Pós-Graduação em Ciência da Computação

A FRAMEWORK FOR AVAILABILITY, PERFORMANCE AND SURVIVABILITY EVALUATION OF DISASTER TOLERANT CLOUD COMPUTING SYSTEMS

By

BRUNO SILVA

Ph.D. Thesis

Federal University of Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2016

BRUNO SILVA

"A FRAMEWORK FOR AVAILABILITY, PERFORMANCE AND SURVIVABILITY EVALUATION OF DISASTER TOLERANT CLOUD COMPUTING SYSTEMS"

A Ph.D. Thesis presented to the Center for Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Advisor: *Paulo Romero Martins Maciel*

RECIFE
2016

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586f Silva, Bruno
A framework for availability, performance and survivability evaluation of disaster tolerant cloud computing systems / Bruno Silva. – 2016.
150 p.: il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2016.

Inclui referências e apêndices.

1. Engenharia de software. 2. Avaliação de desempenho. 3. Redes de Petri. 4. Computação em nuvem.. I. Maciel, Paulo Romero Martins (orientador). II. Título.

005.1 CDD (23. ed.) UFPE- MEI 2016-081

Bruno Silva

A FRAMEWORK FOR AVAILABILITY, PERFORMANCE AND SURVIVABILITY EVALUATION OF DISASTER TOLERANT CLOUD COMPUTING SYSTEMS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutora em Ciência da Computação

Aprovado em: 26/02/2016.

_____

Orientador: Prof. Dr. Paulo Romero Martins Maciel

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Roberto Freire Cunha
Centro de Informática / UFPE

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Dr. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

_____

Prof. Dr. Armin Zimmermann
Systems and Software Engineering / Technische Universität Ilmenau

_____

Prof. Dr. Danielo Gonçalves Gomes
Departamento de Engenharia de Teleinformática / UFC

I dedicate this thesis to all my family, friends and professors
who gave me the necessary support to get here.

Acknowledgements

Initially, I wish to thank God that gave me all the support to get here. Additionally, I thank to my family, professors and colleagues that helped me in this process. I would like thank Suellen, my wife, that understood and supported me the difficulties to write this document and the necessary steps to finish my Ph.D.

I would like to thank Professor Paulo Maciel, who always was present and giving all necessary lessons to raise me the professional I am today. I need to mention Professor Armin Zimmermann, who gratefully received me in Germany during my Ph.D. period. I appreciate the help given by my friends, Eduardo, Jair, Jamilson, Jean, Gustavo, Joao, Danilo, Almir, Vandi, Airton, Ermeson, Julian and many others that were present in the whole process.

Thank you all.

I have yet to see any problem, however complicated, which, when looked at in the right way, did not become still more complicated.

—PAUL ANDERSON

Resumo

Sistemas de Computação em Nuvem (SCNs) permitem a utilização de aplicações como serviços para usuários em todo o mundo. Um importante desafio para provedores de SCN corresponde ao fornecimento de serviços de qualidade mesmo na presença de eventuais falhas, sobrecargas e desastres. Geralmente, um acordo de nível de serviço (ANS) é estabelecido entre fornecedores e clientes para definição dos requisitos de disponibilidade, desempenho e segurança de tais serviços. Caso os parâmetros de qualidade definidos no ANS não sejam satisfeitos, multas podem ser aplicadas aos provedores. Nesse contexto, uma estratégia para aumentar a disponibilidade de SCNs e mitigar os efeitos de eventuais desastres consiste em utilizar subsistemas redundantes e adotar de centros de dados distribuídos geograficamente. Considerando-se esta abordagem, os serviços de centros de dados afetados podem ser transferidos para outros centros de dados do mesmo SCN. Contudo, o tempo de sincronização entre os diferentes centros de dados aumenta com a distância entre os mesmos, o que pode afetar a performance do sistema. Além disso, o provisionamento excessivo de recursos pode afetar a rentabilidade do serviço, dado o alto custo dos subsistemas redundantes. Portanto, uma avaliação que contemple desempenho, disponibilidade, possibilidade de desastres e alocação de centro de dados é de fundamental importância para o projeto de SCNs.

Este trabalho apresenta um *framework* para avaliação de SCNs distribuídos geograficamente que permite a estimativa de métricas de desempenho, disponibilidade e capacidade de recuperação de desastres (naturais ou causados pelo homem). O *framework* é composto de um processo de avaliação, conjunto de modelos, ferramenta de avaliação e ferramenta de injeção de falhas. O processo de avaliação apresentado pode auxiliar projetistas de SCNs desde a representação do sistem de computação em nuvem até a obtenção das métricas de interesse. Este processo utiliza uma modelagem formal híbrida, que contempla modelos de SCN de alto nível, redes de Petri estocásticas (RPEs) e diagramas de bloco de confiabilidade (DBCs) para representação e avaliação de SCNs e seus subsistemas. Uma ferramenta de avaliação é proposta (GeoClouds Modcs) que permite fácil representação e avaliação de sistemas de computação em nuvem. Por fim, uma ferramenta de injeção de falhas em SCN (Eucabomber 2.0) é apresentada para estimar métricas de disponibilidade e validar os modelos propostos. Vários estudos de caso são apresentados e estes analisam a capacidade de recuperação de desastres, desempenho e disponibilidade de SCNs distribuídos geograficamente.

Palavras-chave: Avaliação de performabilidade. Capacidade de recuperação de desastres. Computação em nuvem. Redes de Petri estocásticas. Diagramas de blocos de confiabilidade.

Abstract

Cloud Computing Systems (CCSs) allow the utilization of application services for users around the world. An important challenge for CCS providers is to supply a high-quality service even when there are failures, overloads, and disasters. A Service Level Agreement (SLA) is often established between providers and clients to define the availability, performance and security requirements of such services. Fines may be imposed on providers if SLA's quality parameters are not met. A widely adopted strategy to increase CCS availability and mitigate the effects of disasters corresponds to the utilization of redundant subsystems and the adoption of geographically distributed data centers. Considering this approach, services of affected data centers can be transferred to operational data centers of the same CCS. However, the data center synchronization time increases with the distance, which may affect system performance. Additionally, resources over-provisioning may affect the service profitability, given the high costs of redundant subsystems. Therefore, an assessment that includes performance, availability, possibility of disasters and data center allocation is of utmost importance for CCS projects.

This work presents a framework for geographically distributed CCS evaluation that estimates metrics related to performance, availability and disaster recovery (man-made or natural disasters). The proposed framework is composed of an evaluation process, a set of models, evaluation tool, and fault injection tool. The evaluation process helps designers to represent CCS systems and obtain the desired metrics. This process adopts a formal hybrid modeling, which contemplates CCS high-level models, stochastic Petri nets (SPN) and reliability block diagrams (RBD) for representing and evaluating CCS subsystems. An evaluation tool is proposed (GeoClouds Modcs) to allow easy representation and evaluation of cloud computing systems. Finally, a fault injection tool for CCSs (Eucabomber 2.0) is presented to estimate availability metrics and validate the proposed models. Several case studies are presented and analyze survivability, performance and availability metrics considering multiple data center allocation scenarios for CCS systems.

Keywords: Performability evaluation. Disaster recovery. Cloud computing. Stochastic Petri nets. Reliability block diagrams.

List of Figures

List of Tables

List of Acronyms

Contents

1

Introduction

Both academia and industry have proposed various definitions for cloud computing. A remarkable definition comes from National Institute of Standards and Technology (1). According to it, cloud computing is a paradigm to allow on-demand network access to a shared set of computing resources that can be provided in a fast and easy way. That resources are released with minimal user effort or service provider interaction. Cloud computing has driven the new wave of Internet-based applications by providing computing as a service (2). The concepts of cloud computing were inspired by other technologies, such as grid computing, utility computing, and virtualization (3). By opting for cloud computing, customers do not need to invest heavily in hardware/software platforms to create and maintain an IT infrastructure. Instead, they use cloud services and pay according to the resource utilization (4). Thus, cloud computing platforms represent a viable solution to scalability issues for business in general. Nowadays, common business applications (e.g., spreadsheets) are provided as cloud computing services, in the sense that they are often accessed using a Web browser, and their respective software/data reside on remote servers. This approach has affected all fields of the computational research, from users to hardware manufacturers (5). Such a paradigm is attractive for many reasons and presents the following characteristics (6):

- Shared resources such as CPU, storage or software can be utilized instantaneously without human interaction. It frees users from installing, configuring and updating software applications. Heterogeneous clients platforms like laptops, smartphones and tablets can be adopted to access cloud resources via network access (e.g. Internet).

- Cloud providers utilize a multi-tenancy model, in which computing resources are grouped in a virtualization pool. According to client needs, the pool allocates and reallocates virtual and physical resources to attend requests. As a result, clients have no control where the requested resources located or which machine execute his/her programs.

- For users, resources provisioning is virtually not limited, in the sense that request peaks can be rapidly provisioned. Computing resources can be used as much as it is required and released once the request peak scales down. Although computing

resources are provided in pooled and shared way, users can measure the resource usage and control its utilization according to their needs.

In addition to these characteristics, it is possible to classify cloud computing based on the service model. The most prominent service models are Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). SaaS represents a service model in which software is released on a host environment, which can be accessed simultaneously through several clients via a network connection. This service model is also known as "on-demand software". Important SaaS applications include SalesForce (7), Google App Services (8), and Microsoft Office online (9). Platform as a Service is a software development platform entirely hosted on the cloud. This service model supports the entire software development life-cycle. In this sense, software developers are able to implement cloud services and applications directly on the cloud. PaaS platforms for software development include Google App Engine (10), AWS Elastic Beanstalk (11), Mendix (12), and Heroku (13). Last but not least, Infrastructure-as-a-Service (IaaS) delivers, on-demand, computing resources in the form of Virtual Machines (VMs) running on the cloud provider's data center, satisfying user needs. User requests are provisioned depending on the data center capacity concerning physical machines. According to (14), the IaaS cloud market will see a growth of 31.7% in a five-year period. Important IaaS infrastructures include Amazon EC2 (15) and IBM Smart Business Cloud (Softlayer) (16).

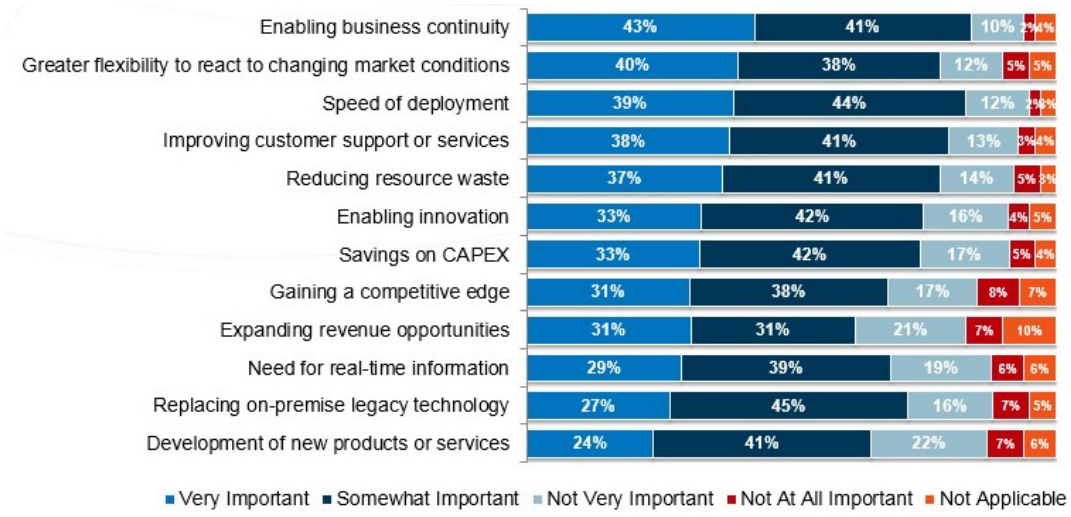
1.1 Motivation and Justification

Cloud system's performance, availability, and survivability represent key aspects for IaaS providers and their customers (17). These metrics consider the effects of system load and failure/recovery behavior of data center subsystems. To define the provider's Quality-of-Service (QoS), an agreement is negotiated between cloud providers and their customers (18). This agreement is often presented as a business warranty contract, such as Service Level Agreements (SLAs), which specifies a list of rules and requirements for cloud services (e.g., maximum annual downtime) (17). The enterprises' reputation can be affected and penalties may be applied if the defined quality level is not satisfied. Thus, to meet SLA requirements, IaaS providers need to evaluate their environment, considering, also, the possibility of disasters.

