The SPN model, on the other hand, is adopted to model the service operation and to obtain performance metrics from the system assuming different amounts of users and download of files with different sizes. Each of theses formalism has its own purpose. For example, RBD cannot be used when there is dependence between devices (e.g. failure of an equipment activates a backup device). For these cases, SPNs or CTMCs may be used, but these formalisms deal with the problem of the state space explosion (BOLCH et al., 2006). Therefore, an approach that combines the use of these formalisms is essential.

Case studies are proposed to illustrate the applicability of the proposed strategy, taking into account crucial performance and availability metrics like system availability, downtime,

system utilization and throughput. The results show how the different levels of availability affect the system performance metrics, which may be adopted by individuals who are interested in building and selecting their own cloud storage environments. The major contribution of this dissertation are the following: (i) a hierarchical model-based approach to assess distinct data storage metrics like throughput, utilization, and availability; (ii) a comprehensive stochastic model is composed for the purpose of availability and performance analysis of complex cloud storage environments; (iii) an approach that allows individuals to plan and build efficient and cost-effective cloud storage environment.

1.1 Motivation and Justification

The number of users who store data in cloud systems are growing, and they depends on the performance and availability of these services. Many companies have to migrate data between cloud environments, thus a downtime of one minute from a provider may significantly impact profits and damage the relation with the customers. Despite the advances in technology, outages in cloud computing are still common. In March 3, 2016, Salesforce customers in Europe had to cope with a Customer Relationship Management (CRM) disruption for over 10 hours; in April 11, an outage took down Google Cloud Platform services for 18 minutes; Apple's cloud experienced a widespread outage in June 2; in June 4, the Amazon Web Services region in Australia lost the power, and a number of Elastic Compute Cloud (EC2) instances and Elastic Block Store (EBS) volumes hosting critical workloads for name-brand companies failed (TSIDULKO, 2016).

A storage service in the cloud that has a high failure rate, offers a negative user experience, resulting in losses to the service provider. The user has several ways to criticize a bad service on the Internet, and even has the possibility to use alternative services in a matter of seconds. Therefore, services are vulnerable to strong user rejection if they do not offer services with quality and high availability levels.

Some software solutions adopt virtualization to provide IaaS services in the cloud, as well as to improve data and service availability, security, resource utilization, cost and other benefits (KIM; MACHIDA; TRIVEDI, 2009). Eucalyptus platform is an example of a cloud platfomr. It has been developed to build private and hybrid clouds in the Infrastructure as a Service (IaaS) model, controlling large resource collections such as network and storage (EUCALYPTUS, 2015a). Even with the support of this platform, hardware and software failures, updates, and planned maintenance may affect the performance and / or availability of the service. In this way, maintaining adequate levels of performance and availability are goals to be achieved.

Considering the challenges presented before, this dissertation aims to propose a modeling approach for availability and performance analysis of a data storage service hosted in a private cloud. This approach employ hierarchical modeling techniques and combine distinct models (e.g. RBD, CTMC, SPN) and availability index to provide information about the performance, availability and components redundancy to help designers and administrators of IaaS cloud to

choose the most appropriate cloud storage environment.

1.2 Objectives

Modeling techniques can provide support to cloud service providers for improvements in the service delivered as well as on the planning of the infrastructure needed without implementing the real system. In this context, the main objective of this work is to propose a modeling approach for integrated evaluation of availability and performance metrics in a data storage service hosted in a private cloud. This approach combines different computational models in a hierarchical strategy to evaluate performance and availability metrics. Furthermore, the specific goals of this research are:

- To implement a private cloud infrastructure with a data storage service in our lab to conduct experiments;
- To define and implement a benchmarking tool that will provide support on the workload for the data storage system;
- To elaborate models for performance and availability evaluation of the private cloud storage service implemented;
- To validate of the proposed models through experiments;
- To apply of techniques to identify the components with the highest availability impact on the system availability;
- To propose new architectures based on the availability and performance metrics previously computed.

1.3 Related works

Over the last years, some works have been developed to evaluate availability and performance of cloud infrastructures. However, none of them focused on private cloud storage. The related works were selected through keywords such as cloud data storage, modeling and evaluation of dependability metrics in cloud computing environments, in which the first string containing cloud storage and the second one can contain stochastic model, modeling availability or performance. The search period occurred between the months of May 2015 and September 2016 and was limited to the main digital libraries in the area of computer science: ACM, ScienceDirect, Springer and IEEE Xplore. As a time limit, the interval defined was between the years 2010 and 2016. The following lines present related works of this research.

(MATOS et al., 2012) proposed a method based on parametric sensitivity analysis of CTMCs for evaluating the availability of data networks. The authors also investigated redundancy mechanisms, using several scenarios evaluated through an analytic-numeric solution of Markov chains. They found that host failure is the most important parameter when the measure of interest is the Mean Time To Failure (MTTF) of the virtual machine system. Considering capacity oriented availability, the authors have shown that the failure rate of applications is of major concern. (DANTAS et al., 2012) investigated the benefits of a warm standby redundancy mechanism in a Eucalyptus cloud computing environment. A hierarchical heterogeneous modeling approach is used to represent redundant architectures, in which the availability of each different redundant approach was compared in relation to the baseline system (without redundancy). Both hardware and software failures are considered in the proposed analytical models, which are used to obtain closed-form equations for computing the availability of the cloud infrastructure. Although both papers evaluate availability and component importance of data networks and cloud infrastructures, performance is not considered.

(WEI; LIN; KONG, 2011) proposed a hierarchical method that combines RBD and Generalized Stochastic Petri Nets (GSPN) models to analyze the availability of virtual data centers and the impacts brought by virtualization mechanisms including consolidated backup and live migration. The authors showed the impact of these mechanisms on dependability, and conclude that heterogeneous and hierarchical modeling approaches can express complex characteristics virtual data centers by combining the advantages of two or more types of models. (JANPET; WEN, 2013) investigated data availability in cloud storage service. The study presented a data backup method for data retrieval and access from distinct places based on the access frequencies of a shared folder. The authors investigate the relationship between reliability and availability issues for cloud storage assignment based on the access frequencies of several user locations. Their approach achieved a high reliability through data replication, which selects the closest backup cloud storage to the user's location, ensuring data availability and data sharing worldwide. Besides adopting heterogeneous and hierarchical models to compute system availability, our work proposes models for performance evaluation and adopts Availability Importance (AI) to improve the system availability.

(HAUNG; MA; SUN, 2012) proposed a method that places a replica in a cloud storage according to user requirement for service. In that work, a performance model is proposed for representing the node storage replica using an analytic hierarchy process. The results showed that the proposed method could provide more reasonable replica strategies effectively. Despite the fact that the performance of a cloud storage system was conducted, the system availability was not the focus of that work. (CHUOB; POKHAREL; PARK, 2011) proposed a private cloud solution selected from suitable cloud environments for an e-government data center, which is based on the Ubuntu Enterprise Cloud (UEC) architecture. The availability of each component of the cloud environment was represented by a Markov model, and experiments were conducted for observing the behavior of the cloud in order to be able to obtain accurate parameters (e.g.,

failure and recovery rates) for the proposed models. Different from this work, we adopted an hierarchical approach that considers the advantage of CTMC, SPN and RBD models for computing system's availability and performance.

(GHOSH et al., 2014) the authors performed availability and performance analysis of IaaS cloud infrastructures. A stochastic modeling approach is used for performability analysis of these infrastructure using Markov chain. The authors revealed that for very large systems, simulation results are obtained faster than the numerical solution (interacting sub-models), and cloud service providers can benefit from modeling approaches during design, development, testing and operation of an IaaS cloud. Even though the proposed approach is very interesting, it does not take into account any method or index (e.g., AI) to indicate which component should be replicated to improve system's availability. In (SOUSA et al., 2012), the performance of distinct Virtual Machine (VM) types in the Eucalyptus platform was evaluated using well-known benchmarks. Their analysis enabled to assess the quality of the services and prevent performance degradation related to fluctuations in workload in private clouds. Although our works are similar, this dissertation focuses on cloud computing storage system.

In (SHENG et al., 2013), an open-source solution for storage in a private cloud was utilized to store and manage scientific data of a university. The authors demonstrated that the use of this solution makes the management of cloud data fast, efficient and easy, providing the benefits of synchronization and collaboration among users. The platform, which is based on Seafile, can achieve the functions of group management, file synchronization, and online collaboration. This solution facilitated and made more convenient the interaction between professors and students. Similar results were found in (HILDMANN; KAO, 2014) considering a high school as the scenario of study, where the solution adopted was based on the open-source project called ownCloud. The authors concluded that the need for an on-premise cloud storage solution is recognized by many organizations, especially universities. Although the two papers analyze the use of open-source solutions for private cloud storage, they do not quantify performance and availability metrics.

Additionally, the authors (WANG; GONG; XIE, 2012) described and evaluated key metrics to be consider in a cloud storage service in order to increase the redundancy level of the service. The authors quantified several main metrics for evaluating relative merits of different redundancy methods in cloud systems and define an evaluation model to provide references for cloud services providers.

Table 1.1 shows the relationship between the proposed work in this dissertation and main related works presented before. Note that most of these works discussed does not focus in cloud storage and has attempted to solve one side of the problem: either availability or performance aspects. However, there are various kinds of cloud storages with conflicting requirements that need approaches to judge which one should be chosen. Note that the majority of the mentioned works considered computational models to evaluate performance or availability without considering the possibility to adopt hierarchical strategies. A hierarchical heterogeneous

approach enables representing details of specific processes (subsystems). Thus, most of the related works do not consider such detailed behavior. In addition, works that are specifically focused on cloud storage services, do not use computational models to evaluate performance and availability, neither do they evaluate the importance of system components in the availability.

Table 1.1: Comparison between the proposed work and related works.

	Cloud type	Uses models	Heterogeneous hierarchical modeling	Evaluates availability	Evaluates performance	Evaluates component importance
This dissertation	Private	RBD, CTMC, SPN	✓	✓	✓	✓
(MATOS et al., 2012)	-	CTMC	-	✓	-	✓
(DANTAS et al., 2012)	Private	RBD, CTMC	-	✓	-	✓
(WEI; LIN; KONG, 2011)	-	RBD, GSPN	✓	✓	-	-
(JANPET; WEN, 2013)	-	-	-	✓	-	-
(HAUNG; MA; SUN, 2012)	-	-	-	-	✓	-
(CHUOB; POKHAREL; PARK, 2011)	Private	CTMC	-	✓	-	-
(GHOSH et al., 2014)	Private	CTMC, SPN	-	✓	✓	-
(SOUSA et al., 2012)	Private	-	-	✓	✓	-
(SHENG et al., 2013)	Private	-	-	-	✓	-
(HILDMANN; KAO, 2014)	Private	-	-	-	✓	-
(WANG; GONG; XIE, 2012)	Public	-	-	✓	✓	-

1.4 Structure of the Dissertation

The remaining parts of this dissertation are organized as follows. Chapter 2 presents theoretical basis for the work, introducing the fundamental concepts on cloud computing, Eucalyptus private cloud platform, stochastic Petri nets, reliability block diagrams, continuous-time Markov chains and availability importance. Chapter 3 describes the cloud computing architecture adopted for the case study analysis and the modeling strategy proposed for availability and performance analysis. Chapter 4 presents and details the proposed models to represent the architecture. Chapter 5 illustrates the applicability of the proposed models through case studies. Finally, Chapter 6 shows the final conclusions on this work and provides directions for future extensions.

2

Background

This chapter presents an overview of relevant concepts for a better understanding of this work. First, concepts related to cloud computing and Eucalyptus platform are presented. Afterwards, a brief introduction to performance and dependability evaluation models (e.g., SPN, CTMC, RBD) are shown.

2.1 Cloud Computing

Cloud computing can be understood as on-demand delivery of computer power, database storage, applications, and other Information Technology (IT) resources through a cloud services platform via the Internet with a pay-as-you-go pricing amazondef. The U.S. National Institute of Standards and Technology (NIST) defines cloud computing as follows:

> Cloud computing is a model for enabling ubiquitous, convenient, on- demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (MELL; GRANCE, 2011).

Figure 2.1 shows the evolution of computational paradigms, from the mainframe to the cloud computing. For instance, Figure 2.1 (a) depicts a mainframe example, which centralizes the computing platform. In addition to the mainframe, this paradigm is composed of terminals that access the computing power, storage, and memory from the mainframe. Figure 2.1 (b) presents the personal computer that is powerful enough to meet daily user needs, so there is no need to share a mainframe with other users. The scenario shown in Figure 2.1 (c) introduces client-server model, a group of computer systems and other computing hardware devices that are linked together through communication channels to facilitate communication and resource-sharing among a wide range of users in a local network. Local networks are connected to other local networks forming a global network (such as the Internet) to use remote applications and resources on the scenario depicted in Figure 2.1 (d). In the following example, grid computing interconnects computer systems providing shared computing power and storage through a distributed computing system in a transparent manner. Finally, Figure 2.1 (f) presents the cloud

computing delivering on-demand computing resources of hardware and software available on the Internet in a scalable and simple way on a pay-for-use basis (FURHT, 2010).

Cloud computing moves computing and data away from local devices into large data centers, and the users use a variety of devices, including Personal computer (PC), laptops, smartphones, and tablets to access programs, storage, application, and development platforms over the Internet, via services provided by cloud computing providers (FURHT, 2010). In cloud computing, resources (hardware or software) are not visible to the user, i.e., they are offered in a transparent manner as services, so there is no need for powerful PCs because all services are processed on servers on the Internet. It looks like cloud computing is a return to the original mainframe computing paradigm. However, these two paradigms have several important differences: a mainframe is a physical machine with finite computing power, and a cloud provides highly scalable set of resources, suggesting virtually infinite power and capacity. Note that unlike a simple terminal acting as a user interface to a mainframe, a PC in the cloud computing paradigm possesses enough power to provide a certain degree of local computing (FURHT, 2010).

(ARMBRUST et al., 2010) consider that three aspects are new in cloud computing, from a hardware point of view: (i) the illusion of infinite computing resources available on demand, thereby eliminating the need for cloud computing users to plan far ahead for provisioning; (ii) the elimination of an up-front commitment by cloud users, thereby allowing companies to start small and increase hardware resources only when their needs grow; (iii) the ability to pay to use computing resources on a short-term basis (e.g., processors by the hour and storage by the day) and release them when they are no longer useful (called as "utility computing"). Therefore, computational services are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony.

The cloud model comprises five essential characteristics, four deployment models, and three service models (MELL; GRANCE, 2011), which are described as follows.

2.1.1 Essential Characteristics

The NIST has defined five essential traits for cloud computing (MELL; GRANCE, 2011):

- **On-demand self-service:** Users are able to provision cloud computing resources, such as Central processing unit (CPU) time, network and storage, without requiring human interaction, mostly done through a web-based self-service portal (management console). Typically, users are billed for a monthly subscription or a pay-for-what-they-use scenario (MELL; GRANCE, 2011).

- **Broad network access:** Cloud computing resources (e.g., storage, processing, memory, and network) are accessible over the network, supporting heterogeneous client platforms such as mobile devices and workstations, in a transparent manner through standardized mechanisms (MELL; GRANCE, 2011).

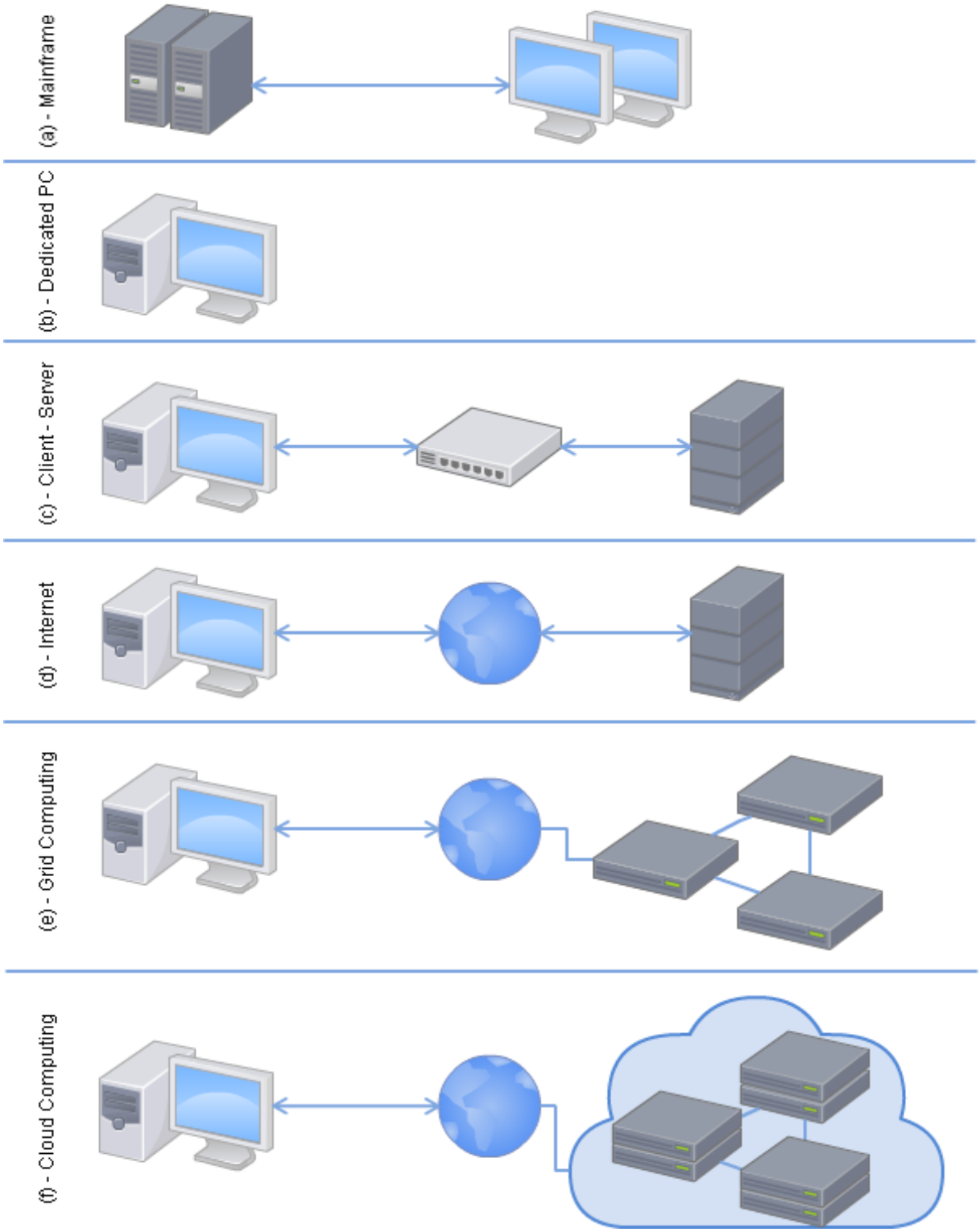


Figure 2.1: Computational paradigms evolution: from mainframe to cloud.

- **Resource pooling:** The resources of several physical servers are combined and securely separating on a logical level, to serve multiple customers, dynamically distributed according to the current demand of the customers. There is a sense of location independence of these resources, i.e., customers do not have control or knowledge over the exact location of the resources. But, at a higher level of abstraction, through a customer's control panel or directly with the provider, these options can be configured (MELL; GRANCE, 2011).

- **Rapid elasticity:** Resources are provisioned and released on-demand and/or automated based on triggers or parameters. To customers, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time (SCHOUTEN, 2014).

- **Measured service:** The amount of resources used can be monitored, controlled, measured, and reported (billed) from both sides, customers and cloud computing service provider, which provides transparency (SCHOUTEN, 2014).

2.1.2 Deployment Models

Assuming deployment models, clouds can also be classified by who own and manage the cloud (MELL; GRANCE, 2011). Figure 2.2 depicts the three main deployment models: public, private, and hybrid clouds.

- Public cloud: This cloud corresponds to the infrastructure that is owned and operated by companies offering access over a public network to computer resources. With public cloud services, users don't need to purchase hardware, software, or supporting infrastructure, which is owned and managed by providers. However, this model suffers security requirements issues, which are the subject of several studies and debates (SCHOUTEN, 2014). Public clouds offer several key benefits to service providers, including no initial capital investment on infrastructure and shifting risks to the infrastructure providers (ZHANG; CHENG; BOUTABA, 2010).

- Private cloud: This model is usually adopted for a single organization, whether managed internally or by a third party and hosted either internally or externally. Private clouds can take advantage of cloud's efficiencies while providing more control of resources (SCHOUTEN, 2014). Beyond privacy and security concerns, a need for fine-grained level of control on software, network, and hardware components might lead to the choice of private clouds instead of public clouds.

- Hybrid cloud: This type of cloud combines two or more distinct cloud infrastructures (private or public) that remain unique entities, but are accessed jointly by means of standardized or proprietary technology that enables data and application portability

(MELL; GRANCE, 2011). Hybrid clouds might be arranged to keep confidential or other sensible data exclusively on local premises whereas common information is stored and processed on public infrastructures. Load balancing between clouds is also an application for hybrid clouds.

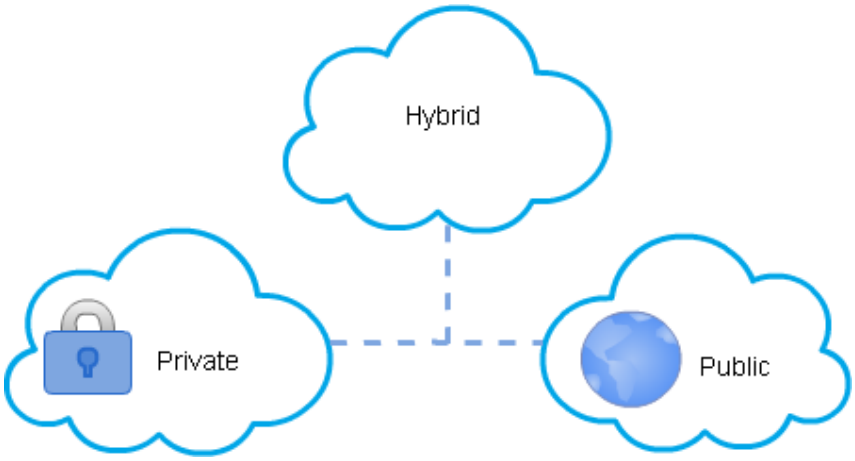


Figure 2.2: Cloud computing deployment models.

2.1.3 Service Models

The services offered by cloud computing can be presented as a layered architecture (SCHOUTEN, 2014), as shown in Figure 2.3. These services are also known as business models, and the NIST defines three service models for cloud computing: infrastructure as a service, platform as a service, and software as a service.

- Software as a Service (SaaS): SaaS provides a completed product that is run and managed by a service provider. In most cases, SaaS is referred to as end-user applications. A common example of a SaaS application are email, file servers, content management, where you can use these programs without having to manage product or maintaining the servers and operating systems that the program is running on. The customer does not manage or configure the cloud infrastructure, but it is possible for limited settings of specific applications (AMAZON, 2016b). SaaS has become a major trend in the development of applications. There are important examples of cloud-based software in almost any category, including office suites, Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), photo edition tools, and media players. Google Docs, DropBox, Adobe Creative Cloud, Salesforce CRM, Gmail for Work, and Youtube which correspond to services that have been replacing traditional client-server or stand-alone desktop applications. Such services are usually built on top of Platform as a Service (PaaS), or directly on IaaS environments without the aid of PaaS features.

- Platform as a Service (PaaS): PaaS provides a cloud-based environment with everything required to develop and support web-based (cloud) applications. PaaS removes from the consumer the control of underlying cloud infrastructure including the network, servers, operating systems, or storage. PaaS also provides support for deployed applications and configuration settings of the application-hosting environment (SCHOUTEN, 2014). An example is the Google App Engine, that allows developers to create and deploy applications with Python, Java, PHP, and Go programming languages directly on the Google infrastructure. In this way, users are benefited through ready-to-use libraries and frameworks, as well as the support to scale once the traffic and data storage needs improvements (GOOGLE, 2016).

- Infrastructure as a Service (IaaS): IaaS typically provides access to networking features, computers (virtual or dedicated hardware), and data storages. The consumer does not control or manage the underlying cloud infrastructure. However, they have control over operating systems, storages, and deployed applications (MELL; GRANCE, 2011). Amazon, Google, Microsoft, and Rackspace are the main IaaS providers. Those companies enable fast deployment of various computational resources paying only for the effectively used data and resource. Elastic Compute Cloud (EC2) is one of the pioneers and most successful IaaS product followed by Google Compute Cloud, and Microsoft Azure.

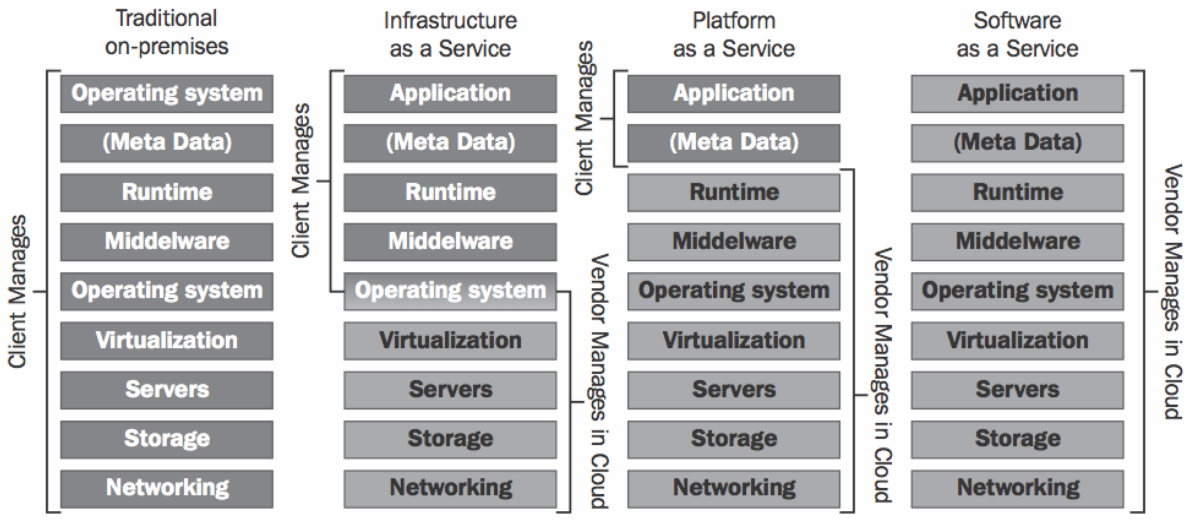


Figure 2.3: Cloud computing service models (SCHOUTEN, 2014).

2.2 Eucalyptus Platform

Eucalyptus (EUCALYPTUS, 2015a) is an open source software for building private and hybrid clouds. This software is a platform that allows high scalability as well as it allows

dynamically increase or decrease the number of resources depending on the workload. This platform focuses on IaaS, where users can provide their own collections of resources (hardware, storage, and network) through a self-service interface according to their needs (EUCALYPTUS, 2015b). The platform is compatible with commercial distributions of Linux (e.g., CentOS and Red Hat Enterprise Linux) and adopts Java in the core as well as in the user interface. Eucalyptus platform provides support for Linux and Windows (since 2008), so that it can be used as virtual instances within virtual machines.

Eucalyptus platform is composed of five essential components: Cloud Controller (CLC), Object Storage Provider (OSP), Cluster Controller (CC), Storage Controller (SC) and Node Controller (NC) (EUCALYPTUS, 2015b). Figure 2.4 shows an example of Eucalyptus-based cloud computing environment components.

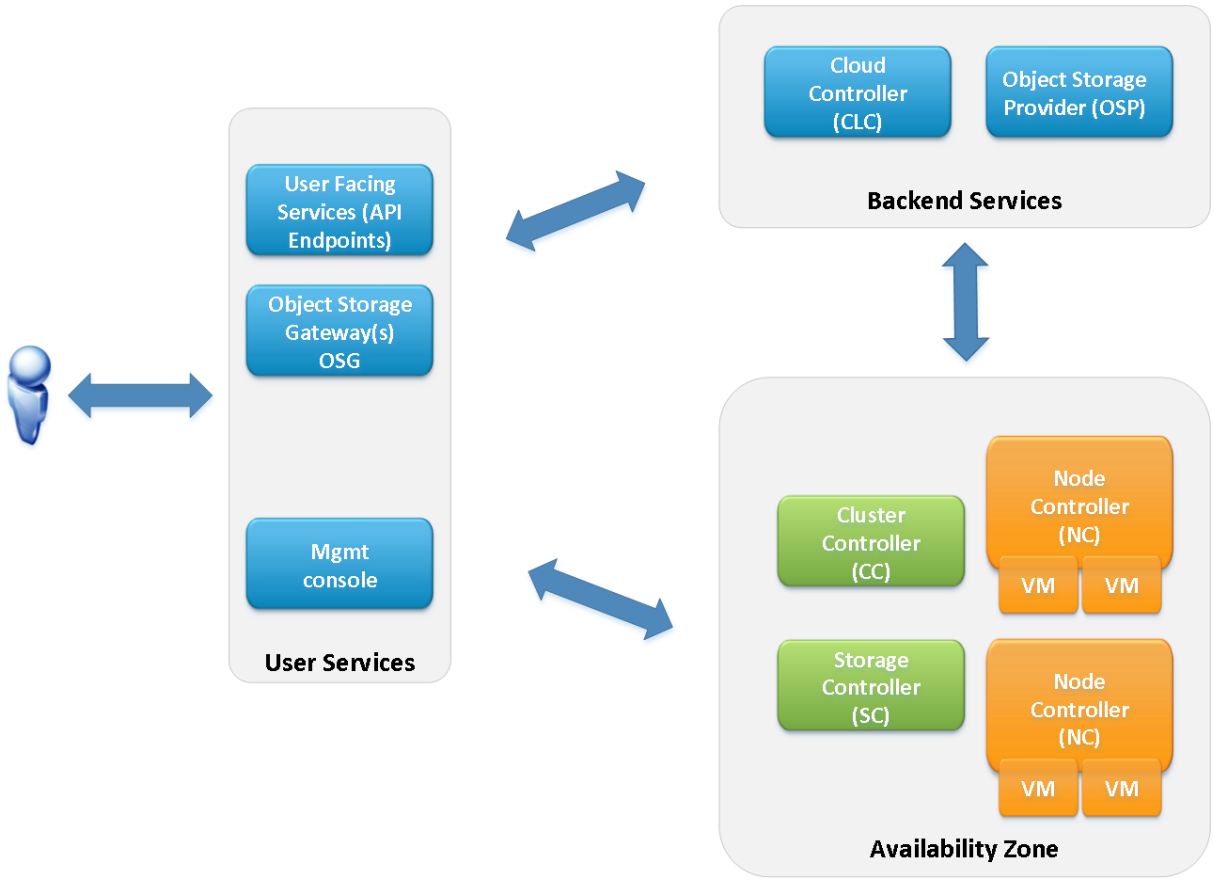


Figure 2.4: Example of Eucalyptus components (EUCALYPTUS, 2015b).

2.2.1 Cloud Controller

The Cloud Controller (CLC) is the front-end of the entire cloud infrastructure, exposing and managing the underlying virtualized resources (servers, network, and storage) (EUCALYPTUS, 2015b). This component uses web services interfaces to receive the client requests on one side and interact with the remaining Eucalyptus components on the other (EUCALYPTUS, 2015a). Additionally, this component is responsible for:

- Monitoring the availability of resources across multiple components of the cloud infrastructure, including hypervisor nodes that are used to actually provision the instances, and the cluster controllers which manage the hypervisor nodes.
- Resource arbitration – decide the clusters used for provisioning the instances.
- Monitoring the running instances.

2.2.2 Object Storage Provider

The Object Storage Provider (OSP) allows users to store persistent data, which is organized as eventually-consistent buckets and objects. It allows users to create, delete, list buckets, put, get, and delete objects, and set access control policies. They can be either the Eucalyptus Walrus backend or Riak CS. Walrus and Riak CS are interfaces compatible with Amazon's S3, and supports the Amazon Machine Image (AMI) which is an image-management interface. Therefore, Walrus provides a mechanism for storing and accessing both the virtual machine images and user data (EUCALYPTUS, 2015a). The main functions of this component are:

- Store virtual machines images.
- Store snapshots of the virtual instances.
- Store and provide files using Amazon's Simple Storage Service (S3) API.

2.2.3 Cluster Controller

Cluster Controller (CC) generally executes on a cluster front-end machine or any machine that has network connectivity to both the nodes running NCs and to the machine running the CLC. CCs gather information about a set of VMs and schedules VM execution on specific NCs. The CC also manages the virtual instance network and participates in the enforcement of Service Level Agreements (SLAs) as directed by the CLC (EUCALYPTUS, 2015a). The main functions of this component are:

- To receive requests from CLC to deploy instances.
- To decide which NCs to use for deploying the instances on.
- To control the virtual network available to the instances.
- To collect information about the NCs registered with it and report it to the CLC.

2.2.4 Storage Controller

The Storage Controller (SC) implements a block-accessed network storage (e.g., EBS), which is capable of interfacing with various storage systems (e.g., Network File System (NFS), Internet Small Computer System Interface (iSCSI), etc.). An elastic block store is a Linux block device that can be attached to a virtual machine, but it sends disk traffic across the locally attached network to a remote storage location (EUCALYPTUS, 2015a). The main functions of the SC are:

- Creation of persistent EBS devices.
- Providing the block storage over Ethernet or iSCSI protocol to the instances.
- Allowing creation of snapshots of volumes.

2.2.5 Node Controller

The NC runs on each node and controls the life cycle instances running on the node. The NC interacts with the Operating System (OS) and the hypervisor running on the node on one side and the CC on the other side. NC queries the operating system running on the node to discover the node's physical resources – the number of cores, the memory size, and available disk space. Additionally, NC is able to report the state of VM instances running on the node and propagates this data to the CC (EUCALYPTUS, 2015a). Basically, the NC functions are:

- Collection of data related to the resource availability and utilization on the node and reporting the data to CC.
- Instance life cycle management.

2.3 Performance and Dependability evaluation

Specific techniques are required for achieving the desired level of quality of service, which might include prototyping, measurement, and modeling. (BUKH; JAIN, 1992) established significant guidelines to evaluation techniques. Performance evaluation can be classified into performance modeling and performance measurement (JOHN; EECKHOUT, 2005). Figure 2.5 shows the classification of performance evaluation.

The most direct method for performance evaluation is based on actual measurement of the system under study. Although measurement techniques can provide exact answers regarding the performance, measurements are not possible if a new system is being planned or improved. Performance measurement is possible only if the system of interest is available for measurement and only if one has access to the parameters of interest. Building a prototype of the upcoming system just for measurement purposes is not always feasible, due to timing and budget constraints.

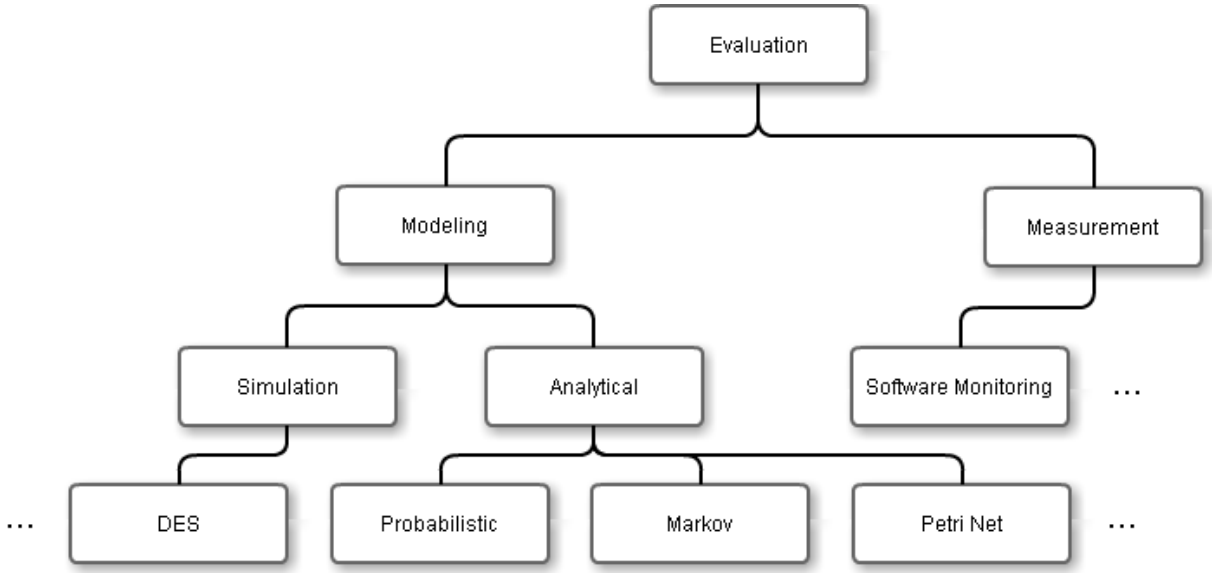


Figure 2.5: Performance Evaluation Techniques.