There were several natural disasters in the last decade. Among them, we can cite the great floods in Thailand in 2011, Hurricane Sandy in the United States (US) in 2012, and Typhoon Haiyan in the Philippines in 2013 (20). In the last decades, the frequency of natural disasters recorded in the US Emergency Events Database (EM-DAT) has increased almost three-fold, from over 1,300 events in 1975–1984 to over 3,900 in 2005–2014 (20). Therefore, cloud computing designers must consider eventual disasters to plan these infrastructures.

According to (19), the most important business driver investment in cloud computing technology is associated with business continuity (Figure 1.1). The report shows that 84% of interviewed cloud managers believe that enabling business continuity is a very or somewhat

Figure 1.1: Importance of business drivers to costumers.

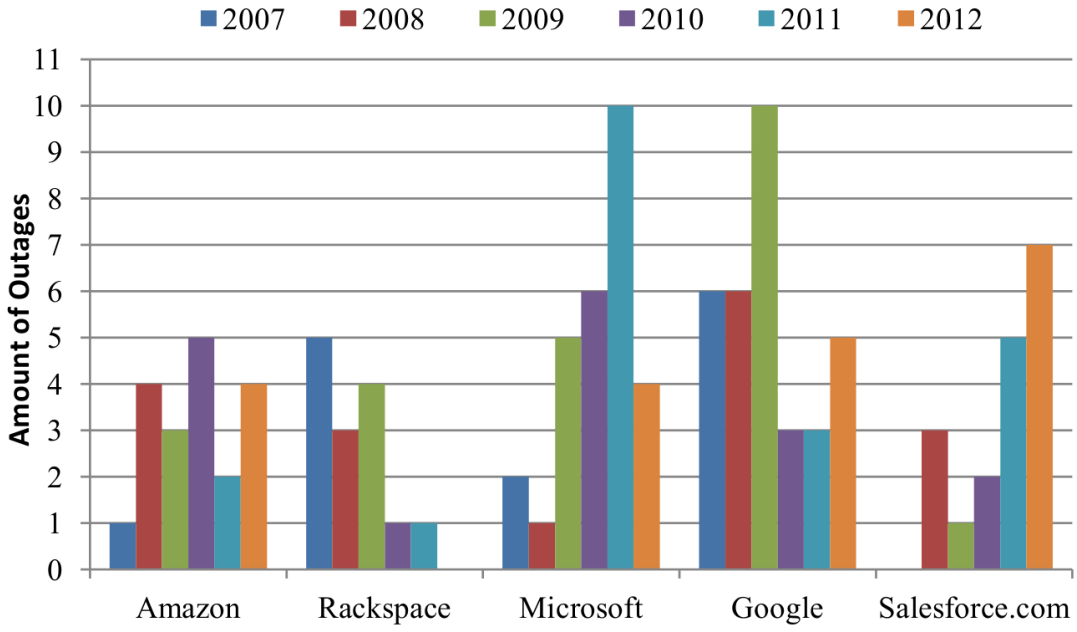


Source: (19).

important investment to cloud environments. For cloud computing providers, even a few minutes of service interruption may cause important cost impact. For instance, in August 2013, a five-minute downtime in Google services led to an estimated revenue loss of over US-$ 500,000 and an Internet traffic decrease of 40% (21).

Figure 1.2 as pointed out in (22) shows the cloud computing outages during the 2007-2012 period for the major cloud computing providers. It is possible to see that almost all providers had outages every year, except Rackspace in 2012 and Salesforce in 2008. Regarding the cloud service outage duration, the same study (22) shows that 43% of outages lasted from one to six hours, and just 14% of outages took more than 24 hours (not shown in figure).

Figure 1.2: Cloud computing outages over providers and years



Source: (22).

Table 1.1: Main Disastrous Events (adapted from (24))

Avalanche	Flood	Shooting
Severe Weather	Natural Gas Leak	Fuel Shortage (associated with a loss of main electrical power)
Biological Hazard	Temperature Control Failure	Bomb Threat
Civil Disorder	Hostage Situation	Kidnapping
Telecom Outage	Acts of Terrorism	Theft
Robbery	Train Crash or Derailment	Lightning Strike
Computer/Software Failure or Virus	Employee/Union strike	Acts of Vandalism
Pandemic	Hacker Attack	Power Outage
Fire Damage	Water Damage	Radiological Hazard

The main purpose of a business continuity (or disaster recovery) plan is to mitigate the impact of interruption events that may affect business process (23). This plan contains activities or processes for avoiding, preparing and recovering the system in case unforeseen events that may affect the service operation. These events range from natural disasters (such as floods or avalanches) to human-made attacks (e.g., terrorism, hacker attacks). Table 1.1 presents a list of the main potentially disastrous events (24) for information technology companies.

A disaster recovery plan requires the utilization of different data centers located far enough apart to mitigate the effects of unforeseen disasters (25). Considering these systems, backup services can be adopted to receive copies of VM data during data center operation. Hence, whenever a disaster makes one data center unavailable, affected VMs can be re-instantiated in another operational data center. Unfortunately, some data between the last VM backup and the disaster may be lost and it is necessary some time to restart the operation after a failure. However, this may be traded off by changing the time between backups and the distance between data centers. If multiple data centers are located in different geographical locations (considering disaster independent places), the availability level of whole system should improve. On the other hand, VM synchronization time increases due to distance between data centers. Additionally, failures and overloads can lead to system downtime considering cloud computing systems. Consequently, a cloud system evaluation considering VM data backup time and different user loads levels is of utmost importance when considering the analysis of distributed cloud systems. Modeling techniques, with a strong mathematical foundation, such as Stochastic Petri Nets (SPNs) (26) can be adopted to evaluate dependability, performance and survivability in complex infrastructures.

1.2 Objectives

Considering the previously stated issues, this work proposes a framework to evaluate performance, dependability and survivability metrics in IaaS systems deployed into geographically distributed data centers. More specifically, the objectives of this research are:

- To construct a high level model that represents geographically distributed IaaS cloud architectures. This model represents cloud systems structure in terms of number of data center, hardware and software components as well as transmission, dependability, performance and disaster parameters. By using this model, designers may have a complete and unified description of system architecture and the relationship between its components. It serves as a base for the creation of the final evaluation model.
- To propose a set of stochastic Petri net models for estimating survivability, dependability and performance metrics of disaster tolerant IaaS cloud infrastructures. The presented models capture important aspects of distributed cloud systems, such as: (i) VM data transfer, (ii) VM data replication time, (iii) disaster occurrences.
- To develop a tool (GeoClouds Modcs) which automatically creates and evaluates the aforementioned models and allows IaaS designers to conduct a joint evaluation of survivability, dependability and performance. The proposed tool presents a simple graphical interface that permits users to create the evaluation models in a user-friendly way.
- To propose algorithms that translate high level IaaS models into SPN dependability, performance and survivability models. The proposed algorithms are implemented in GeoClouds Modcs tool and users perform the translation process in a transparent way.
- To propose an evaluation process that allows IaaS designers to assess IaaS cloud systems. The system is evaluated in parts and the evaluation results are combined to create the final results.
- To develop a validation tool, namely Eucabomber 2.0, which provides fault injection to cloud system components (software and hardware) and estimates dependability related metrics. The estimated results are adopted to validate the SPN evaluation models.

1.2.1 Expected Contributions

The main contribution of the proposed work is to provide a framework (including tools, models, evaluation and validation process) for assessing performance, dependability and survivability metrics for disaster tolerant cloud computing systems. The estimated metrics

include (but are not limited to) availability, probability to complete a request, server utilization, probability to recover the service, and probability to finish a backup in a given time. Additionally, an evaluation process is also proposed for helping designers to obtain the desired metrics. This process adopts a hybrid formal modeling, which contemplates RBD/SPN and a tool to perform automatic generation and evaluation of the proposed models.

By providing the expected results, the proposed work will have valuable importance for cloud computing companies and academic community. Regarding the academic community, the results of this research have been published in academic journals and conferences. Taking into account the importance of this work to cloud computing research area, academic works (e.g., master's thesis) have been extending this research. Considering cloud computing companies, we expect that the proposed framework will be adopted to improve their systems QoS and reduce the impact of disasters.

1.3 Outline

The document is organized as follows. Initially, Chapter 2 shows the related works and presents the contribution of the proposed thesis compared to the current state of the art. Chapter 3 presents some background concepts related to this research. Chapter 4 depicts the framework models. Chapter 5 describes the formal composition rules adopted in the proposed modeling approach. Chapter 6 shows GeoClouds Modcs and Eucabomber 2.0 tools' overview and infrastructure. Chapter 7 presents the case studies. Finally, Chapter 8 concludes this work and introduces future works.

2

Related Works

In the last few years, some research has been done on dependability, performance and survivability assessment of cloud computing systems and a subset of them has also considered the impact of failures, workloads and disaster as well as some model validation techniques. The following sections present some of those research efforts.

2.1 Model-Based Approaches for Cloud System Evaluation

Over the last years, some authors have been devoting efforts to study dependability issues on cloud computing systems. Longo et al. (27) proposed an approach to quantify the availability of cloud computing systems based on Petri nets and Markov chains. The authors adopt interacting Markov chain models and closed-form equations to evaluate large systems. Their results show that the solution time reduces when compared to a single monolithic model. Notwithstanding the paper's contributions, the authors do not consider the possibility of disasters neither the utilization of distributed clouds. In (28), a performability analysis for cloud systems is presented. The authors quantify the effects of variations in workload, failure rate and system capacity on service quality.

The paper presented in (29) proposes a stochastic model (based on stochastic reward net) to evaluate Quality of Service (QoS) metrics of IaaS cloud systems. The proposed work assess system availability, utilization, and responsiveness. A resiliency analysis is provided to take into account periodic and burst loads. The proposed stochastic model does not represent the occurrence of disasters and component failures (e.g., server failure) which may have a great impact on system behavior.

Bradford et al (30) describe a system design approach for supporting transparent migration of VMs adopting local storage for their persistent state. The approach is transparent to the migrated VM, and it does not interrupt open network connections during VM migration. In (31), the authors present a case study that quantifies the effect of VM live migrations in the performance of an Internet application. Such study helps data center designers to plan environments in which SLAs determine a desired level for the specified metrics, such as service availability and responsiveness.

Dantas et al. (32) present a study on warm-standby mechanisms in Eucalyptus-based private clouds. Their results demonstrate that replacing machines by more reliable counterparts would not produce significant improvements in system availability, whereas some techniques of fault-tolerance can indeed increase dependability levels. The work presented in (33) describes an experiment-based approach for studying the dependability of a disaster recovery solution supported by a private cloud. The authors employed fault injection and obtained dependability measures, in an approach that can be used to cross-check analytical and simulation models, as well as to give foundations for the definition of service level agreements with customers.

In (34), the authors present an sensitivity analysis for a variant of Hadoop Distributed File System (HDFS), which contemplates energy saving techniques. The proposed system divides the cluster data in Hot and Cold Zones. In this approach, data that present long periods (i.e., several days) of idleness are allocated in the Cold Zone. That analysis also shows the energy-saving behavior considering the variation of file system parameters.

The work presented in (35) adopts a two-level hierarchical modeling approach for virtualized systems which uses fault trees in the upper level, and CTMC in the lower level. The support for sensitivity analysis in these analytical models is important for detecting bottlenecks in system availability.

In (36), the authors show four different sensitivity analysis techniques to determine the parameters that cause the greatest impact on the availability of a mobile cloud system. The authors use a combined evaluation of results from different analysis techniques to deal with the evaluation of the system. Their results show that the availability can be effectively improved by changing a reduced set of parameters.

An approach for dependability modeling of cloud environments is also found in (37). The authors propose a model called CDSV (Cloud Dependability by using System-level Virtualization), which uses a combinatorial technique for computing metrics such as availability and reliability. Their approach enables computing the maximum number of VMs that can be hosted on a physical node while keeping a certain desired dependability level. Although, the combinatorial modeling imposes many restrictions for representing in detail mechanisms such as the live migration of VMs, that may affect the accuracy of results achieved through the CDSV model.

CloudSim Toolkit (38) quantifies resource allocations policies and scheduling algorithms in cloud computing environments. The environment considers different service models as well as evaluates energy and performance related metrics based on cloud characteristics. However, the proposed toolkit does not takes into account dependability assessment or disaster occurrences. In (39), the authors adopted model checking algorithms to decide if a given system is survivable or not. By using this approach, the system behavior right after the disaster occurrence is evaluated. CSL logic and continuous time Markov chain models are adopted to represent and estimate survivability metrics in Given-Occurrence-of-Disaster (GOOD) models.

Cloth et al. (40) propose stochastic reward nets (SRNs) for availability evaluation of

cloud architectures by using consumers perspective. A calibration phase is proposed by authors to improve model results accuracy. Important aspects such as disaster recovery issues and availability zones are considered in this work. Despite the work's remarkable quality, the authors do not consider high level tools to create the proposed models.