Furthermore, the measurement results may or may not be accurate depending on the current states of the system, in which such a technique has been performed. Performance measurement may further be classified into on-chip hardware monitoring, off-chip hardware monitoring, software monitoring, and microcoded instrumentation (JOHN; EECKHOUT, 2005).

Modeling methods are typically used in early stages of the design process, when actual systems are not available for measurement or if the actual systems do not have test points to measure every detail of interest. Therefore, analytical modeling and simulation are the techniques usually chosen when a new infrastructure is being designed. Performance modeling may further be classified into simulation modeling and analytical modeling.

The analytical models rely on probabilistic models, queueing theory, Markov models or Petri nets (JOHN; EECKHOUT, 2005). The basic principle of the analytic approaches is to represent the formal system description either as a single equation from which the interested measures can be obtained as closed-form solutions or as a set of system equations from which exact or approximate metrics can be calculated through numerical methods (BOLCH et al., 2006). However, in order to be able to have tractable solutions, often, simplifying assumptions are made regarding the structure of the model. As a result, analytical models do not capture all the details typically built into simulation models (JOHN; EECKHOUT, 2005).

An alternative to analytical models is the adoption of simulation-based models. The results obtained through simulation approaches have not been so accurate as the ones provided by measurement techniques, but it is possible to calculate the estimates precision. The principal drawback of simulation models, however, is the time taken to run such models for large, realistic systems, particularly when results with high accuracy (i.e.: narrow confidence intervals) are desired. Simulation approaches deal with a statistical investigation of output data of performance analysis, and the verification and validation of simulation experiments. Simulation models may further be classified into numerous categories depending on the level of detail of the simulation.

Performance and dependability modeling of computer systems enable one to represent the behavior of a system and compute measures which describe, in a quantitative way, the service provided and the confidence on the system operation. Assuming performance evaluation techniques, metrics of interest can be: response time, throughput, and level of resource utilization. The response time quantifies how long the user must wait for a response to a query, regardless of the quality of the response. The throughput refers to the amount of data transferred or work completed in a given time, and the level of resource utilization measures the proportion of computational resource (e.g., CPU, memory, disk storage) that is actually used. These metrics are directly related to the user perception of system performance and they may also highlight the need for improvements. Besides performance, dependability aspects are on great attention on both sides the industry as well as on the academy for the assurance of quality of service provided by a system. System dependability can be understood as the ability to deliver a specified functionality that can be justifiably trusted (AVIZIENIS et al., 2004). Dependability studies look for determining reliability, availability, security, and safety attributes for the infrastructure under analysis (MALHOTRA; TRIVEDI, 1994).

Formal techniques may be used for modeling computer systems and estimating measures related to system availability, reliability, and performance, and distinct model types may be hierarchically combined (e.g., Reliability Block Diagrams (RBD), Markov chains, Stochastics Petri net (SPN)) when dealing with large systems, enabling the representation of many kinds of dependency between components.

2.3.1 System Classifications

This dissertation adopts a composition of stochastic modeling, to evaluate both availability and performance of a cloud storage service system. We can understand system as a combination of components that act together to perform a function not possible with any of the individual parts (IEEE Standard Dictionary of Electrical and Electronic Terms). The system classifications are important in order to understand the adopted models and simulation mechanism used in this work. Figure 2.6 depicts these classifications.

The systems are divided into two main groups, *Static*, system that does not depend on the past, and *Dynamic*, which are systems whose output depends on the input. The *Dynamic* systems can be divided into *time-varying* and *time-invariant*, where the behavior of the latter does not change with time. The *time-invariant* systems, also called stationary systems, are divided into *linear* and *non-linear* systems. The *non-linear*, which is a system whose performance cannot be described by equations, is divided into *discrete-state* and *continuous-state* systems. The *discrete-state* system, where the state variables are elements of a discrete set, is divided into *event-driven* and *time-driven* systems. The *event-drive* system, in which the state is changed by the occurrence of an event, is divided into *stochastic* and *deterministic* systems. The stochastic system, which one or more of its output variables is a random variable, is divided into *discrete-*

time and *continuous-time*. In continuous-time systems, all input, state, and output variables are defined for all possible values of time.

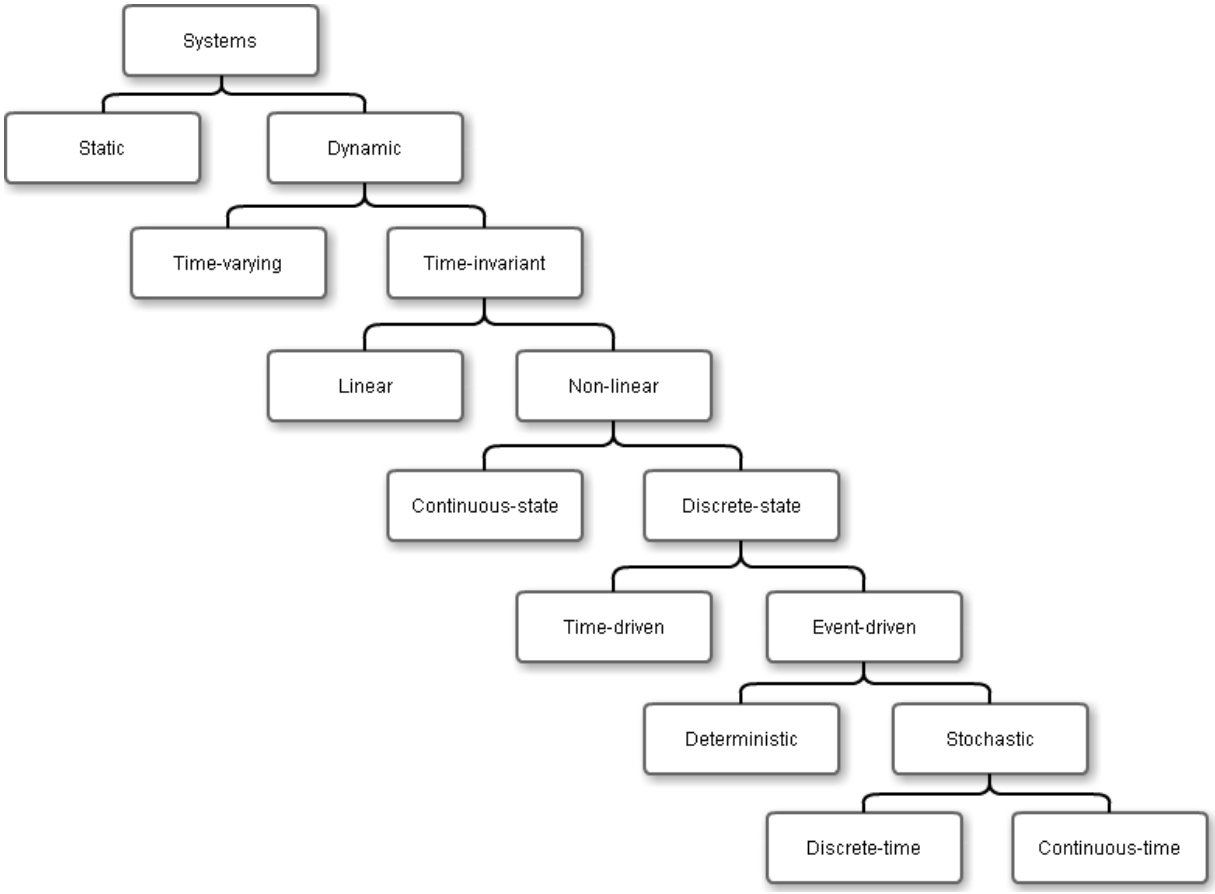


Figure 2.6: System classifications.

2.4 Petri Nets

Petri nets (PNs) are a graphical and mathematical modeling tool that can be applied in several types of systems and allow the modeling of parallel, concurrent, asynchronous and non-deterministic systems (MURATA, 1989). PNs are a useful tool that can deal with cloud computing systems (GOMES; COSTA, 2014) and (RIBAS et al., 2014). Since its seminal work, many representations and extensions have been proposed for allowing more concise descriptions and for representing system features not observed in the early models. Thus, the simple Petri net has subsequently been adapted and extended in several directions, in which timed, stochastic, high-level, object-oriented and coloured nets are a few examples of the proposed extensions.

The PNs, also called a marked Place/Transition Petri nets, is a bipartite directed graph, usually defined as follows:

Definition 2.4.1. ***(A Petri net)*** *(MURATA, 1989) is a 5-tuple:*

$$PN = (P, T, F, W, M_0)$$

where:

1. $P = \{p_1, p_2, ..., p_m\}$ *is a finite set of places;*
2. $T = \{t_1, t_2, ..., t_n\}$ *is a finite set of transitions;*
3. $F \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs (flow relation);*
4. $W : F \rightarrow \{1, 2, 3, ...\}$ *is a weight function;*
5. $M_0 : P \rightarrow \{0, 1, 2, 3, ...\}$ *is the initial marking;*

This class of Petri net has two kinds of nodes, called places (P) represented by circles and transitions (T) represented by bars, such that $P \cap T = \{\}$ and $P \cup T \neq \{\}$. Figure 2.7 depicts the basic elements of a simple PN. The set of arcs F is used to denote the places connected to a transition (and vice-versa). W is a weight function for the set of arcs. In this case, each arc is said to have multiplicity k, where k represents the respective weight of the arc.

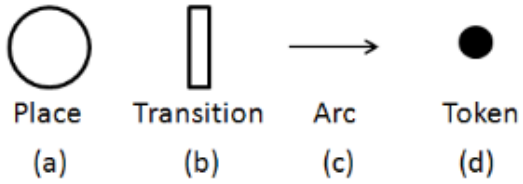


Figure 2.7: Petri net basic elements.

Places and transitions may have several interpretations. Using the concept of conditions and events, places represent conditions, and transitions represent events, such that an event may have several pre-conditions and post-conditions. The behavior of many systems can be described in terms of system states and their changes. In order to simulate the dynamic behavior of a system, a state (or marking) in a Petri net is changed according to the following firing rule:

1. A transition t is said to be enabled if each input place p of t is marked with at least the number of tokens equal to the multiplicity of its arc connecting p with t. Adopting a mathematical notation, an enabled transition t for given marking m_i is denoted by $m_i[t >$, if $m_i(p_j) \geq W(p_j, t), \forall p_j \in P$.
2. An enabled transition may or may not fire (depending on whether or not the respective event takes place).
3. The firing of an enabled transition t removes tokens (equal to the multiplicity of the input arc) from each input place p and adds tokens (equal to the multiplicity of the output arc) to each output place p'. Using a mathematical notation, the firing of a transition is represented by the equation $m_j(p) = m_i(p) - W(p,t) + W(t,p), \forall p \in P$. If a marking m_j is reachable from m_i by firing a transition t, it is denoted by $m_i[t > m_j$.

Figure 2.8 (a) shows the mathematical representation of a Petri net model with three places (p_0, p_1, p_2) and one transition (t_0). Additionally, there is one arc connecting the place p_0 to the transition t_0 with a weight of two, one arc from the place p_1 to the transition t_0 with a weight of one, and one arc connecting the transition t_0 to the place p_2 with a weight of two. The initial marking (m_0) is represented by three tokens in the place p_0 and one token in the place p_1. Figure 2.8 (b) outlines its respective graphical representation and Figure 2.8 (c) provides the same graphical representation after the firing of t_0. For this example, the set of reachable markings is m = $\{m_0 = (3,1,0), m_1 = (1,0,2)\}$. The marking m_1 was obtained by firing t_0, such that, $m_1(p_0)$ = 3 - 2 + 0, $m_1(p_1)$ = 1 - 1 + 0, and $m_1(p_2)$ = 0 - 0 + 2.

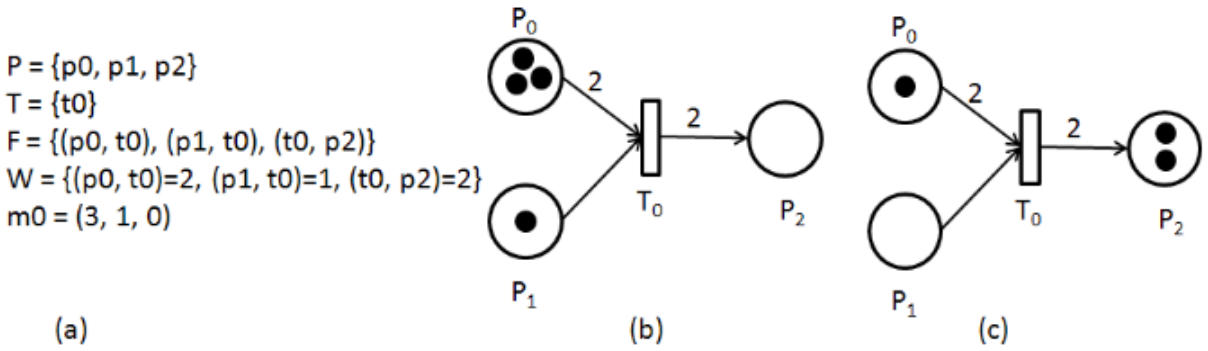


Figure 2.8: (a) Mathematical formalism; (b) Graphical representation before the firing of t_0; (c) Graphical representation after the firing of t_0.

2.4.1 Stochastic Petri Nets

Stochastic Petri Nets (SPNs) are an extension of Petri nets, where the transitions are associated with time allowing the evaluation of performance and dependability metrics. In SPN, a random time variable is associated with transitions which can be fireable through an exponential distribution of probability (TRIVEDI, 2008). The SPN can be mapped to Continuous-time Markov Chains (CTMC), which is useful because building a Markov model manually may be tedious and error prone, especially when the number of states becomes very large. SPNs also allow the adoption of simulation techniques for obtaining dependability and performance metrics as an alternative to the generation of a CTMC, which is sometimes prohibitive due to the state-space explosion. The extended stochastic Petri net (GERMAN, 2000) definition adopted in this work is:

Let $SPN = (P, T, I, O, H, \Pi, M_0, Atts)$ be a stochastic Petri net, where

- $P = \{p_1, p_2, ..., p_n\}$ is the set of places, which may contain tokens and form the discrete state variables of a Petri net.
- $T = \{t_1, t_2, ..., t_m\}$ is the set of transitions, which model active components.
- $I \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ is a matrix of marking-dependent multiplicities of input arcs, where i_{jk} entry of I gives the (possibly marking-dependent) arc multiplicity of input arcs

from place p_j to transition t_k [$A \subseteq (P \times T) \cup (T \times P)$ — set of arcs]. A transition is only enabled if there are enough tokens in all input places.

- $O \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is a matrix of marking dependent multiplicities of output arcs, where o_{jk} entry of O specifies the possibly marking-dependent arc multiplicity of output arcs from transition t_j to place p_k. When a transition fires, it removes the number of tokens specified by the input arcs from input places and adds the number of tokens given by the output arcs to all output places.

- $H \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is a matrix of marking-dependent multiplicities describing the inhibitor arcs, where h_{jk} entry of H returns the possibly marking-dependent arc multiplicity of an inhibitor arc from place $p_j$to transition t_k. In the presence of an inhibitor arc, a transition is enabled to fire only if every place connected by an inhibitor arc contains fewer tokens than the multiplicity of the arc.

- $\Pi \in \mathbb{N}^m$ is a vector that assigns a priority level to each transition. Whenever there are several transitions fireable at one point in time, the one with the highest priority fires first and leads to a state change.

- $M_0 \in \mathbb{N}^n$ is a vector that contains the initial marking for each place (initial state).

- $Atts : (Dist, W, G, Policy, Concurrency)^m$ comprises a set of attributes for the m transitions, where

    - $Dist \in \mathbb{N}^m \to \mathscr{F}$ is a possibly marking dependent firing probability distribution function. In a stochastic timed Petri net, the time has to elapse between the enabling and firing of a transition. The actual firing time is a random variable, for which the distribution is specified by $\mathscr{F}$. We differ between immediate transitions ($\mathscr{F} = 0$) and timed transitions, for which the domain of $\mathscr{F}$ is $(0, \infty)$.
    - $W \in \mathbb{R}^+$ is the weight function that represents a firing weight w_t for immediate transitions or a rate λ_t for timed transitions. The latter is only meaningful for the standard case of timed transitions with exponentially distributed firing delays. For immediate transitions, the value specifies a relative probability to fire the transition when there are several immediate transitions enabled in a marking, and all have the same probability. A random choice is then applied using the probabilities w_t.
    - $G \in \mathbb{N}^n \to \{true, false\}$ is a function that assigns a guard condition related to place markings to each transition. Depending on the current marking, transitions may not fire (they are disabled) when the guard function returns false. This is an extension of inhibitor arcs.

- *Policy* $\in \{prd, prs\}$ is the preemption policy (*prd — preemptive repeat different* means that when a preempted transition becomes enabled again the previously elapsed firing time is lost; *prs — preemptive resume*, in which the firing time related to a preempted transition is resumed when the transition becomes enabled again),
- *Concurrency* $\in \{ss, is\}$ is the concurrency degree of transitions, where *ss* represents single server semantics and *is* depicts infinity server semantics in the same sense as in queueing models. Transitions with policy *is* can be understood as having an individual transition for each set of input tokens, all running in parallel.

In many circumstances, it might be suitable to represent the initial marking as a mapping from the set of places to natural numbers ($m_0 : P \rightarrow \mathbb{N}$), where $m_0(p_i)$ denotes the initial marking of place p_i and $m(p_i)$ denotes a reachable marking (reachable state) of place p_i.

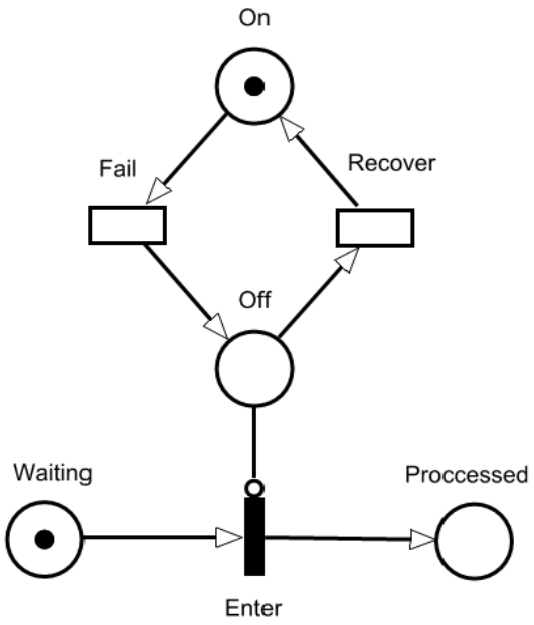


Figure 2.9: Example of a SPN Model.