The work proposed by (41), introduces an approach for mitigating the impact of failures in distributed systems due to provider's outages by using cloud computing technology. The authors suggest a so-called Cloud Standby as a new method for disaster recovery of distributed systems in the Cloud. The work adopts a disaster recovery process for monitoring a standby site. Whenever a disaster occurs, the recovery process activates the standby system and initiates the emergency operation. Although this work presents a relevant contribution to the scientific and industrial communities, this work does not evaluate the impact of the proposed approach adoption regarding availability issues.

2.2 Experiment-Based Techniques for Cloud Evaluation

Fault injection tools have been largely adopted for the evaluation of dependable systems. Zhang et al. (42), presents a fault injector tool for real time embedded systems called Debug-based Dynamic Software Fault Injection System (DDSFIS). The authors validated the effectiveness and performance of DDSFIS adopting real world experiments.

The work proposed in (43) presents a Cloud Dependability Analysis (CDA) framework that adopts fault injection techniques to evaluate cloud computing dependability. The authors designed different failure metrics to model and quantify the correlation of performance metrics with failure events in virtualized and non-virtualized systems. It is important to stress that the data collecting task can be time consuming. Additionally, the tool just can evaluate infrastructures that are already deployed. Hence, it is not possible to perform system evaluation at project design time.

In (44), the authors present a benchmark for performance tests in databases for cloud-based systems. The authors defined a set of benchmarks and report results for four widely used databases (e.g., MySQL). Although the proposed benchmark also intend to be a tool for evaluating availability in databases for cloud systems, only performance and elasticity aspects are addressed in the work.

Souza et al. (45) initiate the study of fault injection into Eucalyptus cloud with Eucabomber (version 1.0). Despite the important contribution, this version does not consider dependencies between components and user-oriented metrics are not taken into account.

In (46), the authors investigate software aging effects on Eucalyptus framework (47), and they also propose a strategy to mitigate such issues during system execution. (48) describes a software testing environment, using cloud computing technology and virtual machines with fault injection facility. The software injects failures in cloud environments in order to evaluate its dependability metrics by using measurements activities.

2.3 Final Remarks

Table 2.1 summarizes the main characteristics of this thesis and compare them to previous works. The aspects explored in this thesis are the following: (i) adoption of analytical/simulation models, (ii) evaluation tool proposition, (iii) disaster recovery issues, and (iv) fault injection aspects. The first column (analytical/simulation models) shows that some works deal with dependability, performance, performability, energy, and survivability. The works indicated with *hierachical heterogeneous* present combined modeling techniques that adopt more than one formalism (e.g., RBD and SPN). Most works are interested in evaluating some quality aspect of the cloud system. Just a few works provide a tool to make the system evaluation process easier. Most of evaluation works do not consider the occurrence of disaster for cloud systems. Additionally, some works adopt fault injection techniques to evaluate the system under real-world conditions.

Table 2.2 presents a more detailed view of the most related previous works. It is possible to observe that all of them deals with VM transmission, and some are related to availability, performance, and survivability evaluation. Among all of the previous work, (40) is the most similar. The main differences between these works are the following. The work presented in (40) does not consider performance issues in the evaluation. Another important difference is related to the cloud's utilization level. Whereas this thesis considers failures/repairs at infrastructure and application levels, (40) works only with application level's failures and repairs. (40) also does not consider the impact of data transmission latency due to the distance between data centers.

Different from the previous, this work proposes a framework for evaluating cloud computing systems deployed into geographically distributed data centers, concerning VM migration, disasters occurrence and different user loads. Moreover, performability metrics taking into account user and provider perspectives are adopted. A set of stochastic models is proposed to support the evaluation. The models are divided into blocks that can be combined to create larger models. A software tool (GeoClouds Modcs) is introduced to allow designers to evaluate geographically distributed clouds systems even if the designer does not have experience with stochastic models. To the best of our knowledge, no other software contemplates all the before-mentioned characteristics in a single platform. Additionally, a validation method for cloud computing system is proposed which considers VM life-cycle operations and dependency between cloud components. The availability results contemplate user perspective, in which the service is available as long as the contracted VMs are operational. Model results are validated by adopting Eucabomber 2.0 in a real world environment.

Table 2.1: Related Works

	Analytical / Simulation Models	Evaluation Tool	Disaster Recovery	Fault Injection
Longo et al. 2011 (27)	Dependability	no	no	no
Gosh et al. 2010 (28)	Hierachical Heterogeneous	no	no	no
Bruneo et al. 2014 (29)	Performability	no	no	no
Bradford et al. 2007 (30)	no	no	yes	no
Voorsluys et al. 2009 (31)	Performance	no	yes	no
Dantas et al.2012 (32)	Dependability	no	no	no
Mattos et al. 2014 (1) (33)	Dependability	no	no	yes
Kaushik et al. 2010 (34)	Energy Model	no	no	no
Kim et al. 2009 (35)	Hierachical Heterogeneous	no	no	no
Matos et al. 2014 (2) (36)	Hierachical Heterogeneous	no	no	no
Sun et al. 2010 (37)	Dependability	no	no	no
Buyya et al, 2009 (38)	Performance and Energy Models	yes	no	no
Cloth, 2005 (39)	Survivability Models	no	yes	yes
Xu et al, 2013 (40)	Availability	no	yes	yes
Lenk et al, (41)	no	no	yes	no
Zhang, 2011 (42)	no	no	no	yes
Guan et al. 2012 (43)	no	yes	no	yes
Cooper, 2010 (44)	no	no	no	yes
Souza, 2013 (45)	no	no	no	yes
Araujo, 2011 (46)	no	no	no	yes
Banzai, 2010 (48)	no	no	no	
This Thesis	Hierachical Heterogeneous	yes	yes	yes

Table 2.2: Even more similar works

	VM Transmission	Availabilty Evaluation	Performance Evaluation	Survivability Evaluation
Bradford et al. 2007 (30)	yes	no	no	no
Voorsluys et al. 2009 (31)	yes	no	yes	no
Cloth et al, 2005 (39)	yes	no	no	yes
Lenk et al, 2014 (41)	yes	no	yes	no
Xu et al, 2013 (40)	yes	yes	no	yes
This Thesis	yes	yes	yes	yes

3

Background

This chapter shows a summary of the background information needed for a better understanding of this work. First of all, an overview of disaster tolerant cloud computing infrastructures is presented. After that, dependability, performance and survivability concepts are shown. Next, the adopted formalism such as SPN and RBD are explained. Finally, fault injection and availability monitoring concepts are discussed.

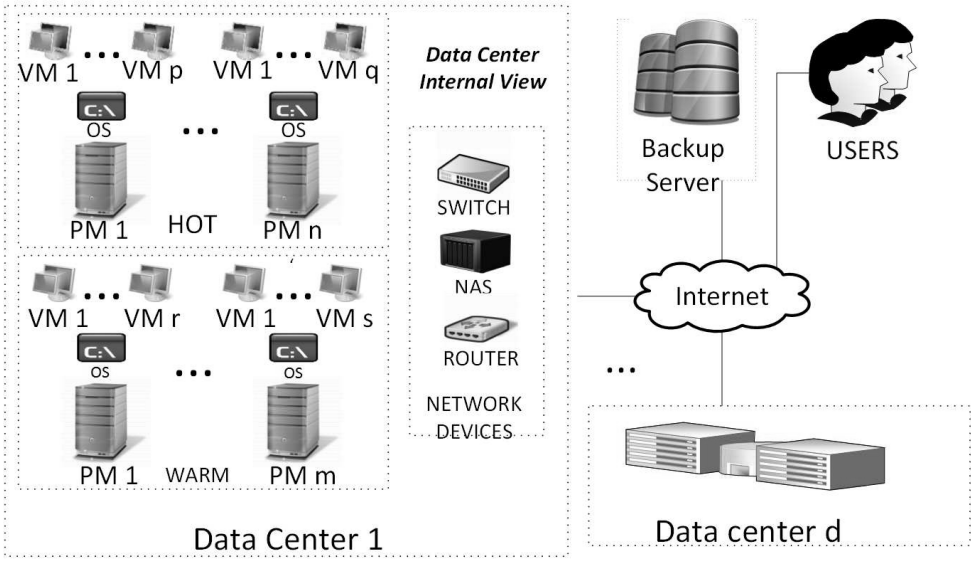
3.1 System Architecture of Reliable Distributed Data Centers

This section presents an overview of the cloud computing system considered in this work, which contemplates a set of components, distributed over distinct data centers (Figure 3.1). The Infrastructure-as-a-Service model is adopted, considering delivery of computing resources on-demand as virtual machines. The proposed system is composed of a set of facilities that can be a data center or a Backup Server (BS). There are d data centers to serve client requests. In this work, a single Backup Server is responsible for providing backup of VM data. Data centers are composed of physical machines and network components. We consider a limited number of users requests during system operation. Each user request should result in a new instantiated VM.

BS periodically receives a copy of each VM image during data center operation. Hence, whenever a disaster makes one data center unavailable, BS sends VM copies to an operational data center. We assume the BS as an external service and its internal components are not known. The only perceived behavior of a BS is related to its operational state (failed/working). The system is composed of d data centers. Each data center contains a two machine sets (hot and warm pools). The hot pool is composed of n Physical Machines (PMs), which are active and running VMs. The warm pool consists of m PMs that are active, but without running VMs. A disaster can happen to a facility at any time during system operation. After that, we assume that the recovery operation starts immediately after the disaster occurrence.

Depending on the capacity of physical machines, multiple VMs can run in the same host. In this study, we assume that each machine is able to run l virtual machines. PMs may share a common Network Attached Storage (NAS) or a Storage Area Network (SAN) to provide

Figure 3.1: Distributed Cloud System Example



Source: Made by author.

distributed storage and to allow the migration of a virtual machine from one server to another in the same data center (49). In the case of failure, a VM must be instantiated in another physical machine of the same data center. If there is no available PM in the current data center, the VM image is moved to another data center.

The following list summarizes the assumptions taken to represent disaster tolerant IaaS systems.

- All physical machines identical as well as the VMs have the same characteristics.
- A VM can be started on any PM machine if it is not broken.
- The number of requests that the system can attend at a given time is limited. If the system is overloaded, new requests are discarded.
- All users requests are identical. For each user request, a VM should be instantiated. Therefore, each client requests corresponds to a VM instantiation request.
- A facility can always be repaired after a disaster.

3.2 Performance and Dependability Evaluation

Generally, for evaluating computing systems, performance, and dependability metrics have been adopted considering different perspectives. Performance evaluation takes into account "how well the systems performs" and dependability assessment considers "probabilities of a system executing successfully". However, in degradable systems (i.e., the system does not necessarily interrupt the service in case of faults) performance and dependability issues must

be evaluated simultaneously to estimate system effectiveness (50). Performance evaluation refers to a set of techniques and methods which permits the assessment of temporal system behavior. More specifically, performance evaluation involves the system assessment under a workload (51). Performance metrics (e.g., throughput and response time) can be evaluated by adopting measurement approaches and modeling techniques. The most suitable models to evaluate performance metrics are: Temporal Logics, Networks of Queues and Markov Chain based models (e.g., SPN) (26).

The dependability of a system can be understood as the ability to deliver a set of services that can be justifiably trusted (52). Indeed, dependability is also related to disciplines, such as security, survivability, reliability and availability. For instance, the reliability of a system at time t' is the probability of such a system have not failed from $t = 0$ until $t = t'$, whereas its steady state availability is the probability of such a system being operational. Dependability metrics can be calculated either by combinatorial models, such as RBD, or state-based models (e.g., SPN). RBDs are networks of functional blocks connected according to the effect of each block failure (and repair) on the system reliability (and availability) (26). Combinatorial models capture conditions that make a system fail (or to be working) in terms of structural relationships between the system components. It is assumed that the failure or recovery of a component is not affected by the behavior of any other component.

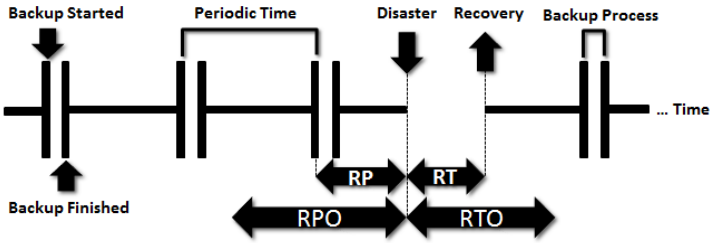
State-based models (53) represent a more suitable choice to model complex interactions between components such as dynamic redundant mechanisms and maintenance policies. These models represent the system behavior (failures and repair activities) by its states and the event occurrence are expressed as labelled state transitions. Labels can be probabilities, rates or distribution functions. Some state-based models may also be evaluated by discrete event simulation in case of intractable large state spaces or when combination of non-exponential distributions prohibits an analytic solution. In some special cases (e.g., dynamic redundant mechanisms), state-based analytic models cannot be solved by closed-form equations. The adoption of a numerical solution is required to solve those cases. The most prominent combinatorial model types are Reliability Block Diagrams, Fault Trees and Reliability Graphs. Markov Chains, Stochastic Petri nets, and Stochastic Process algebras are most widely used state-based models. The reader should refer to (26) for more details about dependability and performance concepts and metrics.

3.3 Survivability

Survivability can be defined as the ability of a system to recover a predefined service level in a timely manner after the occurrence of disasters (39). In this context, companies adopt Business Continuity Management (BCM) (54) to support the ability to operate in spite of unforeseen events and recover in a short time frame. The main BCM's outcome is the Business Continuity Plan (BCP) or Disaster Recovery Plan (DRP), which is a document that describes the

business continuity process in order to reduces or minimizes impact of events that disrupt critical services and their supporting resources (54). Two indexes are utilized to define survivability objectives: (i) Recovery Point Objective (RPO), which corresponds to the time limit of the most recent backup prior to disaster and (ii) Recovery Time Objective (RTO) that specifies the maximum time to repair the service after a disaster occurs. These objectives are based on business decisions that contemplate costs of inactivity periods and data loss. Additionally, technological factors (e.g., system performance) must be considered to stablish these parameters (55)(56).

Figure 3.2: RPO and RTO requirements.



Source: Made by author.

Figure 3.2 illustrates the backup operation along the time for a general system. During the system operation, a backup is periodically performed and whenever a disaster happens, the last backup should be recovered. If the age of the last backup (Recovery Point - RP) is higher than the RPO or the time to recover the system (Recovery Time - RT) is higher than the RTO, then the survivability requirements are not satisfied. In this case (Figure 3.2), the system meet the requirements. It is important to state that the amount of data that should be restored or backed up is not fixed for some applications. Consequently, the time to perform backup and restore operations is stochastic and depends on the amount of data involved and the technology utilized (56). Additionally, for some applications (e.g., financial trading systems) the RPO and RTO should not be higher than a few minutes. On the other hand, for other applications (e.g., static websites) these requirements are not so critical.

3.4 Modeling

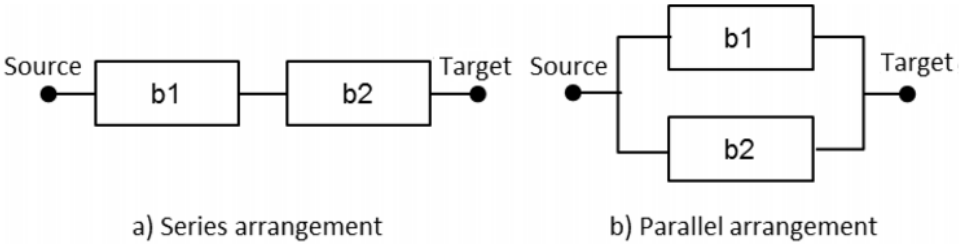
This section presents two important models for evaluating dependability and performance in general systems. Firstly, basic concepts about RBD model are presented. Then, the fundamentals of stochastic Petri nets are shown. It is important to emphasize that RBD models in this work are only adopted to estimate dependability metrics whereas SPN models evaluate system using survivability, dependability and performance metrics.

3.4.1 RBD Models

Reliability Block Diagram (57) is a combinatorial model that was initially proposed as a technique for calculating reliability on large and complex systems using intuitive block diagrams.

The blocks (e.g., components) are usually arranged using the following composition mechanisms: series, parallel, bridge, k-out-of-n blocks, or, even, a combination of previous approaches.

Figure 3.3: Reliability Block Diagram



Source: Made by author.

Figure 3.3 depicts two examples, in which blocks are arranged through series (Figure 3.3(a)) and parallel (Figure 3.3(b)) compositions. In the series arrangement, if a single component fails, the whole system is no longer operational. Assuming a system with n components, the reliability (availability) (57) is obtained by

$$P_s(t) = \prod_{i=1}^{n} P_i(t) \tag{3.1}$$

in which $P_i(t)$ is the reliability or the availability of block b_i.

For a parallel arrangement (see Figure 3.3(b)), at least, one component must be operation in order to the system be operational. Taking into account n components, the system reliability (availability) is

$$P_p(t) = 1 - \prod_{i=1}^{n} (1 - P_i(t)) \tag{3.2}$$

in which $P_i(t)$ is the reliability or the availability of block b_i. For other examples and closed-form equations, the reader should refer to (57).

3.4.2 SPN Models

Petri nets (PNs) (58) are a family of formalisms very well suited for modeling several system types, since concurrency, synchronization, communication mechanisms as well as deterministic and probabilistic delays are naturally represented. This work adopts a particular extension, namely, Stochastic Petri Nets (59), which allows the association of stochastic delays to timed transitions, and the respective state space can be converted into Continuous Time Markov chain (CTMC) (60). SPN models present a strong mathematical foundation, and they are suitable for representing and analyzing parallel systems with heterogeneous components and that exhibit concurrency and synchronization aspects. (26). Therefore, this formalism represent a prominent choice to model cloud computing systems. In SPNs, Places are represented by circles, whereas transitions are depicted as filled rectangles (immediate transitions) or hollow rectangles (timed transitions).

Arcs (directed edges) connect places to transitions and vice-versa. Tokens (small filled circles) may reside in places, which denote the state (i.e., marking) of a SPN. An inhibitor arc is a special arc type that depicts a small white circle at one edge, instead of an arrow, and they usually are used to disable transitions if there are tokens present in a place. The behaviour of a SPN is defined in terms of a token flow, in the sense that tokens are created and destroyed according to the transition firings (61). Immediate transitions represent instantaneous activities, and they have higher firing priority than timed transitions. Besides, such transitions may contain a guard condition, and a user may specificy a different firing priority among other immediate transitions. SPNs also allow the adoption of simulation techniques for obtaining system metrics, as an alternative to the generation of a CTMC. The extended stochastic Petri net (61) definition adopted in this work is:

Let $\mathscr{N} = (P, T, I, O, H, \Pi, M_0, Atts)$ be a stochastic Petri net (SPN), 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. $ordp_{\mathscr{N}}$ corresponds to a bijective function ($ordp_{\mathscr{N}} : P \to \{1, 2, ..., n\}$) that maps each place to a unique natural number.

- $T = \{t_1, t_2, ..., t_m\}$ is the set of transitions, which model active components. $ordt_{\mathscr{N}}$ is a bijective function ($ordt_{\mathscr{N}} : T \to \{1, 2, ..., m\}$) that maps each transition to a unique natural number.

- $I \in (\mathbb{N}^n \to \mathbb{N})^{n \times m}$ is a matrix of marking-dependent multiplicities of input arcs, where $I[ordp_{\mathscr{N}}(p_j), ordt_{\mathscr{N}}(t_k)]$ 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 $O[ordp_{\mathscr{N}}(p_k), ordt_{\mathscr{N}}(t_j)]$ 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 amount 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 $H[ordp_{\mathscr{N}}(p_j), ordt_{\mathscr{N}}(t_k)]$ 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.

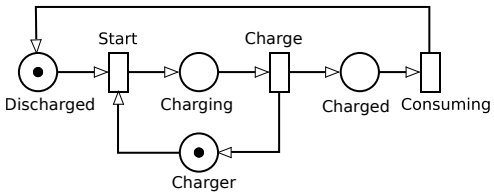
- $M_0 \in \mathbb{N}^n$ is a vector that contains the initial marking for each place (initial state). In this work, $M(p_n)$ denotes the number of tokens of place p_n at marking M.
- $Atts : (Dist, W, G, Policy, Concurrency)^m$ comprises a set of attributes for the m transitions, where
  - $Dist \in \mathbb{N}^m \rightarrow \mathscr{F}$ is a possibly marking dependent firing probability distribution function. In a stochastic timed Petri net, 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.
  - $G \in \mathbb{N}^n \rightarrow \{true, false\}$ 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. In this work, the notation $\#p_i$ has also been adopted for representing $m(p_i)$. For more detail about SPN concepts and semantic, the reader is referred to (59).

SPN modeling example

A simple cellphone charging/discharging process can be modeled through SPN. Initially, the cellphone battery is discharged and takes some time to start the charging process (mean duration $1/\lambda$). Then, the cellphone is connected to a charger and the charging process begins. The duration of charging process is assumed to be $1/r$ on average. After the charging process, the charger is removed and the discharging process is started (mean duration $1/\mu$). We can model the behavior of the charging/discharging process by using the SPN depicted in Figure 3.4. Figure 3.5 shows the correspondent reachability graph for that SPN model. Considering this SPN:

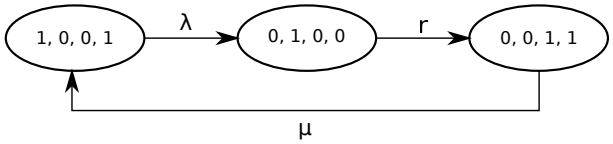
Figure 3.4: SPN representing a cell phone charging/discharging process.



Source: Made by author.

- place *Discharged* represents the local state of the cellphone when it is discharged;
- place *Charging* corresponds to the state of the cellphone when it is in charging process;
- place *Charged* is the state that models the situation when the cellphone is charged;
- place *Charger* represents the local state of the charger when it is not in use;
- transition *Start* models the action starting the cellphone charging process. The rate of this transition is λ;
- transition *Charge* represents the cellphone charging process itself. The rate of this transition is r;
- transition *Consuming* represents the cellphone discharging process. The rate of this transition is μ.

Figure 3.5: Reachability Graph of the SPN model of Figure 3.4.



Source: Made by author.

3.4.2.1 Place Invariants (P-Invariants)

In Petri nets, invariants are related to the conservative and repetitive stationary components, which are denoted by place invariants and transition invariants. Both invariants are useful to determine properties in Petri nets, but in this work the focus is on places invariants.

Definition 1 (Place Invariant - P-semiflow or P-Invariant)**.** *Assume a Petri net $\mathcal{N}$ with m places and its incidence matrix A. A vector of non-negative integers*

$$I_p = \begin{matrix} p_0 & p_1 & \dots & p_m \end{matrix} \begin{bmatrix} x_0 & x_1 & \dots & x_m \end{bmatrix}^T$$

is a P-semiflow or place invariant (P-invariant), if and only if ${I_p}^T \times C_{\mathcal{N}} = 0$. A value x_m is the weight associated with place p_m.

$C_{\mathcal{N}}$ corresponds to the Petri net incidence matrix, where $C_{\mathcal{N}} = O_{\mathcal{N}} - I_{\mathcal{N}}$. If there is a vector $I_p > 0$ and ${I_p}^T \times C_{\mathcal{N}} \leq 0$, $\mathcal{N}$ is *structurally bounded*. A Petri net $\mathcal{N}$ is *structurally conservative* if only if there is a P-invariant I_p, such that $I_p > 0$ (all elements are positive integers) and ${I_p}^T \times C_{\mathcal{N}} = 0$.

3.4.2.2 Transition Invariants (T-Invariants)

Definition 2 (Transition Invariant - T-semiflow or T-invariant)**.** *Assume a Petri net $\mathcal{N}$ with n transitions and its incidence matrix A. A vector of non-negative integers*

$$I_t = \begin{matrix} t_0 & t_1 & \dots & t_n \end{matrix} \begin{bmatrix} y_0 & y_1 & \dots & y_n \end{bmatrix}^T$$

is a T-semiflow or transition invariant (T-invariant), if and only if $C_{\mathcal{N}} \times I_t = 0$. A value y_n is the weight associated with transition t_n.

If there is a P-invariant (or T-invariant) such that all weights are not null, a Petri net is said to be *covered* with P-invariants (or T-invariants). A net is *consistent* if exists a T-invariant $I_t > 0$ and $C_{\mathcal{N}} \times I_t = 0$. If there is a vector $I_t > 0$ and $C_{\mathcal{N}} \times I_t \geq 0$, the net is *structurally repetitive*.

3.4.2.3 Juxtaposition of Invariants

The composition of Petri net basic building blocks can be adopted to represent large and complex systems. For instance, model generation tools (e.g., GeoClouds Modcs (62)) can be adopted to automatically create high level models by merging basic building blocks. In this thesis, a proof is provided to demonstrate that the proposed modelling approach always create models with structural properties (see Section 5.3). Place and transition invariants represent important

methods to determine structural properties in Petri net models. The following definition present a technique for obtaining P-invariants of Petri nets composed by union of Places of other Petri nets (63).