Figure 2.9 presents an example of a SPN model. The circles represent places, while rectangles model the transitions. Directed arcs connect places to transitions or transitions to places. Inhibitor arc is a special arc type that depicts a small white circle at one edge, instead of an arrow, and they are used to disable a transition if there are tokens present in its input place. The small circles (tokens) present in the places define the state of the SPN. Immediate transitions represent instantaneous activities, and they have higher firing priority than timed transitions.

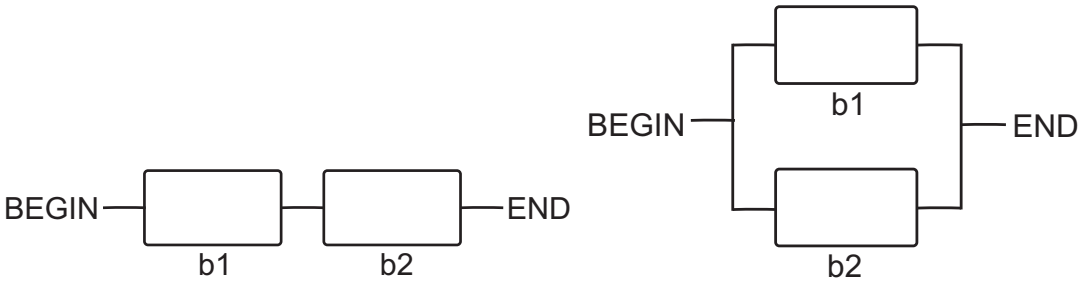
The behavior of the SPN is defined by the structure of the Petri net, as each component is linked or associated with one another. In the example of Figure 2.9, the modeled system starts in the operational state (indicated by a token in the place *On*). The fire of the *Fail* transition removes

a token from the place *On* and generates a token in the place *Off*, representing the failure of the system. In this case, a token in the place *Off* inhibits the firing of the immediate transition *Enter*. The recovery of the system is modeled by the firing of the *Recover* transition. In this state, the transition *Enter* can be fired and the token is removed from the place *Waiting* and added in place *Proccessed*. Note that the *Fail* and *Recover* transitions can be associated with exponential time distributions. The availability of this system is computed based on the probabilities of finding a given number of tokens in the place *On* ($P\{\#On{=}1\}$).

2.5 Reliability Block Diagrams

Reliability is defined as the probability that a system will satisfactorily perform its specific purpose for a given period until the first failure (MACIEL et al., 2011). Reliability Block Diagrams (RBD) (EBELING, 2004) are networks of functional blocks connected according to the effect of each block failure on the system reliability and are based on a combinatorial model initially proposed as a technique for calculating systems reliability by means of block diagrams. RBDs indicate how the operational state (down or functioning) of the system's components affect the functioning of the system. Subsequently, this technique was extended for the calculation of other measurements such as availability and maintainability, which together with reliability, security, and integrity, become part of the concept of dependability (KUO; ZHU, 2012).

In RBDs, the system state is described as a Boolean function of states of its components or sub-systems, where the Boolean function is evaluated as true whenever at least the minimal number of components is operationally enabled to perform the intended functionality (MACIEL et al., 2011) and (KUO; ZHU, 2012). The system state may also be described by the respective structure functions of its components or sub-systems so that the system structure function is evaluated to 1 (one) whenever at least the minimal number of components are operational. Basically, the components can be configured as a series or in parallel arrangement and more generic structures, such as bridges, k-out-of-n, or a combination of previous compositions. Figure 2.10 illustrates two examples, where the blocks are positioned in series (Figure 2.10(a)) and in parallel (Figure 2.10(b)).



(a) Series (b) Parallel

Figure 2.10: Reliability Block Diagrams.

When grouped in a series configuration, the system as a whole becomes inoperative if a single component fails. If we consider a system with n independent components, the reliability (instantaneous availability or steady state availability) is obtained by:

$$P_s(t) = \prod_{i=1}^{n} P_i \tag{2.1}$$

where P_i is the reliability - $C_i(t)$ (instantaneous availability $(D_i(t))$ or steady state availability (D_i)) of block b_i.

In the parallel configuration of Figure 2.10(b), the system continues to be operational if one or more of the components is operational. For a system having n independent components, reliability (instantaneous availability or stationary availability) is obtained by:

$$P_s(t) = 1 - \prod_{i=1}^{n} (1 - P_i) \tag{2.2}$$

where P_i is reliability - $C_i(t)$ (instantaneous availability $(D_i(t))$ or availability in a stationary state (D_i)) of block b_i.

The k-out-of-n configuration is a special case of parallel redundancy. This type of configuration requires that at least k of n components are working for the system to be functional. Figure 2.11 depicts a RBD example of 2-out-of-3 system.

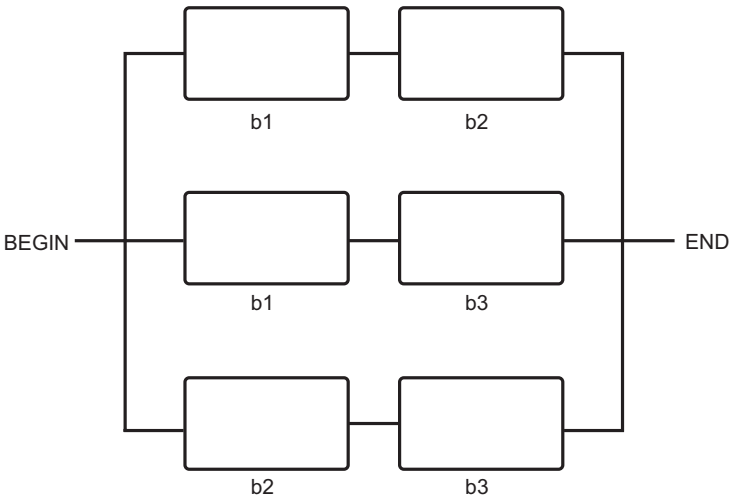


Figure 2.11: Example of a RBD k-out-of-n configuration.

The block diagram indicates that if at least two out of 3 components are functioning (e.g., b1 and b2, or b1 and b3, or b2 and b3) the system is also functioning. The reliability of the system can be evaluated using:

$$P_s(t) = \sum_{i=k}^{n} \binom{n}{i} p^k (1-p)^{n-k} \tag{2.3}$$

where n is the total number of units in parallel, k is the minimum number of units required for system success and P is the reliability of each unit.

For other more complex system configurations, such as the bridge configuration (Figure 2.12) other techniques such as the minimal path set and the minimal cut set, should be used to construct the system state function (DE CARLO, 2013), where equivalent configurations are built in terms of a series or parallel subsystem.

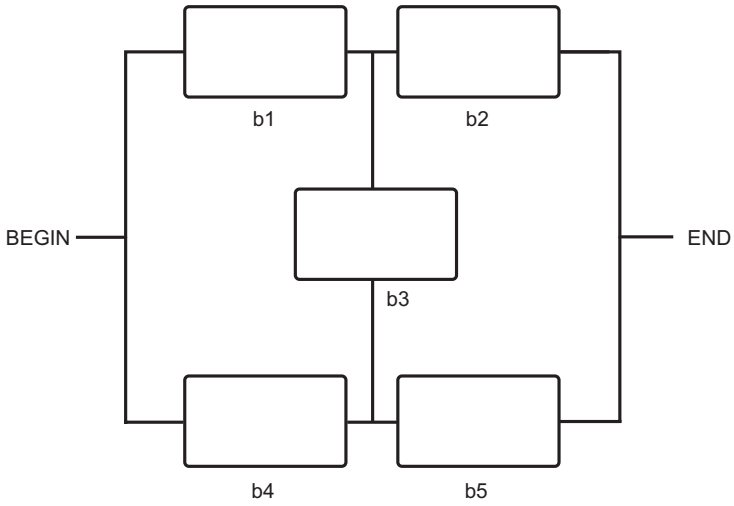


Figure 2.12: Example of an RBD bridge configuration.

2.6 Markov Chains

Markov models are the fundamental building blocks upon which many quantitative analytical performance techniques are built (TRIVEDI, 2008). Such models may be used to represent the interaction among various system components for both descriptive and predictive purposes. It has been used extensively in performance and dependability modeling since the fifties (MACIEL et al., 2011). Besides computer science, the range of applications for Markov models is very extensive. Economics, meteorology, physics, chemistry, and telecommunications are some examples of fields which found in this technique a good approach to address various problems.

A Markov model can be described as a state-space diagram associated with a Markov process, which constitutes a subclass of stochastic processes. A definition of a stochastic process is presented:

Definition 2.6.1. *A stochastic process is a family of random variables* $\{X_t : t \in T\}$ *where each random variable* X_t *is indexed by parameter* $t \in T$*, which is usually called the time parameter if* $T \subset R_+ = [0,\infty)$ *, i.e., T is in the set of non-negative real numbers. The set of all possible values of* X_t *(for each* $t \in T$*) is known as the state space S of the stochastic process (BLOCH et al., 1998).*

Let $P_r\{k\}$ be the probability that a given event k occurs, a Markov process is a stochastic process in which $P_r\left\{X_{t_{n+1}} \leq s_{i+1}\right\}$ depends only on the last previous value X_{t_n}, for all $t_{n+1} > t_n >$

$t_{n-1} > ... > t_0 = 0$, and for all $s_i \in S$. This is the Markov property which means that the future evolution of the Markov process is totally described by the current state and is independent of past states (HAVERKORT, 2001).

In this work, we are only interest on discrete (countable) state space Markov models, also known as Markov chains, which are distinguished in two classes: Discrete-time Markov Chains (DTMC) and Continuous-time Markov Chains (CTMC) (KLEINROCK, 1975). In DTMCs, the transitions between states can only take place at known intervals, that is, step-by-step. Systems where transitions only occur on a daily basis, or following a strict discrete clock are well represented by DTMCs. If state transitions may occur at arbitrary (continuous) instants of time, the Markov chain is represented by a CTMC.

The CTMC are an important class of stochastic processes that have been widely used in practice to determine system performance and dependability characteristics (FELLER, 2008). A CTMC makes transitions from state to state, independent of the past, according to a discrete-time Markov chain. Once entering in a state, the system remains in that state, for an exponentially distributed amount of time, before changing again (MACIEL et al., 2011).

A CTMC can be defined by a transition matrix $P = (P_{ij})$, which describes the state changes on the chain step by step at transition epochs, together with a set of rates $\{a_i : i \in S\}$, the holding time rates (SIGMAN, 1990). Each time a state i is visited, the chain spends, on average, $E(H_i) = 1/a_i$ units of time there before moving on.

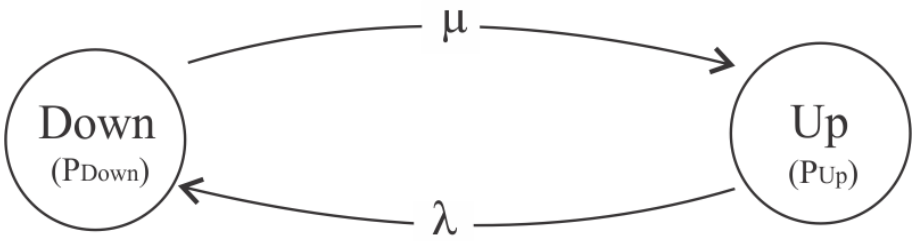


Figure 2.13: Example of a CTMC model.

Markov chains can be represented as a directed graph with labeled transitions, indicating the probability or rate at which such transitions occur. Figure 2.13 depicts an example with two states: *Down* and *Up*. P_{Up} represents the probability of the system being *Up*; P_{Down} represents the probability of the system being *Down*. Therefore, $P_{Up} + P_{Down} = 1$.

When the system is *Up*, and a fault occurs, it can reach the *Down* state at a λ rate. In the *Down* state, when repaired, the system transitions back to the *Up* state at the rate μ. In the steady state, the system availability is stable (BAUER; ADAMS; EUSTACE, 2011) and, thus, the rate of entering in a state i deducted by the rate of leaving that state i corresponds to 0 (zero). Therefore, assuming that the *Up* state is stable, the following equation must be true:

$$P_{Down} \times \mu - P_{Up} \times \lambda = 0 \tag{2.4}$$

Considering that $P_{Up} + P_{Down}$ corresponds to one and adopting the Equation 2.4, the system availability can be computed by the probability of the system being in the P_{Up} state, which can be obtained by:

$$\begin{aligned} P_{Up} \times \lambda &= (1 - P_{Up}) \times \mu \\ P_{Up} \times \lambda &= \mu - P_{Up} \times \mu \\ P_{Up} \times \lambda + P_{Up} \times \mu &= \mu \\ P_{Up} \times (\lambda + \mu) &= \mu \\ P_{Up} &= \frac{\mu}{\lambda + \mu} \end{aligned} \tag{2.5}$$

Adopting MTTF and Mean Time To Repair (MTTR), where $\lambda = 1/MTTF$ and $\mu = 1/MTTR$, then the system availability can be computed by:

$$Availability = \frac{MTTF}{MTTR + MTTF} \tag{2.6}$$

For the analysis using Markov chains, an important aspect that must be kept in mind is the exponential distribution of transition rates. The behavior of events in many computer systems may be fit better by other probability distributions, but in some situations, the exponential distribution is considered an acceptable approximation, enabling the use of Markov models. It is also possible to adopt transition in Markov chains to represent other distributions by means of phase approximation, as shown in (TRIVEDI, 2008).

2.7 Availability Importance

Important performance measures for repairable system designers and operators are system reliability and availability (BARABADY; KUMAR, 2007). Improvement in the system availability has been subject of a large volume of research and articles in the area of reliability. Availability and reliability are important metrics for computing the performance of a system. The system availability and reliability values depend on the system structure as well as on the availability and reliability of each component. Various measures are available for estimating component importance (i.e, Structure Importance, Birnbaum Component Importance, Reliability Criticality Importance), which often relates the contribution of a component to the system's failure. This work adopts AI to identify the most critical components for system availability.

AI measure is a function that relates the failure and repair times and the system structure. The AI measure $\left(I_A^i\right)$ assigns a numerical value between 0 and 1 to each subsystem or component,

so that the value 1 meaning the highest level of importance (BARABADY; KUMAR, 2007). The AI of component i in a system of n components is given as follows:

$$I_A^i = \frac{\partial A_s}{\partial A_i} \tag{2.7}$$

where A_s is the system availability and A_i is the availability of the component i.

Availability importance measure shows the effect of the availability of a subsystem or a component i on the availability of the whole system. The subsystem or component with the largest value has the greatest effect on the availability of the whole system. It is useful to obtain the value of the availability importance of each component in the system prior to deploying resources toward improving a specific component (BARABADY; KUMAR, 2007).

Availability importance measure based on the failure rate/MTTF shows the effect of the failure rate/MTTR of component i on the availability of the whole system, and the failure rate/MTTF of the component with the largest value has the greatest effect on the availability of the whole system (BARABADY; KUMAR, 2007). It can be calculated by Equation 2.8.

$$I_{A,MTTF_i}^i = \frac{\partial A_s}{\partial MTTF_i} = \frac{\partial A_s}{\partial A_i} \times \frac{\partial A_i}{\partial MTTF_i} \tag{2.8}$$

Whilst availability importance measure based on the repair rate/MTTF shows the effect of the repair rate/MTTR of component i on the availability of the whole system, and the repair rate of the component with the largest value has the greatest effect on the availability of the whole system (BARABADY; KUMAR, 2007). It can be calculated by Equation 2.9.

$$I_{A,MTTR_i}^i = -\frac{\partial A_s}{\partial MTTR_i} = -\frac{\partial A_s}{\partial A_i} \times \frac{\partial A_i}{\partial MTTR_i} \tag{2.9}$$

We can apply the availability importance measures to a series system, which consists of n independent subsystems connected in series and which fails when at least one of its components fails. The steady-state availability for a series-system is the product of the component availabilities (EBELING, 2004);(PHAM, 2003):

$$A_s = \prod_{i=1}^{n} A_i = \prod_{i=1}^{n} \frac{MTTF_i}{MTTF_i + MTTR_i} \tag{2.10}$$

Availability importance measures for component i of a series system is given by:

$$I_A^i = \frac{\partial A_s}{\partial A_i} = \prod_{k=1}^{n} A_k; k \neq i \tag{2.11}$$

We can also apply the availability importance measures to a parallel system, which consists of n independent subsystems connected in parallel and which works when at least one of its components works. The steady-state availability of a parallel-system is given by (EBELING,

2004):

$$A_s = \coprod_{i=1}^{n} A_i = \coprod_{i=1}^{n} \frac{MTTF_i}{MTTF_i + MTTR_i} \tag{2.12}$$

Availability importance measures for component i of a parallel system is given by:

$$I_A^i = \frac{\partial A_s}{\partial A_i} = 1 - \prod_{k=1}^{n} (1 - A_k); k \neq i \tag{2.13}$$

The priority in term of increasing availability of the system should be assigned to component i which is the component with the minimum and maximum availability estimate, respectively. To illustrate the applicability of AI, consider the simple system example which is illustrated in Figure 2.14.

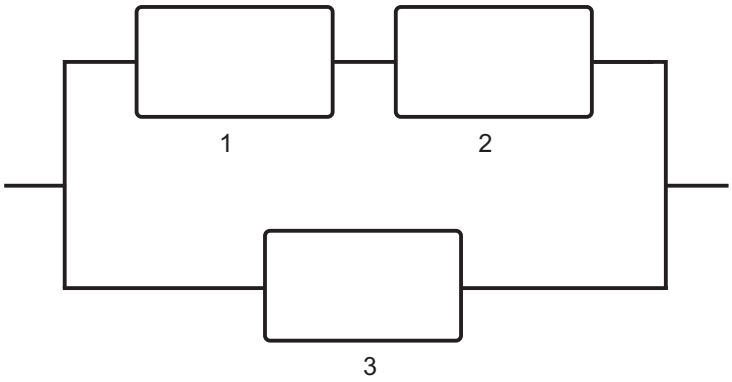


Figure 2.14: Availability Importance numerical example. Adapted from (BARABADY; KUMAR, 2007).

In this system, we have three components with the failure rate and repair rate of each one shown in Table 2.1.

Table 2.1: MTTF and MTTR of all the components.

Component	***MTTF (h)***	***MTTR (h)***
1	142.85	55.55
2	46.72	20
3	57.14	33.33

Based on Equation 2.7, 2.8, and 2.9 the availability importance measures for all components are calculated and shown in Table 2.2.

Table 2.2: Availability importance measures for all the components.

Component	I_A^i	$I_{A,MTTF}^i$	$I_{A,MTTR}^i$
1	0.258	7.430	2.890
2	0.265	2.602	1.114
3	0.496	6.592	3.846

The availability importance measure I_A^i indicates that component 3 has more influence on the availability of the whole system, and therefore, an improvement in the availability of component 3 will result in the highest increase on the system availability. In addition, it is important to highlight the MTTF of component 1, which the effect of this measure on the availability of the whole system, is about 2 times higher than the corresponding effect of the MTTR measure, which is indicated by a comparing $I_{A,MTTF}^i$ and $I_{A,MTTR}^i$.

2.8 Summary

This chapter presented fundamental concepts ranging from the definition of cloud computing to the availability importance index. Initially, cloud computing was presented focusing on the Eucalyptus private platform. Afterwards, the concept of performance and dependability evaluation was presented. After that, attention was devoted to Petri nets models, giving particular focus to stochastic Petri nets. Next, reliability block diagrams were presented focusing on dependability evaluation of systems. Afterwards, the concept of Markov chains and its application was presented. Finally, the availability importance index was shown with an application example.

3

Methodology and Architecture

This chapter starts by presenting the adopted methodology for conducting availability and performance analysis on private cloud storage systems. Afterwards, the private cloud storage architecture is presented, showing their components, characteristics and behaviour.

3.1 Methodology

The adopted strategy is divided into six activities: (i) system understanding, (ii) subsystem modeling, (iii) subsystem evaluation, (iv) creating scenarios based on AI, (v) availability evaluation and (vi) performance evaluation. Figure 3.1 shows the modeling strategy, where each step is described as follows.

- **System understanding:** The first activity concerns the understanding of the system architecture and its components, as well as the physical and logical structure of the private cloud storage services. Physical components like computers, switches and logical components (e.g., OS, databases, hypervisors, web servers, storage services), and others are taken into account. Understanding the system requires great attention and special care from the evaluator to avoid errors of interpretation that may compromise other stages of the methodology. During this step, the system can be partitioned into smaller subsystems, which makes the identification of system design errors easier.

- **Subsystem modeling:** The second activity consists in the creation of availability and performance submodels using analytic models (e.g., SPN, CTMC and RBD) to represent the adopted architecture. Therefore, the result of this step corresponds to the creation of availability as well as performance models. For the availability modeling strategy, we have considered RBD (when no dependency between components is present) and CTMC (when dependency is identified). However, it is also possible to consider SPN for modeling systems with dependency among components.

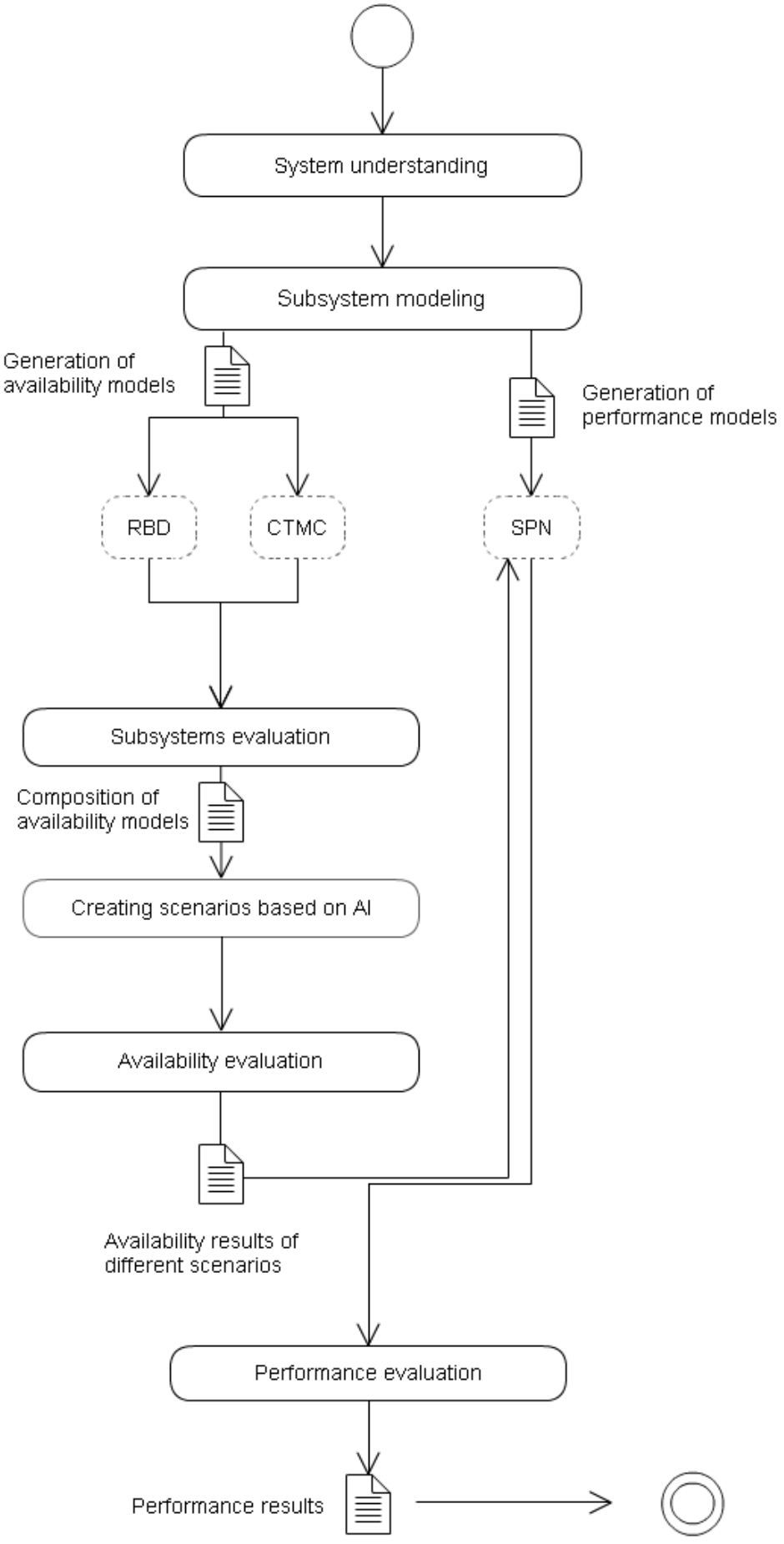


Figure 3.1: Modeling strategy for availability and performance evaluation.

In summary, this modeling strategy aims to adopt the most suitable approach for representing the cloud infrastructure subsystems.

- **Subsystem evaluation:** In the subsystem evaluation activity, the submodels are composed (hierarchical model) in order to represent the failure/repair behaviour of the cloud storage service infrastructure. This modeling strategy considers the advantages of both RBDs and CTMCs to mitigate the complexity for representing cloud infrastructures failure and repair behaviour. In this activity, the cloud infrastructure components are grouped in order to generate subsystems. For each subsystem, the MTTF and MTTR are computed. Once the MTTF and MTTR are calculated, a composite model is generated representing the cloud storage service infrastructure availability, without considering redundant components.

- **Creating scenarios based on AI:** The fourth activity concerns the generation of scenarios through the attribution of redundant components to the adopted architecture (e.g., controller, node, storage, and network). It allows the creation of cloud infrastructures with different redundancy levels according to the availability importance index of each component. It helps the understanding of the influence of each component in the whole system availability, providing support for designers and administrators of storage services.

- **Availability evaluation:** The fifth activity concerns the evaluation of the architectures created before through AI index. Some metrics are obtained such as: MTTF, MTTR, availability, uptime, and downtime. Note that the evaluation of the architecture is initially performed by the CTMC and RBD models, and then, the availability data obtained from such evaluation is used as input parameter for the performance model. In this step, the models are also validated through experiments, so that we compare the results obtained from the model an experiments and validate the former with a certain confidence interval.

- **Performance evaluation:** The last activity uses the results obtained from the previous step (RBD and CTMC models) as input parameter in the SPN model for computing performance metrics like throughput and utilization. The metrics are obtained from the stationary analysis of the SPN model and expressions are adopted to compute each of these metrics.

3.2 Private Storage Cloud Environment

This work adopts a private cloud storage service based on the Eucalyptus platform. This platform was chosen due to the wide utilization by large companies and researchers. Eucalyptus API is strongly compatible with the Amazon Web Services (AWS) public cloud engine, enabling

users to move workloads between AWS and Eucalyptus environments. The compatibility allows the adoption of tools, scripts, and images in both AWS and Eucalyptus, which provides support for the creation of hybrid clouds.

Pydio (SAS, 2015) has been adopted as the storage service due to the fact that it consists of an open source solution for sharing and synchronizing files in the cloud. Individual and enterprise users can host their own data and have control over them using Pydio. A web interface is also available from which individuals can perform upload, download and share data. In addition, Pydio offers applications for smartphones and tablets, allowing the ubiquitous access to data and their synchronization across devices. In summary, Pydio application was chosen because it is open source, widely used, supports a great variety of plugins and storage services, and can acts as a front-end to Amazon S3, Dropbox, Samba servers and others.

Figure 3.2 presents the physical and logical structure of the storage service hosted in a private cloud. The cloud infrastructure is composed of four main components: (i) a controller, which manages communication, instantiation and cloud processes; (ii) a node, where the virtual machine hosting and executing Pydio is instanced; (iii) a storage, which stores the data of the users of the service; and (iv) a switch, which allows communication among all components. The controller is a machine running a Linux operating system (OS) and some Eucalyptus components (e.g., CLC). A user administrator can create and manage virtual machine's instances via the controller. The access can be done through a web interface called Management Console or a command line interface. The storage is another machine running an OS, a Network File System (NFS) server and a Mysql Data Base (DB). The user's authentication data and files are placed in the storage machine by the DB and NFS. By using NFS and DB on the storage machine, user's data are accessible from one or more VMs, allowing users to authenticate and access files from any virtual instance running on the Pydio service.

The node corresponds to a machine running an OS, a hypervisor or a Kernel-based Virtual Machine (KVM) and the eucalyptus node controller component. In the KVM, we have a Virtual Machine (VM), which was created and customized from Virt-Manager (BERRANGé, 2016), running another OS and an Apache Web Server. In this web server, the Pydio service application is deployed. Finally, the network is composed of a gigabit switch connecting all devices.

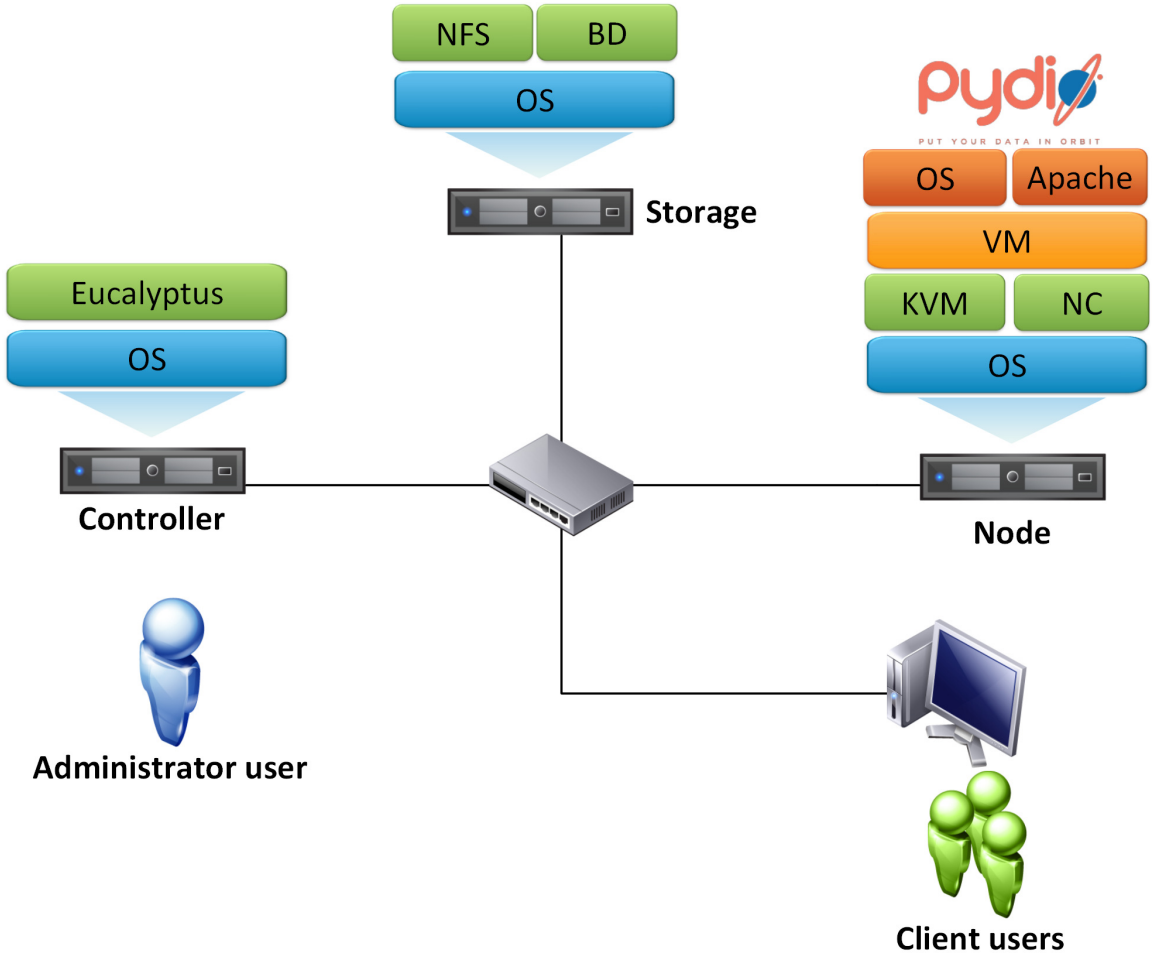


Figure 3.2: Adopted architecture.

The cloud storage service starts when a user performs login to the Pydio server hosted in the VM present in the node. The user authentication data and files are stored on the storage machine. Once performed the login, clients connected to the service can upload/download documents, images, share and sync files.

3.3 Summary

This chapter presented the adopted methodology which is composed of: (i) system understanding, (ii) subsystem modeling, (iii) subsystem evaluation, (iv) creating scenarios based on AI, (v) evaluation and (vi) performance evaluation. Next, the private cloud storage architecture is presented. It is composed of four main components: (i) a controller, which manages communication, instantiation and cloud processes; (ii) a node, where the virtual machine hosting and executing Pydio is instanced; (iii) a storage, which keeps user data within the service; and (iv) a switch, which allows communication between all components, and the sub-components of each module.

4

Models

This chapter shows the proposed RBD, CTMC and SPN models for representing the adopted cloud storage system previously presented in Section 3.2. A hierarchical strategy is adopted where RBD and CTMC models are used to verify hosted service availability (Section 4.1), and an SPN model is used to model service operation and obtain performance (Section 4.2) metrics from the system. The achieved results obtained through the RBD and CTMC models evaluation are used as input parameters for the proposed performance model in described SPN.

This chapter starts by presenting the model for the baseline architecture (without redundancy). Afterwards, the chapter proceeds to present the proposed SPN model for representing the architectural system behavior. Mercury tool (SILVA et al., 2015) has been adopted for creating and evaluating the proposed models.

4.1 Availability Models

RBD and CTMC models were created to evaluate the cloud system presented in Figure 3.2. The architecture is divided into four main modules (node, VM, storage, and controller) and each module is composed of specific components. Besides, a serie RBD is used for connecting each module for representing the whole system service. The following subsections present such models.

4.1.1 Node

The node module is composed of the following components: Hardware (HW), Operating System (OS), KVM, VM (detailed in Figure 4.2), and NC. All these components should be operational in order to the system to be considered available. The VM component is further refined by a CTMC, which allows computing of availability values to be considered in the RBD. This CTMC is proposed because of the interdependency between the system's components. Figure 4.1 presents the RBD model for the node subsystem.

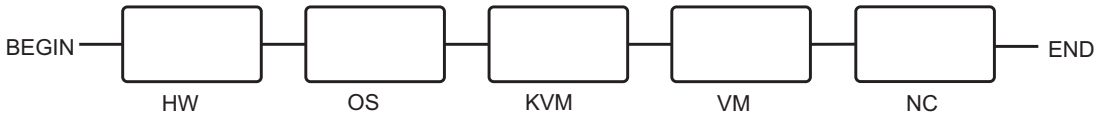


Figure 4.1: Model of the RBD Node Module.

4.1.2 Virtual Machine

Figure 4.2 shows the CTMC model which represents the virtual machine component. This component is composed of: the Operating System (OS), the Apache web server and Pydio. Due to the presence of the VM activation time (or instantiation rate) and interdependency between the system's component, this module can not be modeled by RBD, so a CTMC was used to obtain the VM availability values and pass as an input parameter to VM component of Figure 4.1.

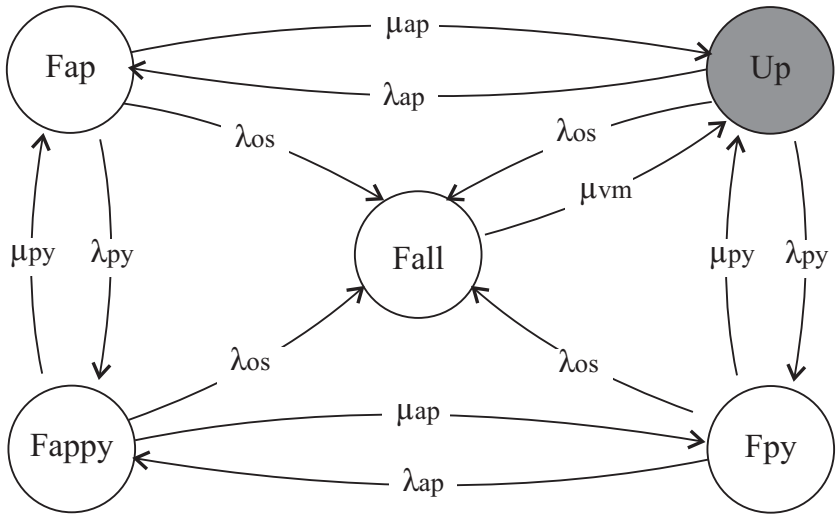


Figure 4.2: CTMC of the VM Module.