Definition 3 (Juxtaposition of P-invariants - $\mathscr{J}$). *Let $\mathscr{N}_1$ and $\mathscr{N}_2$ be two Petri nets, and P_1 and P_2 be the respective sets of places, such that $P_1 \cap P_2 = P_s \neq \emptyset$.*

$$\mathscr{I}_{p(1)} = \begin{matrix} p_0^1 & \cdots & p_n^1 & p_{k_0} & \cdots & p_{k_q} \end{matrix} \begin{bmatrix} x_0^1 & \ldots & x_n^1 & x_{k_0}^1 & \ldots & x_{k_q}^1 \end{bmatrix}^T$$

is a P-invariant of net $\mathscr{N}_1$ and

$$\mathscr{I}_{p(2)} = \begin{matrix} p_{k_0} & \cdots & p_{k_q} & p_0^2 & \cdots & p_m^2 \end{matrix} \begin{bmatrix} x_{k_0}^2 & \ldots & x_{k_q}^2 & x_0^2 & \ldots & x_m^2 \end{bmatrix}^T$$

is a P-invariant of net $\mathscr{N}_2$. Finally, assume a Petri net $\mathscr{N}_3$ generated by merging places of subnets $\mathscr{N}_1$ and $\mathscr{N}_2$, more specifically, P^1_{shared} and P^2_{shared}. A P-invariant $\mathscr{I}_{p(3)}$ of net $\mathscr{N}_3$ may be obtained from $\mathscr{I}_{p(1)}$ and $\mathscr{I}_{p(2)}$ via juxtaposition, $\mathscr{I}_{p_3} = \mathscr{J}(\mathscr{I}_{p(1)}, \mathscr{I}_{p(2)})$, if the following condition holds: $\forall p \in P_1 \cap P_2, \mathscr{I}_{p(1)}(p) = \mathscr{I}_{p(2)}(p)$. Thus, assuming $x^1_{k_0} = x^2_{k_0} = x_{k_0}$, ... ,$x^1_{k_q} = x^2_{k_q} = x_{k_q}$,

$$\mathscr{I}_{p_3} = \begin{matrix} p_0^1 & \cdots & p_n^1 & p_{k_0} & \cdots & p_{k_q} & p_0^2 & \cdots & p_m^2 \end{matrix} \begin{bmatrix} x_0^1 & \ldots & x_n^1 & x_{k_0} & \ldots & x_{k_q} & x_0^2 & \ldots & x_m^2 \end{bmatrix}^T.$$

For the condition stated in Definition 3 to be satisfied, sometimes the invariants need to be multiplied by positive integers. For instance, assume a net $\mathscr{N}_3$ composed by merging places of subnets $\mathscr{N}_1$ and $\mathscr{N}_2$. Additionally, consider the P-invariant $\mathscr{I}_{p(1)} = [x_1 \; x_{k_1}]$ of net $\mathscr{N}_1$, in which x_{k_1} represents the weights associated to the merged places and x_1 the weights associated to the untouched ones. Similarly, $\mathscr{I}_{p(2)} = [x_{k_2} \; x_2]$ represents a P-invariant of net $\mathscr{N}_2$, in which x_{k_2} represents the weights associated to the merged places. If $x_{k_1} \neq x_{k_2}$, it is necessary to find $a, b \in \mathbb{N}$ such that $a.x_{k_1} = b.x_{k_2}$, which will result in $\mathscr{I}_{p(3)} = \mathscr{J}(a.\mathscr{I}_{p(1)}, b.\mathscr{I}_{p(2)})$. If it is not possible to find $a, b \in \mathbb{N}$ that satisfy the condition, the juxtaposition of these P-invariants can not be performed.

Consider this example of juxtaposition technique (64). Assume a net $\mathscr{N}_c$ composed by merging a common place of subnets $\mathscr{N}_a$ and $\mathscr{N}_b$ ($P_a \cap P_b = \{p_{shared}\}$). Additionally, consider the following basic P-invariants for each subnet:

- $\mathscr{N}_a$: $\mathscr{I}_{p(a)(1)} = \begin{matrix} p_0 & p_1 & p_{shared} \end{matrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T$ and $\mathscr{I}_{p(a)(2)} = \begin{matrix} p_0 & p_1 & p_{shared} \end{matrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}^T$;
- $\mathscr{N}_b$: $\mathscr{I}_{p(b)(1)} = \begin{matrix} p_{shared} & p_2 & p_3 & p_4 \end{matrix} \begin{bmatrix} 5 & 1 & 1 & 0 \end{bmatrix}^T$ and $\mathscr{I}_{p(b)(2)} = \begin{matrix} p_{shared} & p_2 & p_3 & p_4 \end{matrix} \begin{bmatrix} 5 & 1 & 0 & 1 \end{bmatrix}^T$.

The following lines demonstrate the P-invariants obtained through juxtaposition for net $\mathscr{N}_c$:

- $\mathscr{I}_{p(c)(1)} = \mathscr{J}(\mathscr{I}_{p(a)(1)}, 0.\mathscr{I}_{p(b)(1)}) = \begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$;
- $\mathscr{I}_{p(c)(2)} = \mathscr{J}(\mathscr{I}_{p(a)(1)}, 0.\mathscr{I}_{p(b)(2)}) = \begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$;
- $\mathscr{I}_{p(c)(3)} = \mathscr{J}(5.\mathscr{I}_{p(a)(2)}, \mathscr{I}_{p(b)(1)}) = \begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ 5 & 0 & 5 & 1 & 1 & 0 \end{bmatrix}^T$;
- $\mathscr{I}_{p(c)(1)} = \mathscr{J}(5.\mathscr{I}_{p(a)(2)}, \mathscr{I}_{p(b)(2)}) = \begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ 5 & 0 & 5 & 1 & 0 & 1 \end{bmatrix}^T$.

A general P-invariant for $\mathscr{N}_c$ (as well as for $\mathscr{N}_a$ and $\mathscr{N}_b$) can be obtained as follows:

- $\mathscr{I}_{p(a)} = \alpha.\mathscr{I}_{p(a)(1)} + \beta.\mathscr{I}_{p(a)(2)} = \begin{bmatrix} p_0 & p_1 & p_{shared} \\ \alpha+\beta & \alpha & \beta \end{bmatrix}^T$, where $\alpha, \beta \in \mathbb{N}$;
- $\mathscr{I}_{p(b)} = \gamma.\mathscr{I}_{p(b)(1)} + \delta.\mathscr{I}_{p(b)(2)} = \begin{bmatrix} p_{shared} & p_2 & p_3 & p_4 \\ 5\gamma+5\delta & \gamma+\delta & \gamma & \delta \end{bmatrix}^T$, where $\gamma, \delta \in \mathbb{N}$;
- Assuming $\beta = 5\gamma+5\delta$, $\mathscr{I}_{p(c)} = \mathscr{J}(\mathscr{I}_{p(a)}, \mathscr{I}_{p(b)}) =$

$$\begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ \alpha+5\gamma+5\delta & \alpha & 5\gamma+5\delta & \gamma+\delta & \gamma & \delta \end{bmatrix}^T.$$

Since $\exists \alpha, \gamma, \delta \in \mathbb{N}^*, {\mathscr{I}_{p(c)}}^T \times A_c = 0$, in which A_c is the incidence matrix of net $\mathscr{N}_c$, $\mathscr{N}_c$ is conservative as well as structurally bounded, in other words, for any initial marking, the state space size is finite.

3.4.2.4 Distribution Moment Matching

A well-established method that considers *expolynomial distribution* random variables is based on distribution moment-matching . The moment matching process presented in (65) and considers that Hypoexponential and Erlangian distributions have the average delay (μ) greater than the standard-deviation (σ) -$\mu > \sigma$-, and Hyperexponential distributions have $\mu < \sigma$, in order to represent an activity with a generally distributed delay as an Erlangian or a Hyperexponential subnet referred to as s-transition. One should note that in cases where these distributions have $\mu = \sigma$, they are, indeed, equivalent to an exponential distribution with parameter equal to $\frac{1}{\mu}$. Therefore, according to the coefficient of variation associated with an activity's delay, an appropriate s-transition implementation model could be chosen. For each s-transition implementation model (see Figure 3.6), a set of parameters should be configured for matching their first and

second moments. In other words, an associated delay distribution (it might have been obtained by a measuring process) of the original activity is matched with the first and second moments of s-transition (*expolynomial distribution*). According to the aforementioned method, one activity with $\mu<\sigma$ is approximated by a two-phase Hyperexponential distribution with parameters

$$r_1 = \frac{2\mu^2}{(\mu^2+\sigma^2)}, \tag{3.3}$$

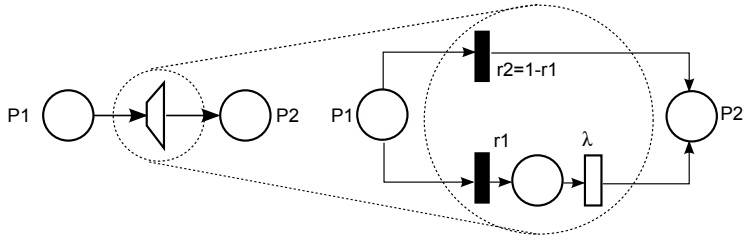
$$r_2 = 1 - r_1 \tag{3.4}$$

and

$$\lambda = \frac{2\mu}{(\mu^2+\sigma^2)}. \tag{3.5}$$

where λ is the rate associated to phase 1, r_1 is the probability of related to this phase, and r_2 is the probability assigned to phase 2. In this particular model, the rate assigned to phase 2 is assumed to be infinity, that is, the related average delay is zero.

Figure 3.6: Hyperexponential Model



Source: Made by author.

Activities with coefficients of variation less than one might be mapped either to Hypo-exponential or Erlangian s-transitions. If $\frac{\mu}{\sigma} \notin \mathbb{N}, \frac{\mu}{\sigma} \neq 1, (\mu, \sigma \neq 0)$, the respective activity is represented by a Hypoexponential distribution with parameters λ_1, λ_2(exponential rates); and γ, the integer representing the number of phases with rate equal to λ_2, whereas the number of phases with rate equal to λ_1 is one. In other words, the s-transition is represented by a subnet composed of two exponential and one immediate transitions. The average delay assigned to the exponential transition t_1 is equal to μ_1 ($\lambda_1 = 1/\mu_1$), and the respective average delay assigned to the exponential transition t_2 is μ_2 ($\lambda_2 = 1/\mu_2$). γ is the integer value considered as the weight assigned to the output arc of transition t_1 as well as the input arc weight value of the immediate transition t_3 (see Figure 3.7). These parameters are calculated by the following expressions:

$$(\frac{\mu}{\sigma})^2 - 1 \leq \gamma < (\frac{\mu}{\sigma})^2, \tag{3.6}$$

$$\lambda_1 = \frac{1}{\mu_1} \; and \; _2 = \frac{1}{\mu_2}, \tag{3.7}$$

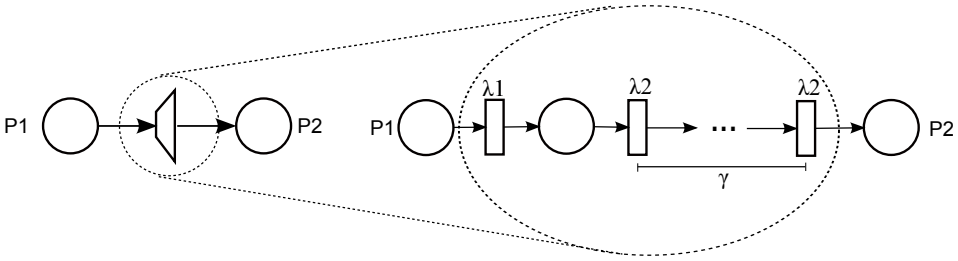
where

$$\mu_1 = \frac{\mu \pm \sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu^2}}{\gamma+1}, \tag{3.8}$$

$$\mu_2 = \frac{\gamma\mu \mp \sqrt{\gamma(\gamma+1)\sigma^2 - \gamma\mu^2}}{\gamma+1} \tag{3.9}$$

If $\frac{\mu}{\sigma} \in \mathbb{N}, \frac{\mu}{\sigma} \neq 1, (\mu, \sigma \neq 0)$, an Erlangian s-transition with two parameters, $\gamma = (\frac{\mu}{\sigma})^2$ is an integer representing the number of phases of this distribution; and $\mu_1 = \mu/\gamma$, where $\mu_1(1/\lambda_1)$ is the average delay value of each phase. The Erlangian model is a particular case of a Hypoexponential model, in which each individual phase rate has the same value. The reader should refer to (65) for details regarding the representation of expolinomial distributions using SPN.

Figure 3.7: Hypoexponential Model



Source: Made by author.

3.5 Fault Injection Techniques

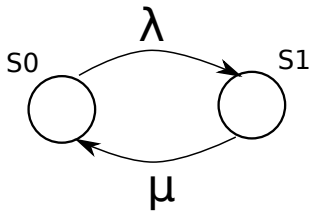
In general, a fault injection and monitoring based strategy encompasses a workload generator (random fault and repairing activities generator), a fault injector, a system monitor and the target system. The fault injector is responsible to inject faults/repairs events to the target system. The workload generator generates commands (faults and repairing actions) and drives the fault injector, whilst the system monitor observes the target system status and behavior, collects data and provides measures and statistics (66)(67)(68).

A hardware fault injector requires specific hardware components to be added to the system which operate at the physical level (e.g., circuits that modify electrical components). A software fault injector, on the other hand, employs programed software to interact with the target system, and is therefore more flexible than the hardware injector (68). Simulation-based fault injection implies the creation of models, the introduction of faults into the model, and observation to discover what effect the fault has on the model. Finally, the execution-based injector requires the real system, introducing faults into it (67).

3.5.1 Confidence interval of availability estimation

This section presents the approach to calculate the confidence interval of availability of a general system (69). This method observes system failures and repairs to estimate the availability confidence interval. In this particular work, this method is adopted for evaluating the system availability confidence interval under fault injection.

Figure 3.8: Markov Chain.



Source: Made by author.

The Markov Chain (26) depicted in Figure 3.8 represents a top down view of system behaviour. State $S0$ represents the system at normal condition and State $S1$ denotes the system failure. The transition from State $S0$ to State $S1$ denotes a failure. The repair activity is depicted by the transition from State $S1$ to State $S0$. The failure rate is λ and the repair rate is specified by μ. System availability can be calculated by Equation 3.10.

$$A = \frac{\mu}{\lambda + \mu} = \frac{1}{1 + \frac{\lambda}{\mu}} = \frac{1}{1 + \rho}, \tag{3.10}$$

Here ρ is the ratio λ/μ. If it is assumed that n failure and repair events were observed during the experiment, then the total failure time is S_n and the total repair time is Y_n. Therefore, the maximum-likelihood estimator for λ is defined in Equation 3.11

$$\hat{\Lambda} = \frac{n}{S_n} \tag{3.11}$$

A $100\text{x}(1-\alpha)$ confidence interval for λ is given by Equation 3.12.

$$\left(\frac{C^2_{2n;1-\frac{\alpha}{2}}}{2S_n}, \frac{C^2_{2n;\frac{\alpha}{2}}}{2S_n}\right) \tag{3.12}$$

An analogous process is performed estimate μ (Equation 3.13).

$$\hat{M} = \frac{n}{Y_n} \tag{3.13}$$

The $100\text{x}(1-\alpha)$ confidence interval for μ is defined in Equation 3.14.

$$\left(\frac{C^2_{2n;1-\frac{\alpha}{2}}}{2Y_n}, \frac{C^2_{2n;\frac{\alpha}{2}}}{2Y_n}\right) \tag{3.14}$$

Consequently, the maximum-likelihood estimator for the ratio λ/μ is $\hat{\rho}$ and is defined by Equation 3.15.

$$\hat{\rho} = \frac{\hat{\Lambda}}{\hat{M}} = \frac{\frac{n}{S_n}}{\frac{n}{Y_n}} = \frac{Y_n}{S_n} \tag{3.15}$$

The $100\text{x}(1-\alpha)$ confidence interval for ρ is given by $(\rho_L, \rho u)$, through F-distribution probability function as specified in Equation 3.16.

$$\rho_l = \frac{\hat{\rho}}{f_{2n;2n;\frac{\alpha}{2}}} \; and \; \rho_u = \frac{\hat{\rho}}{f_{2n;2n;1-\frac{\alpha}{2}}} \tag{3.16}$$

Finally, the maximum-likehood estimator to availability is $\hat{A} = 1/(1+\hat{\rho})$. Since the availability A is a monotonically decreasing function of ρ, the $100\text{x}(1-\alpha)$ confidence interval for A is:

$$(\frac{1}{1-pu}, \frac{1}{1-pl}) \tag{3.17}$$

3.6 Final Remarks

This chapter presented the basic concepts related to the proposed framework, ranging from the definition of system architecture to the estimation of availability confidence interval. Initially, disaster tolerant IaaS system infrastructures were presented focusing on its internal components and behavior.

Afterwards, the concept of performance and dependability were presented. Next, survivability was conceptualized in the context of cloud computing systems. After that, attention was devoted to the adopted models (RBD and SPN), giving particular focus to stochastic Petri nets. Finally, a small review on the concepts related to Fault injection and confidence interval of availability was presented.

4

Modeling Approach

This chapter presents the adopted model in this work. Firstly, the proposed evaluation process is shown. Next, the high-level IaaS model is presented. Then, the performability models are detailed. Finally, the survivability models are discussed.

4.1 Evaluation Process

The main purpose of this work is to propose a set of models, framework, and modeling process for evaluating geographically distributed cloud computing systems taking into account disaster occurrences.

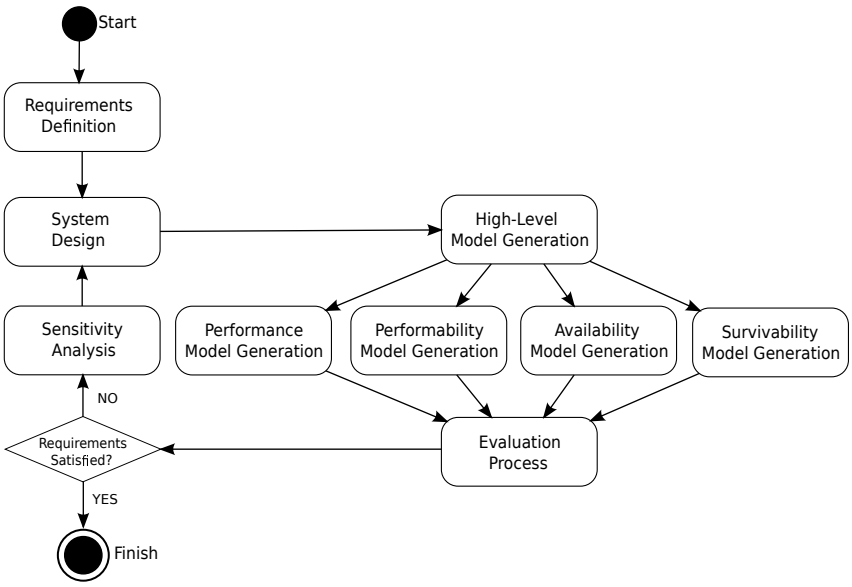
A hybrid modeling strategy which combines combinatorial and state-based models is adopted for evaluating availability, performance and survivability in cloud computing systems. RBD models allow designers to evaluate system dependability by using closed-form equations (26). However, components dependencies are hard to represented in RBD models. On the other hand, state-based models (e.g., SPNs) can represent those dependencies in a simple way. Nevertheless, these models may suffer from state-space explosion problem (59).

This thesis proposes a combined modeling approach based on RBD and SPN models to represent geographically distributed cloud computing systems. RBD models are solved by adopting closed-form equations and analysis or simulation is utilized for computing SPN results. Figure 4.1 depicts an overview of the proposed approach for evaluating geographically distributed cloud computing systems.

Requirements Definition. The evaluation approach's first step concerns requirements definition. In this phase, users provide a set of quality parameters that define the system proper behavior. In this approach, a disaster tolerant cloud computing system can be evaluated to check the satisfaction of the following requirements (26):

- Minimum availability.
- Minimum Capacity Oriented Availability (COA).
- Minimum probability of finishing a request.
- Maximum system utilization.

Figure 4.1: Proposed Approach Overview



Source: Made by author.

- Minimum probability of recovering the system after a disaster. This value is checked at a user specific time.

System Design. The evaluation's second step is related to the system design. In this phase, the disaster tolerant cloud computing system is defined (Section 3.1). The result (artifact) of this evaluation step is a document containing the set of facilities (data centers and backup server), its characteristics (servers and network devices configuration), the disaster characteristics of each location, and the VM transmission characteristics.

High-Level Model Generation. The next stage involves the creation of high-level models that represent the system infrastructure and interactions (Section 4.2). The artifact of this phase corresponds to an instance of high-level model.

Generate Evaluation Model. High-Level Models can be converted to performability or survivability models. It is important to state that submodels may be generated to mitigate the complexity of the SPN final model. The submodels are combined to create the evaluation model. The result of this stage corresponds to an evaluation model (availability, performance, performabilty or survivability). It is important to stress that Geoclouds can be adopted to help the users to automatically create the high-level models, translate them to stochastic models and evaluate the metrics.

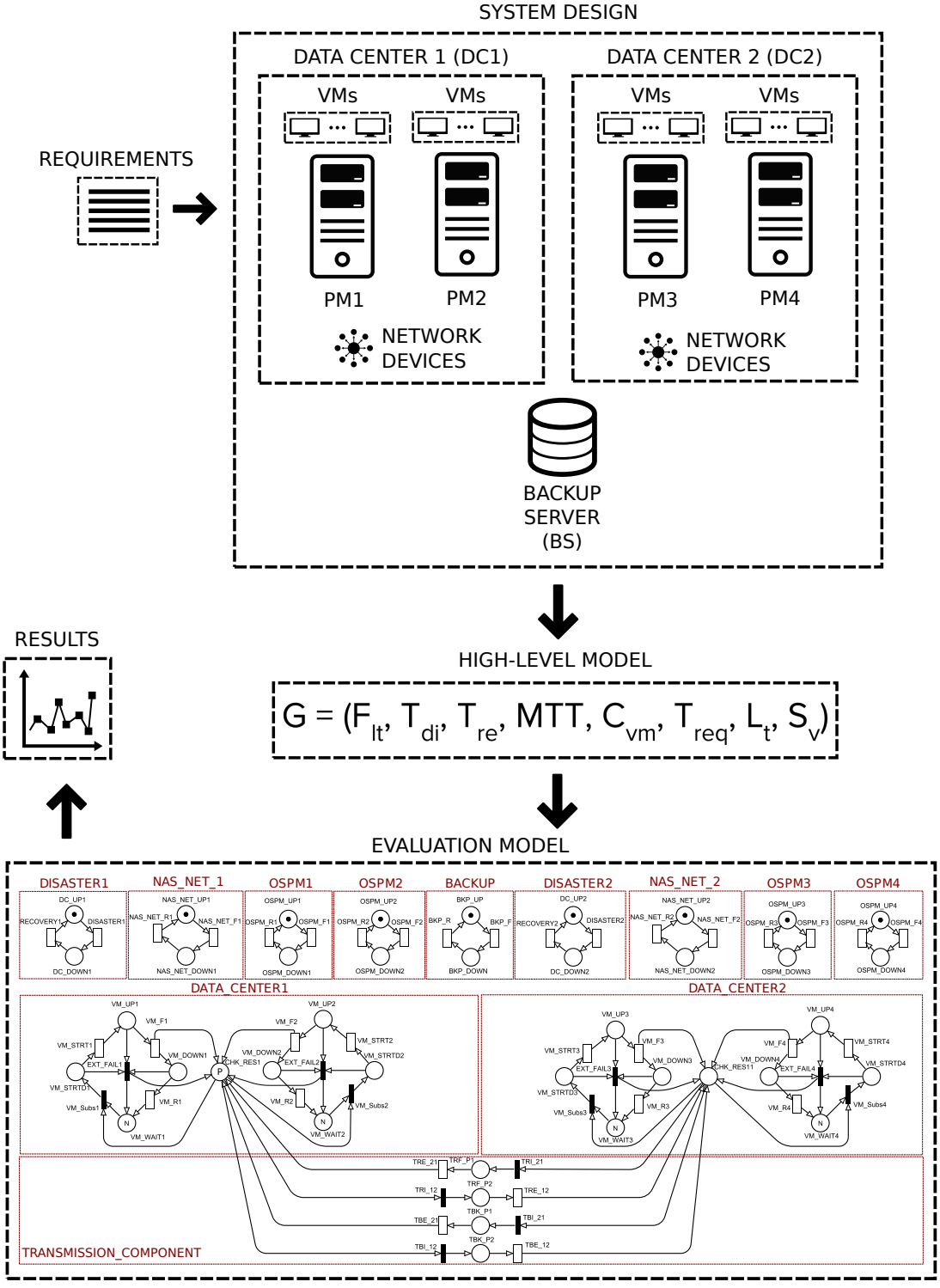
Evaluation Process. The next step involves the final model evaluation, which estimates the user defined metrics. The evaluation can be performed by SPN analysis or simulation through Mercury (70) (71) or Timenet (72) tools. A list of evaluated metrics is provided as the output of this phase.