The CTMC has five states: *Up*, *Fap*, *Fappy*, *Fpy*, and *Fall*. The gray circle indicates the operational state and the white circles indicate down states (where the service is not available due to failure). Only the *Up* state represents service availability, so that all applications (OS, Apache and Pydio) are working properly. From *Up*, the following states can be reached: Apache application failure (*Fap*), Pydio application failure (*Fpy*), or operating system failure (*Fall*). At *Fap*, service is no longer provided, and from this state, the other three states can be achieved: Apache and Pydio application failure (*Fappy*), Apache application repair (*Up*), or operating system failure (*Fall*). The Apache and Pydio failure state (*Fappy*) also indicate that the service is unavailable. From *Fappy*, the three states can be reached: *Fap*, *Fall*, and *Fpy*. In *Fpy* the service is unavailable due to the failure of the Pydio application, and from this location, it is possible to achieve the states: *Fall*, *Fappy* and *Up*. The *Fall* state represents the failure of all components in the subsystem (Apache, Pydio and operating system). When a new VM instantiation occurs, including all necessary applications, the system returns to the *Up* state. The nomenclature of all

states is summarized in Table 4.1.

Table 4.1: Nomenclature of CTMC states.

State	Description
Fap	Apache failure, so that the service is unavailable
Fappy	Apache and Pydio failure, so that the service is unavailable
Fall	Failure of all components (Pydio, Apache and OS)
Up	Service available
Fpy	Failure of the Pydio, so that the service is unavailable

As it can be seen in Figure 4.2, there are three possible causes of service interruption in the cloud storage: Apache failure, Pydio failure or OS failure. While in the fault state, it is necessary to repair the component that causes the service unavailability. The repair time of Apache, Pydio and OS applications is higher than the instantiation time of a new VM. Therefore, once one of these applications has failed, the service automatically terminates the VM containing the crashed application and instantiates a new VM.

The rates λ_{ap}, λ_{py} and λ_{os} denote the failure rate of the Apache, Pydio and OS, respectively. The repair rates for Apache and Pydio are μ_{ap} and μ_{py}. The value μ_{vm} is the instantiation rate of a new VM, that is, the mean time to activate a VM after being requested. The closed-form Equation 4.1, which is obtained from the CTMC, computes the availability of the VM component. Note that the result computed through the Equation 4.1 is incorporated into the RBD model shown in Figure 4.1.

$$\begin{aligned} A_v = &(\mu_{vm}(\lambda_{ap}\lambda_{so}(\beta) + \lambda_{ap}(\beta_1)\mu_{py} + (\beta_1)(\beta_2)(\beta + \mu_{py}))) \\ &\times ((\mu_{ap} + \beta_1)(\lambda_{so} + \mu_{vm}) \times (\beta)(\lambda_{ap} + \beta + \mu_{py}))^{-1} \end{aligned} \tag{4.1}$$

where

$$\begin{aligned} &\beta = \lambda_{py} + \lambda_{so} + \lambda_{ap}\ , \\ &\beta_1 = \lambda_{so} + \mu_{ap} \text{ and} \\ &\beta_2 = \lambda_{so} + \mu_{py}\ .\ ^{1} \end{aligned} \tag{4.2}$$

4.1.3 Storage

The volume, which stores users' data and files, is composed of a HW, an OS, a NFS server and a data base (DB). For the storage system work effectively, all components of this module must be operational. The RBD model which represents this structure is depicted in Figure 4.3.

[1] β, β_1 and β_2 are only used to group the terms and reduce the size of the Equation 4.1.

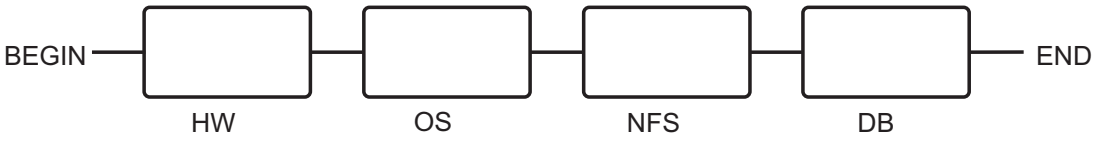


Figure 4.3: Model of the RBD Storage Module.

4.1.4 Controller

The controller module comprises HW, OS and the following Eucalyptus components: CLC, OSP, CC and SC. The RBD for the controller module is presented in Figure 4.4. Similarly to the other modules, for the controller to be considered operational, all the components of this module must be working properly.

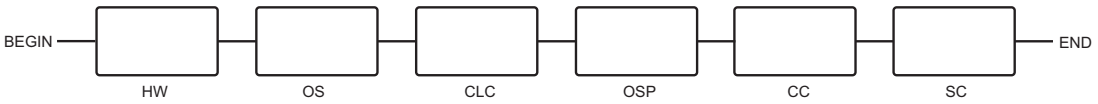


Figure 4.4: Model of the RBD Controller Module.

4.1.5 Composition of models

From the point of view of storage service availability, the service is operational if Controller, Node, Storage, and Network are working properly. Figure 4.5 represents the composition of the RBD models previously described and represented here as a system. This corresponds to the baseline system, which does not consider any redundancy.

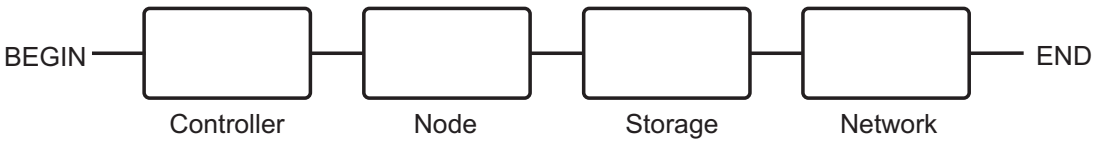


Figure 4.5: RBD of the entire system.

The availability of the entire system (A) can be computed by the equation 4.3.

$$A = \left(\prod_{i \in \{c,no,s,n\}} \frac{\mu_i}{\mu_i + \lambda_i} \right) \tag{4.3}$$

where c, no, s and n correspond to the availability of the controller, node, storage and network, respectively, which are derived from the previous RBD models, and μ_i and λ_i represent the mean failure and repair rates, respectively. The product of the availability ($\frac{\mu_i}{\mu_i+\lambda_i}$) of each one these variables, results in the total availability of the basic architecture that has been represented by A.

4.2 Performance Model

Figure 4.6 shows the proposed SPN model for representing the data storage service in the private cloud architecture presented in Chapter 3. This model has been adopted for computing metrics such as throughput, system utilization, and number of people not attended due to failures. These metrics are detailed as follows.

Metrics analyzed

The following metrics have been considered: Probability of the network buffer to be full (PBF), Probability of timeout occur (POT), Probability the users are not attended due to failures (NUF), System utilization (SU) and System throughput (ST). Table 4.2 shows the expression adopted to compute each metric.

Table 4.2: Expression for the calculation of SPN measurements.

Measure	***Expression***
PBF	$P\{\#BufferNetwork = 0\}$
POT	$P\{(\#PDrop > 0)\}$
NUF	$P\{\#Fail > 0\}$
SU	$E\{\#Queue\}/(E\{\#Queue\} + E\{\#MaxClients\})$
ST	$E\{\#Queue\}/InService$

The PBF metric is defined by the probability of the number of tokens in the place *BufferNetwork* which is equal to 0, meaning that the network no longer supports access to any user. This depletion can occur if the system is failing, not allowing new clients to enter, or if the maximum number of clients supported by the VM is also reached, meaning that the service is running but crowded. The POT metric computes the probability that the user request is not attended due to timeouts. This representing that the service has failed, or the maximum number of clients supported by the service is reached, but the system still have clients waiting to be served, however, it does not allow new customers to enter.

The NUF metric is defined as the probability of the number of tokens in the place *Fail* be higher than 0. Its means that the storage service had clients being served when a failure occurred, causing the unavailability of the system. Then, after a maximum wait time (*TimeOutFail* transition), these customers will leave the service. The SU metric is used to compute the system utilization by getting the expected number of tokens in the place *Queue* divided by the summing of expected tokens in the places *Queue* and *MaxClients*. Finally, the ST metric is the expected value of the number of tokens in the place *Queue* divided by the time spent to download a file, which is the value assigned to the *InService* transition.

Model behavior

In this model, the *Request* transition corresponds to the mean time between user requests (arrival). A token in the *Waiting* place represents the arrival of a user, which may or may not access the private cloud storage system. If the *Drop* transition has fired, the user request can not be attended due to a service unavailability (token in the *PDrop* place). The *TimeOut* transition is the maximum waiting time (e.g.: 60 seconds) for the user to receive a response from the service. The *BufferNetwork* place represents the number of users that can access simultaneously the service using the same network.

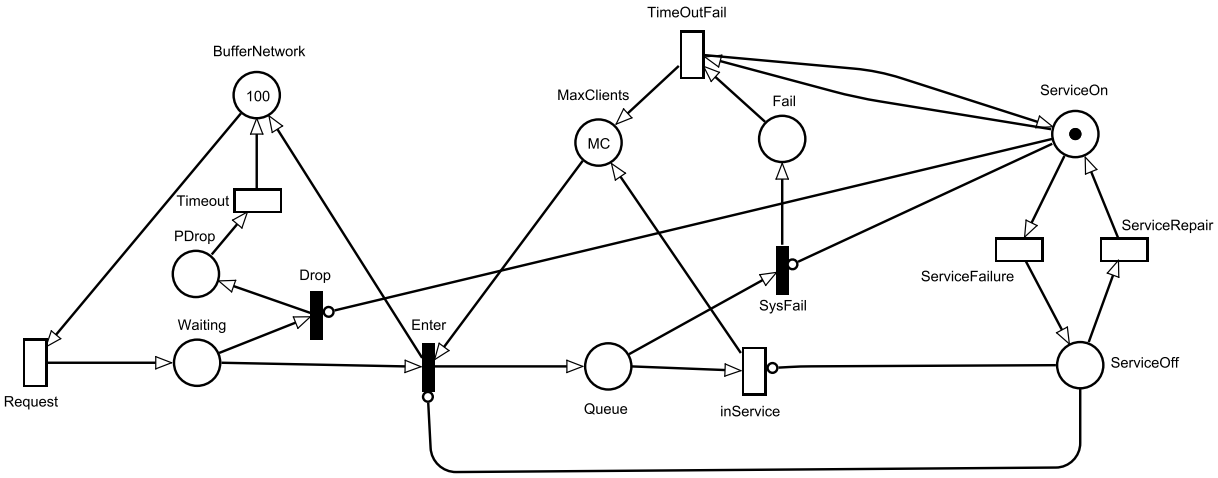


Figure 4.6: SPN model for the cloud storage service.

A user only get access to the service queue (indicated by a token in the *Queue* place) once the transition *Enter* has fired. This transition is immediate, represents instantaneous activities, and have higher firing priority than timed transitions. An inhibitor arc connects the *Enter* transition to a *ServiceOff* place, so in case the service is unavailable (a token in the *ServiceOff* place), *Enter* transition is not enabled and the users are not able to access the system.

The *InService* transition represents the time taken for serving a user. Notice that this transition has an inhibitor arc and an infinite server firing policy (see Table 4.3) in order to properly represent the parallel exclusion of all request if the service is unavailable (a token in the *ServiceOff* place).

The *MaxClients* place represents the maximum number of simultaneous users that the VM can handle. When the user access the system and is being served, the maximum number of users in the system (represented by the label *MC*) is decremented. Finally, when the user's request is made, the token returns to *MaxClients*, releasing access to a new user.

The *Fail* place models the users that were not able to be served due to a service failure. This is preceded by an immediate transition (*SysFail*) with an inhibitor arc that is enabled only if the service is off. That transition removes from the queue the users that would begin to be served, but they were not, due to service failure. The *SysFail* transition is fired if the system has failed,

which adds a token to the *Fail* place.

The *TimeOutFail* transition represents the waiting time that a user can wait in the system without receiving a response due to, for instance, a system failure. This transition has an infinite server firing policy to represent the parallel exclusion of all request if the service is unavailable.

A token in the place *ServiceOn* means that the system is operational. The fire of the *ServiceFailure* transition represents the failure of the system. Therefore, a token is removed from the *ServiceOn* place and a token is added to the place *ServiceOff*. The fire of *ServiceRepair* transition models the repair of the system. It is worth to stress that the results obtained from the evaluation of our previous CTMC and RBD models are used as input parameters for the *ServiceFailure* and *ServiceRepair* transitions.

Table 4.3 shows the attributes of all transitions used in the SPN model. This transitions can be immediate (imm) or exponential (exp) and have a firing policy with semantic infinite server (IS) or exclusive server (ES). The immediate transitions Drop, Enter and SysFail have weight and priority equal to 1, then they have higher firing priority than the exponential ones.

Table 4.3: Transitions attributes.

Transitions	***Type***	***Weight***	***Priority***	***Semantic***
Request	exp	-	-	ES
TimeOut	exp	-	-	ES
InService	exp	-	-	IS
Drop	imm	1	1	-
Enter	imm	1	1	-
SysFail	imm	1	1	-
TimeOutFail	exp	-	-	IS
ServiceFailure	exp	-	-	ES
ServiceRepair	exp	-	-	ES

4.3 Summary

In this chapter, models were proposed to conduct the integrated evaluation of performance and availability of data storage service hosted in a private cloud. First, we presented RBD models used to represent the node, storage and controller module. The VM module was modeled through a Continuous-time Markov Chains (CTMC) due to the presence of the VM activation time. Afterwards, a composition of these models is performed to represent the whole system. Lastly, the proposed SPN model used for performance evaluation is described.

5

Case Study

This chapter presents case studies where availability and performance evaluation of a private cloud storage service is carried out. The main goal of this section is to show the applicability of the proposed models, which considers a hierarchical approach that uses the main strengths of RBD, CTMC and SPN models. Note that the evaluation was initially performed by CTMC and RBD models, and then the results obtained from such evaluation is incorporated into the SPN model to compute the performance related metrics of the adopted architecture. The first case study (Section 5.1) validates the SPN model by comparing with the real system, the second (Section 5.2) evaluates the system availability scenarios with different degrees of redundancy, and the third case study (Section 5.3) evaluates the system performance through the SPN model.

5.1 Case Study I - Validation

This section shows how the environment of the adopted architecture was configured to perform the experiments and how the experiments were conducted to validate the models. The validation process verifies the correspondence between the results obtained from the analytic models and the results acquired from a real infrastructure. More specifically, in this work, we validated the throughput metric for the architecture shown in Section 3.2.

For the test bed experiments, we adopted the Eucalyptus version 4.2.1, running over the CentOS version 6.6. We also created a Eucalyptus VM instance running Ubuntu 14.04 for the storage service hosted in the private cloud infrastructure.The private cloud considered is composed of three identical machines with CPU Intel Core i5 3.4 GHz, 8 GB RAM, 1 TB HD, and 1000 Mbps NIC. The machine which represents the Controller is used to manage the cloud, to create and destroy instances, to register users and to command all communication in the system. The machine representing the Node is responsible for the execution of the VM where Pydio solution is installed. The VM instance used for the Pydio solution was the m1.medium type containing 1 CPU with 512MB RAM. The Storage, represented by the third machine, stores data and user files. All the components are linked across a network of 1000 Mbps.

The validation was performed by comparing the measured throughput in the real infras-

tructure to the throughput obtained from the stationary analysis of the SPN model shown in Figure 4.6. First, we measured the time taken to download each file size as presented in column InService of Table 5.1. Next, we measured the number of files downloaded for each file size during one-hour analysis period using Apache HTTP server benchmarking tool ab. For this analysis, we also considered a number of concurrent users using the Jmeter tool halili2008apache. The sizes of the files downloaded from the Pydio service (VM instance) and the number of concurrent users (see MaxClients and File size columns) adopted for the experiments are shown in Table 5.1.

Afterwards, we computed the throughput of the downloads for the one-hour analysis period. This throughput metric is based on the Little's law, according to the expression $Throughput = NumberInSystem/ResponseTime$. For this work, *ResponseTime* represents the average time taken to download each file size, while *NumberInSystem* indicates the number of files downloaded for each file size in a given period of time. Lastly, the parameters used and obtained from the experiments (see Table 5.1) are adopted for the SPN model to compute the throughput. We also assumed a network buffer of 100 clients and average times of 60s for both the *TimeOut* and *TimeOutFail* transitions. In the Subsection 4.2, we have detailed the metric adopted to compute the throughput of the SPN model.

Table 5.1: Parameters used for the experiments.

File size	***MaxClients***	***InService (s)***
50 MB	150	0.45
100 MB	75	0.89
300 MB	25	2.7
700 MB	10	6.25
1024 MB	7	9.0

Table 5.2 shows the results obtained through the measurements in the real system and results computed from the proposed SPN model. The column *Measured Throughput* represents the measured throughput in the real system, while the column *Model Throughput* corresponds to the results obtained from the SPN model. In addition, the column *Confidence Interval* shows the 95% confidence interval for the measured throughput. Note that the mean values of the throughput obtained from the SPN model are within the confidence interval of the measured values in the real system. Therefore, the results provided by the analytic model are consistent with the test bed experiments using the same input parameters.

Table 5.2: Results obtained from the measurements and models.

File size	*Measured Throughput*	*Confidence interval*	*Model Throughput*
50 MB	7901	(7803;7999)	7875
100 MB	4021	(3958;4084)	3969
300 MB	1341	(1308;1373)	1311
700 MB	566	(551;580)	567
1 GB	398	(390;406)	390

Figure 5.1 depicts a graphical comparison between the results obtained through the measurements of the real system and the results computed through the proposed model, taking into account each file size and the number of downloaded files.