Requirements Checking. The requirements are checked, and the process finishes if the evaluated results meet the user needs. If the modelled system does not comply with the

requirements, the system must be redesigned.

Sensitivity Analysis. A sensitivity analysis can be performed to stablish the model parameter that most affect the user metrics. The sensitivity analysis approach proposed by Matos et al (73) can be adopted to determine the parameters that cause the greatest impact on system

Figure 4.2: Evaluation Example



Source: Made by author.

metrics.

4.1.1 Evaluation Example

Figure 4.2 presents an example of IaaS disaster tolerant cloud evaluation. In this example, the cloud system is deployed into two data centers (DC1 and DC2). After the system design (see Figure 4.1), a high-level is created for representing each system component and the relationship between them (Section 4.2). As stated before, users can create the high-level models from scratch or utilize GeoClouds Modcs (Section 6) to guide the model construction. In this example, G corresponds to a tuple that represents data centers DC1 and DC2 as well as the backup server.

The next phase involves the definition of an evaluation model that represents the system behavior and allow users to estimate system metrics. The proposed translation algorithm guide users to create and evaluate the system models. This translation process creates basic SPN components for representing each system component. As long as new basic blocks are created, the algorithm combines that components to the final model. Observe that, in this figure DC1 and DC2 are represented in the final model. Section 5.1 presents details on how the evaluation model is created based on the high-level model.

A list of evaluated metrics is generated and these metrics are compared to the requirements. If the requirements are not satisfied the system must be redesigned. Next section presents details related to the models adopted to access availability, performance and survivability in disaster tolerant IaaS cloud computing systems.

4.2 High-Level IaaS Model

This section presents a high-level model to represent an IaaS disaster tolerant cloud system. These models are adopted to represent the IaaS system design considering a mathematical modeling. This model includes representation of facilities (data centers and backup server), disaster characteristics, VM synchronization function, and survivability parameters. The tuple $G = (F_{lt}, T_{di}, T_{re}, MTT, C_{VM}, T_{req}, L_t, S_v)$ corresponds to a geographically distributed IaaS system representation in which:

- F_{lt} is a finite set of data centers including the backup server (facilities), such that $F_{lt} = D \cup BS$. D is a finite set of data centers and BS represents the set of backup servers;
- $T_{di} : F_{lt} \rightarrow f_{di}$ denotes the disaster occurrence function. For each facility $d_c \in F_{lt}$, a probability distribution function (pdf) f_{di} is associated. The function f_{di} provides the probability of a disaster for each instant t;
- $T_{re} : F_{lt} \rightarrow f_{re}$ represents the disaster recovery function. Similarly to the previous function, it associates a PDF (f_{re}) with each facility $d_c \in F_{lt}$. For each time t a probability of disaster recovery is provided;

- $MTT : F_{lt} \times F_{lt} \rightarrow f_{MTT}$ denotes the VM transmission function. The function relates a pair of facilities $(d_{c1}, d_{c2}) \in F_{lt} \times F_{lt}$ to a PDF f_{MTT}. The resulted function f_{MTT} provides the probability of finishing the data transmission between d_{c1} and d_{c2} at time t;
- $C_{VM} \in \mathbb{N}$ is the maximum number of VM requests that can be performed by users.
- $T_{req} : [0, \infty) \rightarrow [0, 1]$ corresponds to the VM request time probability function, which provides a PDF that associates each time $t \in [0, \infty)$ with the probability to perform a VM request.
- $L_t : [0, \infty) \rightarrow [0, 1]$ is the VM life-time probability function. It corresponds to a PDF that relates each VM execution time with a probability.
- $S_v = (v_{bd}, b_{pd}, r_{vd})$ corresponds to survivability parameters. v_{bd} is the number of VMs that must be periodically backed up, b_{pd} represents the backup period and r_{vd} is the number of VMs that should be recovered to restore the service.

A data center $dc \in D$ corresponds to the ordered pair (P_d, C_d), where P_d represents a physical machine finite set. C_d represents the finite set of basic components related to the network infrastructure. A physical machine $p \in P_d$ corresponds to the tuple (V_p, S_p, os, hw, m) where:

- V_p represents a virtual machine finite set assigned to the physical machine at cloud system start up;
- $S_p : V_p \rightarrow f_p$ provides the virtual machine set up time probability distribution function;
- $os \in O_p$ corresponds to the physical machines's software component;
- $hw \in H_p$ represents the hardware of the physical machine;
- $m \in \mathbb{N}$ denotes the maximum number of VMs that the physical machine can execute;

O_p and H_p are finite sets of software and hardware components related to physical machines. C ($C = C_d \cup O_p \cup H_p \cup V_p$) corresponds to a finite set of all data center's basic components. $T_{fr} : C \rightarrow f_{fr}$ represents the failure probability distribution function associated with component $c \in C$, and $T_{rp} : C \rightarrow f_{rp}$ represents the repair PDF associated with component $c \in C$.

4.3 Performance and Availability Models

This section presents the performance, availability, and performability models adopted in this work. Five SPN submodels are detailed and they represent the system behavior considering disasters, VM transfer, user requests and failures characteristics. Section 5.1 describes how

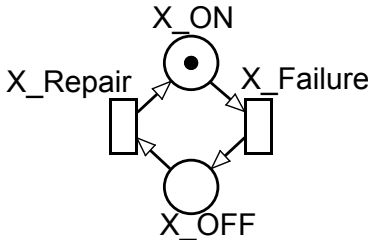
to combine the submodels to estimate availability and performance metrics. This modeling approach allows the creation of four different models to evaluate the IaaS system.

Three different model representations can be utilized to estimate performance, dependability, and performability metrics of IaaS disaster tolerant clouds. Depending on the user metrics, performance, availability or performability models can be created to evaluate the system. For instance, if only availability metrics are considered, the availability model should be adopted. It is important to highlight that the following models do not consider backup time and frequency. In the following models, we assume that the backup is always updated. Section 4.4 presents models that represent the backup process for IaaS cloud systems.

4.3.1 SPN block: generic component

Generic component (Figure 4.3) is adopted for availability and performability models to represent devices that have no redundancy and might be in two states, either functioning or failed.

Figure 4.3: Generic component SPN model



Source: Made by author.

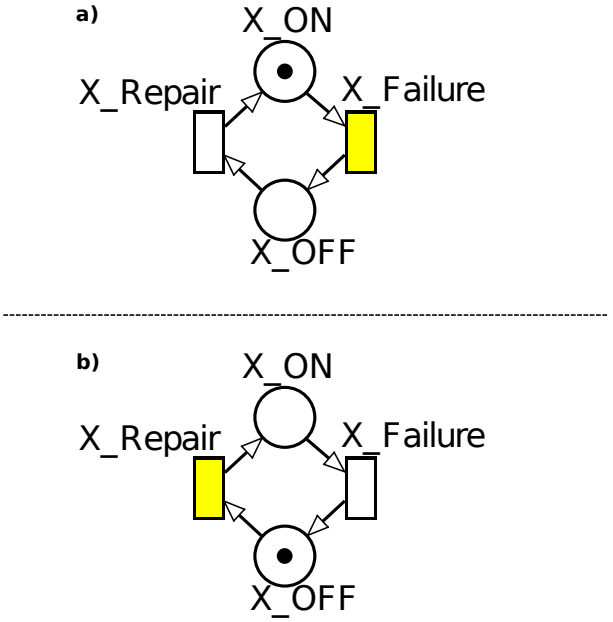
The respective SPN model of this component is shown in Figure 4.3. Places *X_ON* and *X_OFF* are the model component's activity and inactivity states, respectively. In this representation, a subsystem can be operational (*X_ON* marked) or failed (*X_OFF* marked). Label "X" is instantiated according to the component name, for instance, for a component named HW1 these places will be *HW1_ON* and *HW1_OFF*. A component is operational only if the number of tokens in place *X_ON* is greater than zero. Equation 4.1 can be adopted to estimate system availability (*Av*) of generic components.

$$Av = P\{\#X\_ON > 0\} \tag{4.1}$$

Dynamic Behavior of Generic Component.

Figure 4.4 presents the flow of tokens to represent failure/repair behavior of a given subsystem (e.g., Server). The dynamic behavior of this component is represented as follows:

Figure 4.4: Token flow related to a generic component



Source: Made by author.

- **System Operational - Figure 4.4(a)**. Initially, place *X_ON* has one token indicating that the system is operational. When *X_Failure* fires (failure action), the token is removed from pace *X_ON*, and a new token is created in X_OFF.
- **System Failed - Figure 4.4(b)**. In this situation, the system is failed and a repair activity is enabled. When the repair activity is finished (i.e., *X_Repair* fires), the system returns to the initial state.

Formal Description of Generic Component.

A generic component is modeled by a SPN $\mathscr{B}_{ge} = (P_{ge}, T_{ge}, I_{ge}, O_{ge}, H_{ge}, \Pi_{ge}, M_{ge}, Atts_{ge})$, in which:

- $P_{ge} = \{X\_ON, X\_OFF\}$. These places model the following situations:
  - *X_ON*: component operational.
  - *X_OFF*: component failed.
- $T_{ge} = \{X\_Failure, X\_Repair\}$. These transitions model the following actions:
  - *X_Failure*: failure action.
  - *X_Repair*: repair action.

- The component's structure can be represented by the matrix $C_{ge} = O_{ge} - I_{ge} =$

$$\begin{array}{c} \\ X\_ON \\ X\_OFF \end{array} \begin{array}{c} \begin{array}{cc} X\_Failure & X\_Repair \end{array} \\ \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}$$

- This component has no inhibitor arcs, i.e. $H_{ge} =$

$$\begin{array}{c} \\ X\_ON \\ X\_OFF \end{array} \begin{array}{c} \begin{array}{cc} X\_Failure & X\_Repair \end{array} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \end{array}$$

- All transitions have the same priority level: $\Pi_{ge} = (1,1)$.
- $M_{ge} = (r_{es}, 0)$, where $r_{es} \in \mathcal{N}$ represents the number of resources that can be operational or failed.
- $Atts_{ge} = (Dist_{ge}, W_{ge}, G_{ge}, Pol_{ge}, Con_{ge})^2$ corresponds to attributes for the two model transitions:
  - $Dist_{ge}$ corresponds to the firing probability distribution function associated to each transition. These probability distribution functions depend on the pdf adopted in the high-level model.
  - W_{ge} corresponds to the weight function. In the case of the transition with exponential distributed firing delay, this weight model the rate λ_t for the transition.
  - Pol_{ge} is *preemptive resume* for all transitions (74).
  - Con_{ge} is infinite server *is* for all transitions (74).

This component contains a P-invariant covering all places given by:

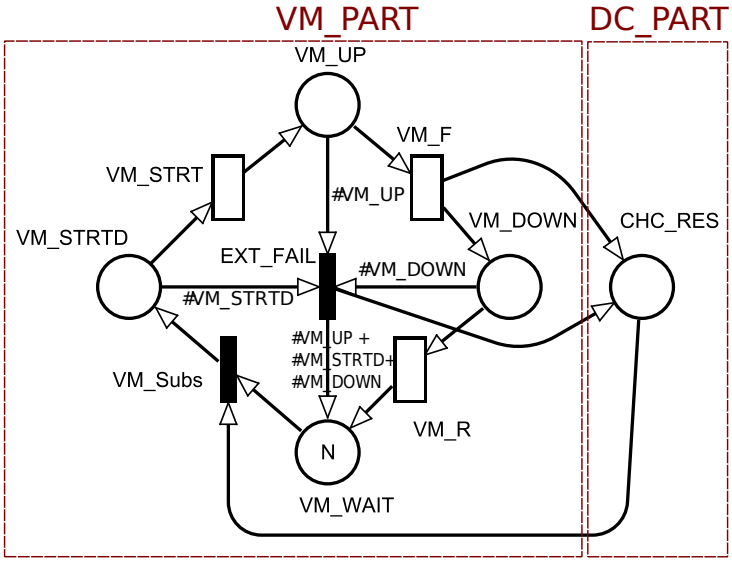
$$\mathscr{I}_{ge} = \begin{array}{c} \begin{array}{cc} X\_ON & X\_OFF \end{array} \\ \begin{bmatrix} w & w \end{bmatrix}^T \end{array}.$$

Since, ${\mathscr{I}_{ge}}^T \times C_{ge} = 0$ and ${\mathscr{I}_{ge}}^T > 0$, the generic block is structurally conservative as well as structurally bounded.