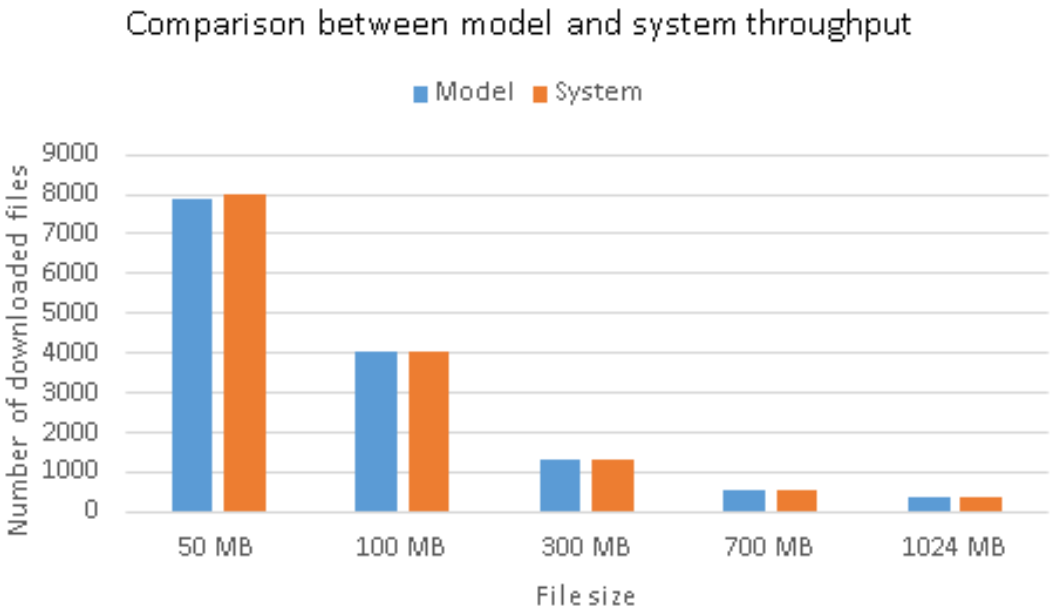


Figure 5.1: Throughput graphical comparison .

5.2 Case Study II - System Availability

The purpose of this case study is to evaluate the system availability scenarios with different degrees of redundancy. This section shows the availability results whose analytical models are represented in Section 4.1. The MTTF and MTTR parameters, for each RBD individual components and CTMC, were derived from (DANTAS et al., 2012) and (KIM; MACHIDA; TRIVEDI, 2009). Table 5.3 presents these parameters. The availability of the VM component is calculated from the CTMC illustrated in Figure 4.2, and all the values required to solve this model are shown in Table 5.4. It should be highlighted that the time to instantiate a new VM was measured in the real environment that we set up for our experiments because such a time can vary according to the application adopted.

Table 5.3: Parameters of the RBD models.

Module	***Entry Parameters***		
	Component	***MTTF***	***MTTR***
VM	OS	2893h	0.25h
	Pydio	336h	1h
	Apache	788.4h	1h
Node	HW	8760h	1.66h
	OS	2893h	0.25h
	KVM	2990h	1h
	VM	217.8h	0.94h
	NC	788.4h	1h
Storage	HW	8760h	1.66h
	OS	2893h	0.25h
	NFS	788.4h	1h
	DB	788.4h	1h
Controller	HW	8760h	1.66h
	OS	2893h	0.25h
	CLC	788.4h	1h
	OSP	788.4h	1h
	CC	788.4h	1h
	SC	788.4h	1h
Network	Network	10000h	1h

Table 5.4: CTMC parameters for the VM component.

Parameters	**Description**	**Values** (h^{-1})
λ_{ap}	Apache failure rate	1/788.4
λ_{py}	Pydio failure rate	1/336
λ_{os}	OS failure rate	1/2893
μ_{ap}	Apache repair rate	1
μ_{py}	Pydio repair rate	1
μ_{vm}	Instantiation rate for a new VM	1/0.022186

Based on the MTTF and MTTR values of each component (see Table 5.3), we calculated the respective availability for each module (VM, Node, Storage, Network and Controller) of the proposed architecture. The MTTF and MTTR obtained for each component are shown in Table 5.5. The whole system availability was computed through the RBD model that represents the entire adopted architecture (see Figure 4.5) and uses the parameter values from Table 5.5.

Table 5.5: Parameters of the RBD representing the entire system.

Module	*MTTF*	*MTTR*
VM	217.8h	0.94442h
Node	150.31h	0.93537h
Storage	346.92h	0.91069h
Network	10000h	1h
Controller	147.02h	0.97569h

Availability Importance

For the evaluation of the first scenario (see Figure 4.5) we find a value of 98.466% for the availability (A) and 1.81 in nines (9's) (-log[1 - A/100]) of the data storage service hosted in the private cloud without redundancy. This availability corresponds to about 128.58 hours of downtime in a year, and therefore highlights the importance of searching for effective solutions to improve this system. In addition to the first scenario (baseline), which does not consider any module redundancy, other four scenarios were also considered and generated according to the availability importance measures. Table 5.6 presents the summary results showing MTTF, MTTR, availability, uptime and downtime results of the system for each scenario.

Table 5.6: Summary results for all scenarios.

Scenarios	*MTTF*	*MTTR*	*Availability(%) (9's)*	*Uptime*	*Downtime*
1	61.17284h	0.95244h	98.466 (1.81)	8631.42h	128.58h
2	79.14709h	0.70585h	99.116 (2.05)	8688.32h	71.68h
3	104.89277h	0.28499h	99.729 (2.57)	8742.06h	17.94h
4	125.27960h	0.01234h	99.990 (4.00)	8759.13h	0.86h
5	125.39298h	0.01110h	99.991 (4.05)	8759.22h	0.77h

The AI for the baseline scenario was calculated using Equations 2.7, 2.8, 2.9 and the MTTF, MTTR values presented in Table 5.5. Table 5.7 shows the AI results of the baseline architecture. The availability importance measure I_a^i shows that the Controller is the component that has the highest impact on the availability of the whole system. As a result, the second scenario was created adopting the redundancy of that component. Figure 5.2 shows the adopted RBD model that represents this scenario.

Table 5.7: Availability Importance values of baseline scenario.

Component	**Availability Importance**
Controller	0.991203
Node	0.990796
Storage	0.987253
Network	0.984678

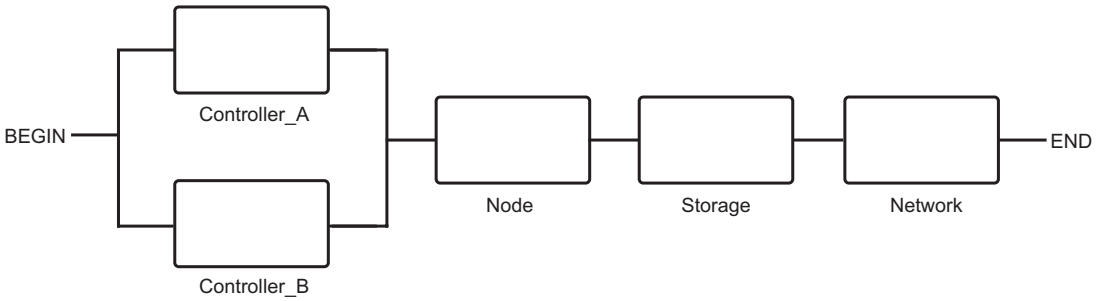


Figure 5.2: RBD model for the Scenario 2.

Table 5.8: Availability Importance values of second scenario.

Component	Availability Importance
Node	0.997328
Storage	0.993762
Network	0.991170
Controller_A	0.006534
Controller_B	0.006534

After adding a redundant *Controller* module (second scenario), we achieved a 44.25% reduction of the downtime in comparison to the baseline scenario (scenario 1), which represents 71.68h of service unavailability. We assumed one year period (or about 8631h service uptime). Still, the unavailability of the system is quite high, and consequently, the availability importance values for the second scenario (presented in Table 5.8) was computed. It shows that *Node* module is the one with the highest impact on the system's availability. And then a third scenario was generated including the redundancy of the *Node* module (see Figure 5.3).

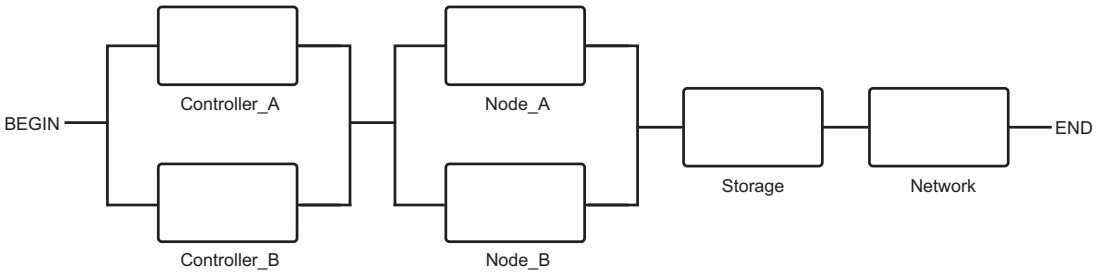


Figure 5.3: RBD model for the Scenario 3.

Table 5.9: Availability Importance value of the third scenario.

Component	Availability Importance
Storage	0.999908
Network	0.997300
Controller_A	0.006575
Controller_B	0.006575
Node_A	0.006167
Node_B	0.006167

The third scenario resulted in a reduction of 86.04% in the downtime, in comparison to the baseline scenario, totalizing 17.94h of service unavailability in the period of one year. Considering a similar approach presented before, the fourth and fifth scenarios were created, so that the storage and network modules are replicated (see Figures 5.4 and 5.5). The availability importance values of the fourth scenario is presented in Table 5.10. The fourth scenario achieved a reduction of 99.33% in the service unavailability in comparison to the baseline scenario, that is, 0.86h of downtime in one year. The obtained results have shown that the system's availability was improved considerably once the redundant components were added to the architecture.

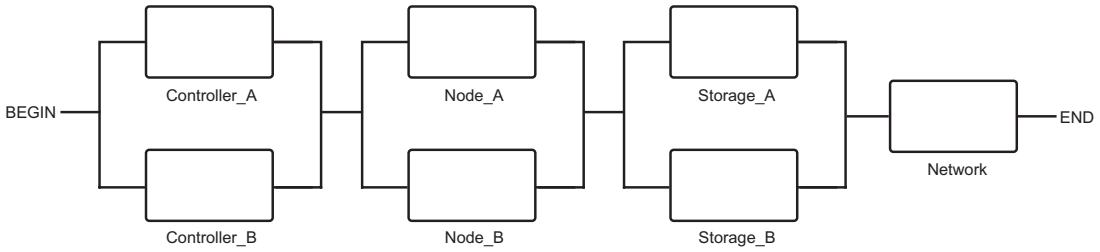


Figure 5.4: RBD model for the Scenario 4.

Table 5.10: Availability Importance value of the fourth scenario.

Component	Availability Importance
Network	0.999998
Controller_A	0.006592
Controller_B	0.006592
Node_A	0.006184
Node_B	0.006184
Storage_A	0.002618
Storage_B	0.002618

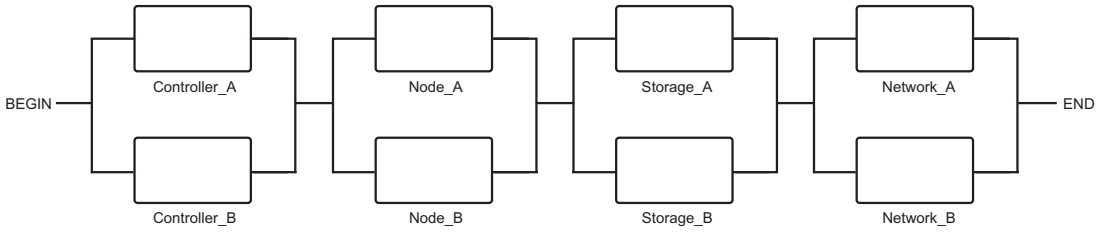


Figure 5.5: RBD model for the Scenario 5.

Despite the fact that the redundancy added in scenario 5 had a considerable impact on the system's availability (reduction of 99.40% in comparison to the baseline scenario), it only represented a decrease of 0.09h in a year if compared to the fourth scenario. In this way, scenario 4 could represent an interesting alternative to be chosen when only availability is taken into consideration. As expected, the scenarios with higher redundancy level achieved better availability results. The availability values obtained were 98.466% in a scenario without replication, and 99.991% for a scenario with all components replicated.

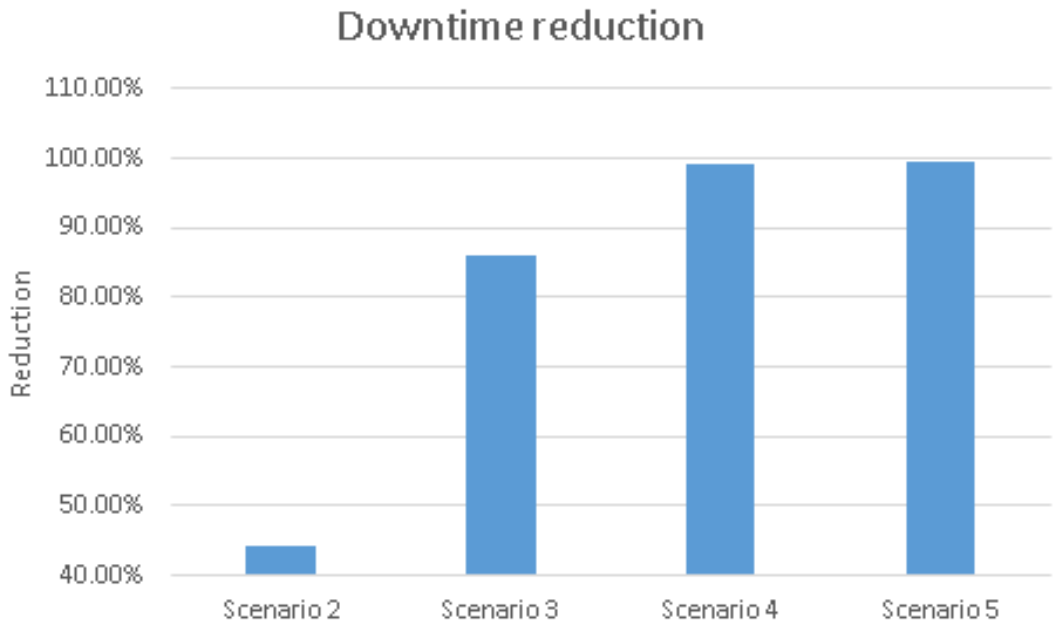


Figure 5.6: Downtime results of each scenario in comparison to the baseline scenario

Figure 5.6 depicts a graphical comparison between the downtime reduction for each scenario in comparison to the baseline scenario. Additionally, Figure 5.7 presents the availability results in terms of nines achieved for each scenario.

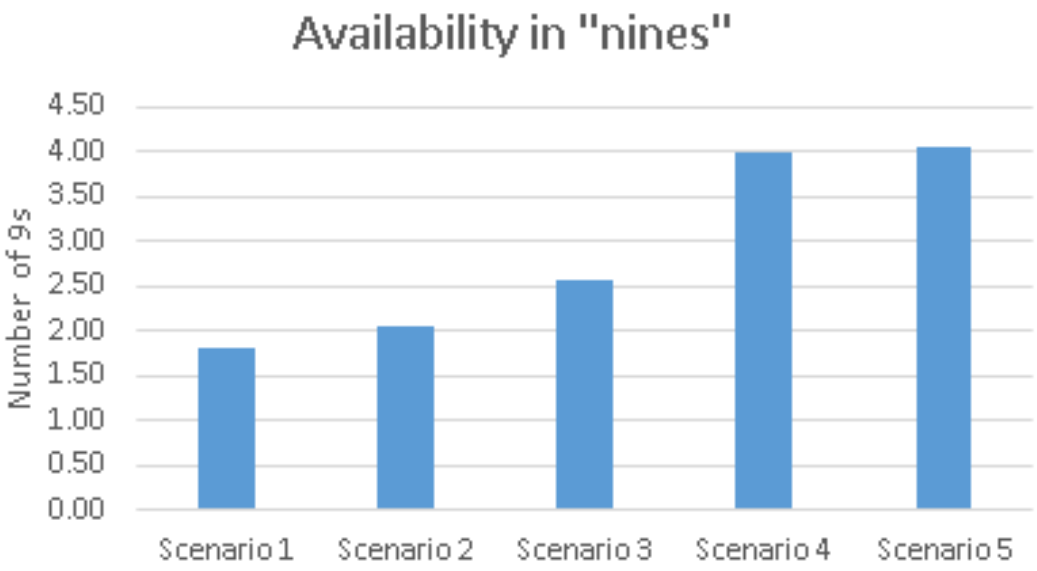


Figure 5.7: Number of 9s for all scenarios.

5.3 Case Study III - Performance Evaluation

We recall Figure 4.6 for performance evaluation, in which simulations were conducted for representing the download of files of sizes 50MB, 100 MB, 300 MB, 700 MB and 1024 MB. The *BufferNetwork* is defined as 1000, and the average time for *TimeOut* and *TimeOutFail* is defined as 60s (ARAUJO et al., 2014). The maximum number of clients supported by VM (*MaxClients*) and the average time to serve a user (value associated to *InService* transition) were measured in the real system that we set up for our experiments using Jmeter tool (HALILI, 2008) and Apache HTTP server benchmarking tool (APACHE, 2016). The *MaxClients* and *InService* values depend on the file size as described in Table 5.1. The *MTTF* and *MTTR* values used for the SPN model are those obtained from the RDB model, which are presented in Table 5.6 and are associated, respectively, with the *ServiceFailure* and *ServiceRepair* transitions.

Results

The main goal of this section is to present the performance evaluation results taking into account scenarios with different availability levels. These results are presented in the Table 5.11. One should note that the following metrics have been considered: Probability of the network buffer to be full (PBF), Probability of timeout occur (POT), Probability the users are not attended due to failures (NUF), System utilization (SU) and System throughput (ST).

The first scenario corresponds to the baseline without redundant components. In the following scenarios, redundancies are added to improve the system availability level. In addition, each scenario has been evaluated considering the download of files with 50, 100, 300, 700 and 1024 MB performed by concurrent users as shown in the MaxClients column of Table 5.1.

Table 5.11: Summary of performance evaluation results.

Scenarios	*File Size (MB)*	*PBF*	*POT*	*NUF*	*SU*	*ST*
1	50	0.01528	0.04160	0.02520	0.00656	7875.23
	100	0.01497	0.04154	0.00995	0.01312	3969.13
	300	0.01373	0.04067	0.00995	0.03940	1311.24
	700	0.01143	0.03911	0.00995	0.09859	567.040
	1024	0.00973	0.03785	0.00995	0.14095	390.000
2	50	0.00882	0.02929	0.02107	0.00661	7927.64
	100	0.00860	0.02924	0.00579	0.01321	3995.54
	300	0.00773	0.02835	0.00579	0.03965	1319.97
	700	0.00618	0.02679	0.00579	0.09919	570.814
	1024	0.00506	0.02554	0.00579	0.14175	392.480
3	50	0.00272	0.01808	0.01718	0.00665	7977.10
	100	0.00258	0.01790	0.00187	0.01329	4020.48
	300	0.00208	0.01650	0.00187	0.03989	1328.22
	700	0.00131	0.01414	0.00187	0.09974	574.388
	1024	0.00087	0.01248	0.00187	0.14251	394.937
4	50	0.00008	0.00850	0.01553	0.00667	7998.81
	100	0.00003	0.00615	0.00019	0.01333	4031.52
	300	0.0	0.00228	0.00019	0.03999	1331.86
	700	0.0	0.00096	0.00019	0.09999	575.935
	1024	0.0	0.00066	0.00019	0.14284	395.992
5	50	0.00007	0.00810	0.01552	0.00667	7998.93
	100	0.00002	0.00572	0.00018	0.01333	4031.57
	300	0.0	0.00205	0.00018	0.03999	1331.87
	700	0.0	0.00087	0.00018	0.09999	575.941
	1024	0.0	0.00059	0.00018	0.14284	395.996

The results show that improving the system availability, reduces slightly the probability of the network buffer being full (PBF). The first scenario, for instance, considers the download of a file with 1024 MB by seven simultaneous users. The probability of the network buffer to be full is 0.97%. For the scenarios 4 and 5 with the same file size, the probability is zero.

Increasing the availability also reduces the probability that a timeout occurs (POT). By adding replicated components, especially the storage device, we observed a reduction of 99.84% when comparing the scenarios 1 and 5 for a file download of 1024 MB. Additionally, a decrease was also observed in the probability of users be not attended due to failures (*NUF*), especially assuming a file of 50 MB. It reduced from 0.0252 to 0.0155 when comparing the scenarios 1 and 5.

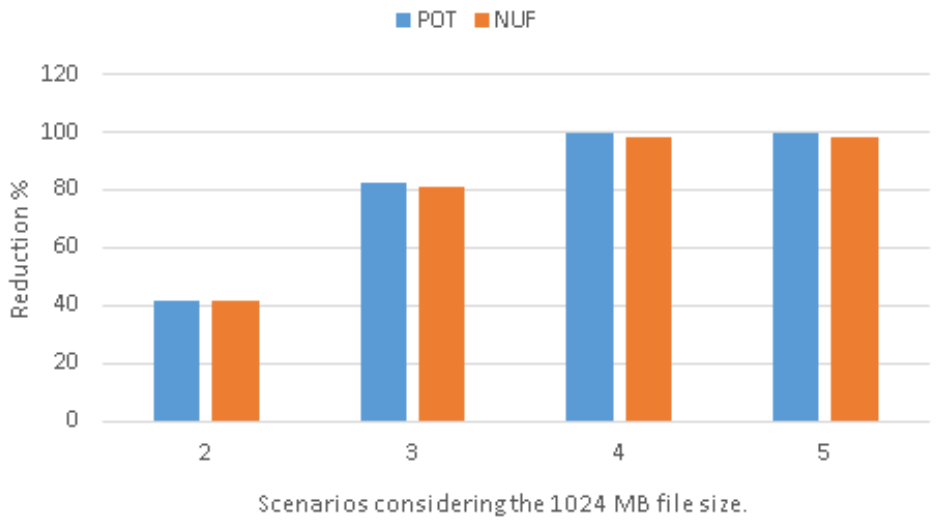


Figure 5.8: POT and NUF results of each scenario in comparison to the baseline scenario.

Figure 5.8 shows a graphical comparison between the probability for a timeout to occur and the number of people not attended due to failures for each scenario, in comparison to the baseline scenario, considering the 1024 MB file size. The most significant reductions were detected in scenarios 4 and 5 in comparison to the scenario without replicated components. Basically, no differences between scenarios 4 and 5 were observed, due to the fact that these scenarios achieved similar availability levels.

Improvements in the availability also enhanced slightly the system utilization (SU) from 14.09% to 14.28% when comparing the scenarios 1 and 5 for a file of 1024 MB. Likewise, the system throughput (ST) is slightly influenced by improvements in the system availability. Comparing the scenarios 1 and 5 with files of 50MB and 1024 MB, we have a difference of only 1.56% and 1.54%, respectively, in the system throughput per hour. Considering the scenario with more replicated components and with the largest files, the maximum system utilization was 14.28% and the throughput was 395.996 files served per hour.

Finally, comparing scenarios 4 and 5 reveal that the impact of the network redundancy is small on the system performance as well as availability. Therefore, such a component should be a minor candidate for redundancy, which may save companies time and money.

5.4 Summary

This chapter presented case studies adopted to illustrate the applicability of the proposed models and the methodology for performing the integrated evaluation of performance and availability of a data storage service hosted in a private cloud infrastructure. Initially, the validation of the proposed SPN model was performed, so that a comparison between the results obtained through the measurements of the real system and the results computed through the proposed model was carried out. Afterwards, the system availability was performed by CTMC and RBD models, taking into account scenarios with different degrees of redundancy generated according to the availability importance measures. Finally, the MTTF and MTTR results from analyzed scenarios were incorporated into the SPN model to compute the performance of the service architecture.

6

Conclusion and Future Works

The increase in the adoption of cloud computing enables many types of services to benefit from this paradigm by reducing deployment costs and on-demand elasticity, for example. More notable examples of these services are the storage of files, bank transactions and streaming of music or videos. However, even in the face of so much evolution and so much investment in this computational model, it is still not possible to guarantee that a cloud system is free of occurrences of failures since computational services demand high availability. Given this, the identification of availability bottlenecks is a good way to mitigate the impacts of system failure where, for example, redundant components can achieve better availability.

Analytical and simulation models are useful for planning computer systems and predicting their behavior even before deployment or during significant changes. Although the creation and analysis of performance and availability models for cloud computing systems is challenging and requires the combination of many techniques for reaching accurate and significant results, this works showed that hierarchical modeling is important and effective for coping with such a hard task. Moreover, it proposed methods for detecting the factors that have the largest importance to the improvement of a cloud storage system availability.

This research achieved a number of results in the areas that it has explored, and the major contributions are the methods and models for identifying performance and availability bottlenecks of private cloud storage systems. The supporting methodology also provides guidance for administrators of storage provider systems that intend to detect points for improvement in their infrastructures. The methods can also be applied in an integrated manner with the process described in this work.

The RBD and CTMC models were proposed to evaluate the service availability in the private cloud considering a non redundant infrastructure and then multiple scenarios using multiple levels of redundancy. The proposed SPN model is used to model service operation and obtain performance metrics from the system, and using the results of RBD and CTMC models as input parameters in the SPN.

To show the applicability of the proposed approach, we presented case studies and conducted experiments to validate the models, were this heterogeneous model are hierarchically

combined to evaluate the impact of the availability on the service performance metrics.

Significant effort was dedicated to reviewing the documentation of cloud computing platforms, and to properly set up test bed infrastructures used in the case studies. The experience acquired through the configuration of every component in such private clouds was essential for a detailed understanding of the systems evaluated here.

6.1 Contributions

The main contributions of this dissertation are the proposed modeling strategy and models for the evaluation of availability and performance metrics of a data storage service hosted in a private cloud. Specific contributions are shown as follows:

- Analytical modeling of a private cloud storage service through the generation of availability and performance models, using a hierarchical and heterogeneous model (e.g., SPN, CTMC, RBD and AI);
- Performance and availability analysis of a private cloud storage service. The availability evaluation of the architecture was performed by the CTMC and RBD models, and then the SPN model was used to compute the performance metrics of the service;
- Identifying infrastructure components that impact on availability of cloud storage services through the generation of scenarios with different redundant components in the cloud infrastructure and according to the availability importance index of each component;
- Obtaining closed-form equations for calculating performance and availability of the private cloud storage service by means of the proposed models.

In addition to the contributions mentioned above, some papers were published based on the results of this research in relevant conferences:

- Elton Bezerra Torres, Gustavo Callou, Gabriel Alves, José Accioly and Hallyson Gustavo, "*Performance and availability evaluation of storage services in private cloud*" in 11th Iberian Conference on Information Systems and Technologies (CISTI 2016), Canary Islands, Spain, 2016.
- Elton Bezerra Torres, Gustavo Callou, Gabriel Alves, José Accioly and Hallyson Gustavo, "*Storage Services in Private Clouds: Analysis, Performance and Availability Modeling.*" in IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC 2016), Budapest, Hungary, 2016.

In addition to the published works, we have the following journal submitted, and the current status is under review:

- Elton Torres, Gustavo Callou, Ermeson Andrade, "*A Hierarchical Approach for Availability and Performance Analysis of Private Cloud Storage Services*" in Springer Computing Journal.

6.2 Future works

Although our proposed modeling strategy and underlying models can be readily extended to other examples and are applicable to private storage services of all sectors and sizes, our case study and experiments are limited to the Eucalyptus cloud computing platform. In order to extend the current work, we intend to:

- To consider more VMs per architecture, and evaluate the impact of availability in each scenario;
- To analyze other dependability metrics for services in the cloud (e.g., reliability, security), in addition to availability;
- To propose an index that makes use of performance and availability parameters;
- To consider other cloud platforms as future work, such as CloudStack, Open Nebula and Open Stack;
- To consider the analysis of integrated metrics, for instance, we can extend this work by analyzing performance, energy consumption, cost, and availability in an integrated way;
- To propose optimization methods (e.g., Greedy Randomized Adaptive Search Procedure - GRASP) to optimize the results estimated through the proposed models;
- To adopt other distribution is another issue that we intend to address in future studies.

References

AMAZON. **Cloud Storage with AWS**. URL: https://https://aws.amazon.com/products/storage/.

AMAZON. **What is Cloud computing**. URL: https://aws.amazon.com/what-is-cloud-computing/.

APACHE. **Apache HTTP server benchmarking tool**. URL: https://httpd.apache.org/docs/2.4/programs/ab.html.

ARAUJO, J. et al. Availability evaluation of digital library cloud services. In: DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2014 44TH ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2014. p.666–671.

ARMBRUST, M. et al. A view of cloud computing. **Communications of the ACM**, [S.l.], v.53, n.4, p.50–58, 2010.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE transactions on dependable and secure computing**, [S.l.], v.1, n.1, p.11–33, 2004.

BARABADY, J.; KUMAR, U. Availability allocation through importance measures. **International journal of quality & reliability management**, [S.l.], v.24, n.6, p.643–657, 2007.

BAUER, E.; ADAMS, R.; EUSTACE, D. **Beyond redundancy**: how geographic redundancy can improve service availability and reliability of computer-based systems. [S.l.]: John Wiley & Sons, 2011.

BELL, K. **Google Drive is back online after brief outage**. [S.l.]: Marsable, 2015. Available on `http://mashable.com/2015/10/09/google-docs-sheets-down/`.

BERRANGé, D. **Virtual Machine Manager**. URL: https://virt-manager.org/.

BLOCH, G. et al. Queueing networks and markov chains. **book, editor John Wiley and sons**, [S.l.], 1998.

BOLCH, G. et al. **Queueing networks and Markov chains**: modeling and performance evaluation with computer science applications. [S.l.]: John Wiley & Sons, 2006.

BUKH, P. N. D.; JAIN, R. **The art of computer systems performance analysis, techniques for experimental design, measurement, simulation and modeling**. [S.l.]: JSTOR, 1992.

CALDER, B. **Windows Azure Storage – 4 Trillion Objects and Counting**. URL: https://azure.microsoft.com/en-us/blog/windows-azure-storage-4-trillion-objects-and-counting/.

CASSERLY, M. **Best cloud storage services 2016 UK**. URL: http://www.pcadvisor.co.uk/test-centre/internet/14-best-cloud-storage-services-2016-uk-3614269/.

CHUOB, S.; POKHAREL, M.; PARK, J. S. Modeling and analysis of cloud computing availability based on eucalyptus platform for e-government data center. In: INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 2011 FIFTH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2011. p.289–296.

CLOUDSTORAGE. **Cloud Computing – Disadvantages**. URL: http://www.top10cloudstorage.com/advice-article/cloud-computing-disadvantages/.

DANTAS, J. et al. An availability model for eucalyptus platform: an analysis of warm-standy replication mechanism. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2012 IEEE INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2012. p.1664–1669.

DE CARLO, F. **Reliability and maintainability in operations management**. [S.l.]: INTECH Open Access Publisher, 2013.

EBELING, C. E. **An introduction to reliability and maintainability engineering**. [S.l.]: Tata McGraw-Hill Education, 2004.

EUCALYPTUS. **HPE Helion Eucalyptus**. URL: http://www8.hp.com/us/en/cloud/helion-eucalyptus-overview.html.

EUCALYPTUS. **Eucalyptus Documentation**. URL: http://docs.hpcloud.com/eucalyptus/4.2.1.

FELLER, W. **An introduction to probability theory and its applications**. [S.l.]: John Wiley & Sons, 2008. v.2.

FIVEASH, K. **AWS outage knocks Amazon, Netflix, Tinder and IMDb in MEGA data collapse**. [S.l.]: The Register, 2015. Avilable on `http://www.theregister.co.uk/2015/09/20/aws-database-outage/`.

FURHT, B. Cloud computing fundamentals. In: **Handbook of cloud computing**. [S.l.]: Springer, 2010. p.3–19.

GERMAN, R. **Performance analysis of communication systems with non-Markovian stochastic Petri nets**. [S.l.]: John Wiley & Sons, Inc., 2000.

GHOSH, R. et al. Scalable analytics for iaas cloud availability. **IEEE Transactions on Cloud Computing**, [S.l.], v.2, n.1, p.57–70, 2014.

GOMES, L.; COSTA, A. Cloud based development framework using IOPT Petri nets for embedded systems teaching. In: IEEE 23RD INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS (ISIE), 2014. **Proceedings...** [S.l.: s.n.], 2014. p.2202–2206.

GOOGLE. **Google App Engine**: platform as a service. URL: https://cloud.google.com/appengine/.

HALILI, E. H. **Apache JMeter**: a practical beginner's guide to automated testing and performance measurement for your websites. [S.l.]: Packt Publishing Ltd, 2008.

HAUNG, K.; MA, Z.; SUN, L. Performance evaluation of node in cloud storage. In: CONSUMER ELECTRONICS, COMMUNICATIONS AND NETWORKS (CECNET), 2012 2ND INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2012. p.2739–2744.

HAVERKORT, B. R. Markovian models for performance and dependability evaluation. In: **Lectures on Formal Methods and PerformanceAnalysis**. [S.l.]: Springer, 2001. p.38–83.

HILDMANN, T.; KAO, O. Deploying and extending on-premise cloud storage based on ownCloud. In: DISTRIBUTED COMPUTING SYSTEMS WORKSHOPS (ICDCSW), 2014 IEEE 34TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2014. p.76–81.

JANPET, J.; WEN, Y.-F. Reliable and available data replication planning for cloud storage. In: ADVANCED INFORMATION NETWORKING AND APPLICATIONS (AINA), 2013 IEEE 27TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2013. p.772–779.

JOHN, L. K.; EECKHOUT, L. **Performance evaluation and benchmarking**. [S.l.]: CRC Press, 2005.

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability modeling and analysis of a virtualized system. In: DEPENDABLE COMPUTING, 2009. PRDC'09. 15TH IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Proceedings...** [S.l.: s.n.], 2009. p.365–371.

KLEINROCK, L. Queueing systems, volume I: theory. , [S.l.], 1975.

KUO, W.; ZHU, X. **Importance measures in reliability, risk, and optimization**: principles and applications. [S.l.]: John Wiley & Sons, 2012.

MACIEL, P. et al. Performance and dependability in service computing: concepts, techniques and research directions, ser. **Premier Reference Source. Igi Global**, [S.l.], 2011.

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

MATOS, R. et al. Sensitivity analysis of server virtualized system availability. **Reliability, IEEE Transactions on**, [S.l.], v.61, n.4, p.994–1006, 2012.

MELL, P.; GRANCE, T. The NIST definition of cloud computing. , [S.l.], 2011.

MURATA, T. Petri nets: properties, analysis and applications. **Proceedings of the IEEE**, [S.l.], v.77, n.4, p.541–580, 1989.

NIELSEN, K. **Top Ten Advantages of Using Online Storage Services**. URL: http://online-storage-service-review.toptenreviews.com/top-ten-advantages-of-using-online-storage-services.html.

PHAM, H. Commentary: steady-state series-system availability. **IEEE Transactions on Reliability**, [S.l.], v.52, n.2, p.146–147, 2003.

RIBAS, M. et al. Assessing cloud computing SaaS adoption for enterprise applications using a Petri net MCDM framework. In: IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2014. **Proceedings...** [S.l.: s.n.], 2014. p.1–6.

SAS, A. **Pydio - Put your data in Orbit**. URL: https://pydio.com/.

SCHOUTEN, E. **Cloud computing defined**: characteristics service levels. URL: https://www.ibm.com/blogs/cloud-computing/2014/01/cloud-computing-defined-characteristics-service-levels/.

SHENG, Y. et al. Research and Application of Private Cloud Storage Platform in High Schools Based on Seafile. In: INTELLIGENT NETWORKS AND INTELLIGENT SYSTEMS (ICINIS), 2013 6TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2013. p.25–28.

SIGMAN, K. The stability of open queueing networks. **Stochastic Processes and their Applications**, [S.l.], v.35, n.1, p.11–25, 1990.

SILVA, B. et al. Mercury: an integrated environment for performance and dependability evaluation of general systems. In: INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORKS CONFERENCE, DSN. **Proceedings...** [S.l.: s.n.], 2015.

SOUSA, E. et al. Evaluating eucalyptus virtual machine instance types: a study considering distinct workload demand. **CLOUD COMPUTING**, [S.l.], p.130–135, 2012.

TRIVEDI, K. S. **Probability & statistics with reliability, queuing and computer science applications**. [S.l.]: John Wiley & Sons, 2008.

TSIDULKO, J. **The 10 Biggest Cloud Outages of 2016**. URL: http://www.crn.com/slide-shows/cloud/300083247/the-10-biggest-cloud-outages-of-2016.htm.

WANG, J.; GONG, W.; XIE, C. A quantitative evaluation model for choosing efficient redundancy strategies over clouds. In: NETWORKING, ARCHITECTURE AND STORAGE (NAS), 2012 IEEE 7TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2012. p.217–226.

WEI, B.; LIN, C.; KONG, X. Dependability modeling and analysis for the virtual data center of cloud computing. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS (HPCC), 2011 IEEE 13TH INTERNATIONAL CONFERENCE ON. **Proceedings...** [S.l.: s.n.], 2011. p.784–789.

XIONG, K.; PERROS, H. Service performance and analysis in cloud computing. In: CONGRESS ON SERVICES-I, 2009. **Proceedings...** [S.l.: s.n.], 2009. p.693–700.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, [S.l.], v.1, n.1, p.7–18, 2010.