4.3.2 SPN block: VM availability component

VM availability component can be adopted in availability models (see Section 4.1) and represent failures and repairs of running VMs on a particular server. Figure 4.5 presents the SPN block, which is composed of two parts: (i) *VM_PART* that represents the behavior of running VMs on a single machine; and (ii) *DC_PART* which expresses the waiting VMs to be instantiated.

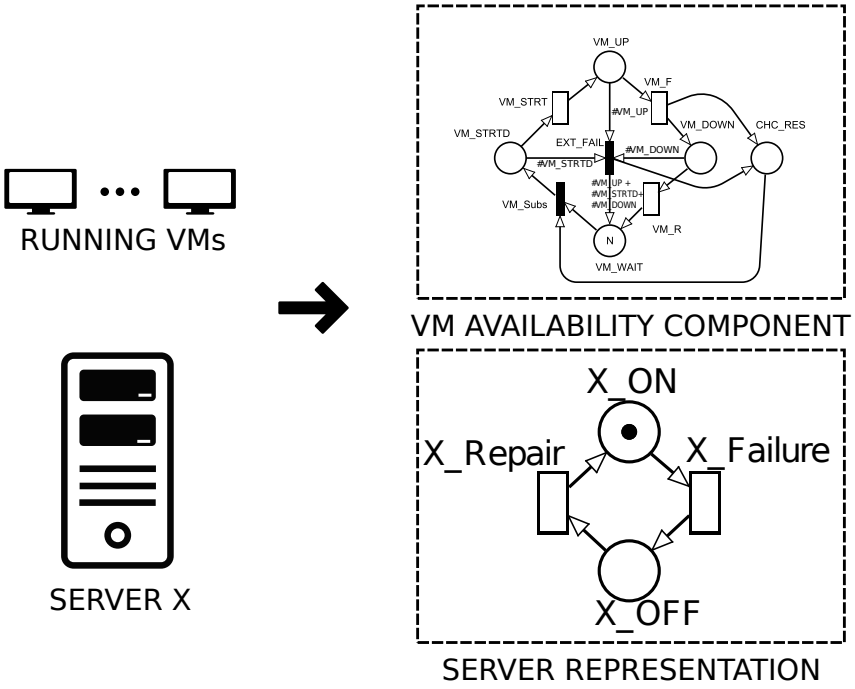
Figure 4.5: VM availability component



Source: Made by author.

Figure 4.6 presents how failure/recovery behavior of VMs and the underlying server can be represented by using a generic component (for the server) and a VM availability component (for virtual machines). Each server is modeled by a generic component, and the VMs that run on this server are represented by a VM availability component.

Figure 4.6: VM availability component representation



Source: Made by author.

Considering this SPN submodel, if the external infrastructure or the physical machine fails during the virtual machine execution, the affected VMs should be migrated to another physical machine (in the same data center or in another). If there is no available physical machine

in the system, failed VMs wait to be instantiated in a physical machine after the repair activity. This component interacts with three generic components: (i) one representing the occurrence of disasters (*DC*); (ii) the network infrastructure (*NAS_NET*); and (iii) the physical machine (*OSPM*).

Regarding *VM_PART*, places *VM_UP*, *VM_DOWN*, *VM_STRTD* and *VM_WAIT* denote, respectively, the amount of VMs in states operational, failed, starting, and waiting for request. Transitions *VM_F*, *VM_R* and *VM_STRT* represent the failure, repair and starting activities related to the virtual machines. The association with the underlying infrastructure is carried out by immediate transitions *EXT_FAIL*, *VM_Subs* and the respective guard conditions (see Table 4.1). *EXT_FAIL* transition verifies external problems that occurs whenever a disaster has occurred (i.e., *#DC_UP=0*) or the underlying physical machine is broken (*#OSPM_UP=0*) or the network is not working (*#NAS_NET_UP=0*). A VM fails whenever the respective infrastructure is not capable to provide the service. Transition *VM_Subs* denotes the opposite idea, in the sense that virtual machines start only if the required infrastructure is operational *((#OSPM_UP>0) AND (#NAS_NET_UP>0) AND (#DC_UP>0))*.

Table 4.1: Guard Expressions for VM availability component.

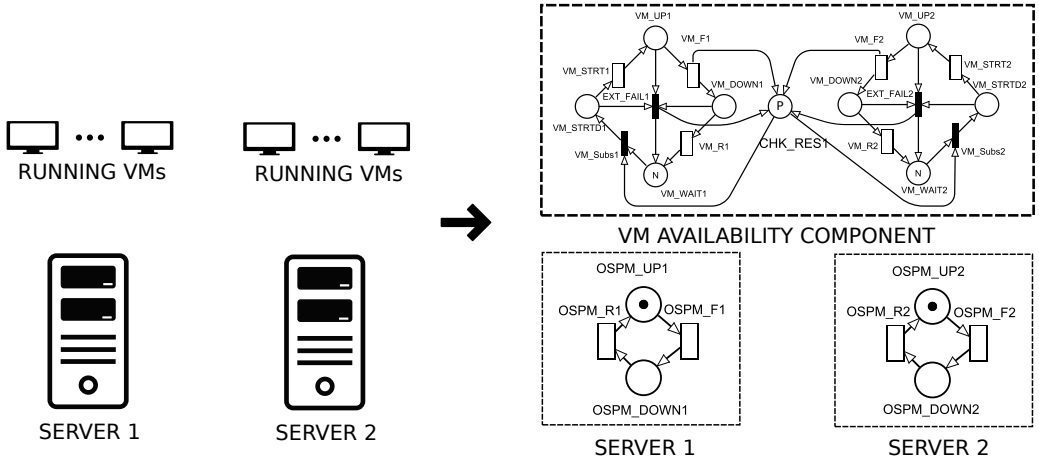
Transition	Condition	Description
EXT_FAIL	*(#OSPM_UP=0) OR (#NAS_NET_UP=0) OR (#DC_UP=0) AND* ((#VM_UP + #VM_DOWN + #VM_STRTD) > 0)	Failure of physical machine or infrastructure
VM_Subs	*(#OSPM_UP>0) AND (#NAS_NET_UP>0) AND (#DC_UP>0)*	Physical machine and infrastructure working

EXT_FAIL presents input arcs from *VM_UP*, *VM_DOWN* and *VM_STRTD*. If there were not arc multiplicities, the transition would fire if the respective guard expression was evaluated to true and all input places had tokens. However, the multiplicity of input and output arcs associated with *EXT_FAIL* allows a different behavior. In this case, the transition fires if at least one input place has tokens *((#VM_UP + #VM_DOWN + #VM_STRTD) > 0)* and a dependency is failed (i.e., *#OSPM_UP=0 OR #NAS_NET_UP=0 OR #DC_UP=0*), see Table 4.1.

Whenever an external failure occurs, tokens of input places are instantaneously eliminated and the sum of removed tokens is inserted in *VM_WAIT*. *DC_PART* of different VM availability components are merged if the respective physical machines are located in the same data center. In other words, just one *DC_PART* is represented for each data center. Place *CHC_RES* represents the VMs represents waiting VMs to be instantiated in a data center.

For instance, Figure 4.7 presents the merging of two VM availability components to represent two servers of the same data center. Observe that place *CHC_RES1* represent the VMs that should be started. Therefore, whenever a VM cannot be started in one server, the other server can start this VM if the server is operational.

Figure 4.7: Example Convertion to VM availability component



Source: Made by author.

VM availability component: Metrics.

By adopting this component, users can evaluate the following metrics: (i) availability, (ii) VM utilization, and (iii) capacity oriented availability. Availability (A_v) is given by Equation 4.2, where *#VM_UP* represent the number of tokens in place *VM_UP*, and *REQ_VMS* is the number of VMs required to consider the system operational. Basically, this expression estimates the probability of the system is running, at least, the required VMs (*REQ_VMS*) to consider the system operational.

$$A_v = P\{\#VM\_UP \geq REQ\_VMS\} \tag{4.2}$$

Equation 4.3 presents the expression to evaluate the server utilization (U_t). The expression is given by the ratio of the expected number of running VMs (E{*#VM_UP*}) to the maximum number of VMs that the server can execute (N).

$$U_t = E\{\#VM\_UP\}/N \tag{4.3}$$

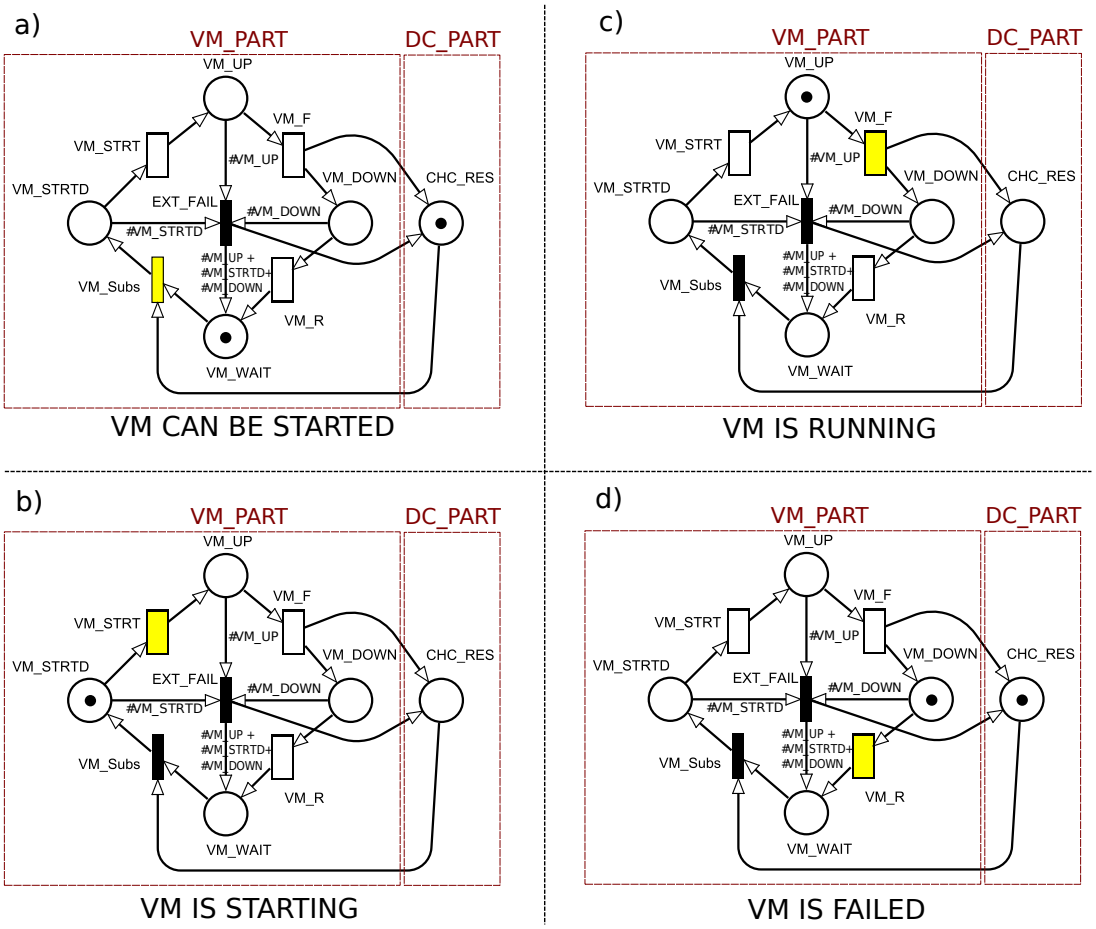
Different from availability that takes into account the probability of the system is up or down, the capacity oriented availability (C_{oa}) evaluates how much service is available to the users (60). The metric is presented in Equation 4.4, where $P\{\#VM\_UP = i\}$ it the probability to have i running VMs, and M corresponds to the number of required VMs. The number of required VMs (M) is represented as the initial marking of place *CHK_RES*.

$$C_{oa} = \sum_{i=1}^{M} (P\{\#VM\_UP = i\} \times i/M) \tag{4.4}$$

Dynamic Behavior of VM availability Component.

Two token flows are presented to represent the dynamic behavior of this component. The

Figure 4.8: Token Flow Availability Component [CHANGE MY NAME] P1



Source: Made by author.

first token flow is presented in Figure 4.8 and considers the situation where a VM performs its life-cycle with no external interruption (e.g., server failures). The second token flow (Figure 4.9) illustrates a VM life-cycle and an external event (e.g., disaster occurrence) interrupts its execution. The dynamic behavior of this component is represented as follows:

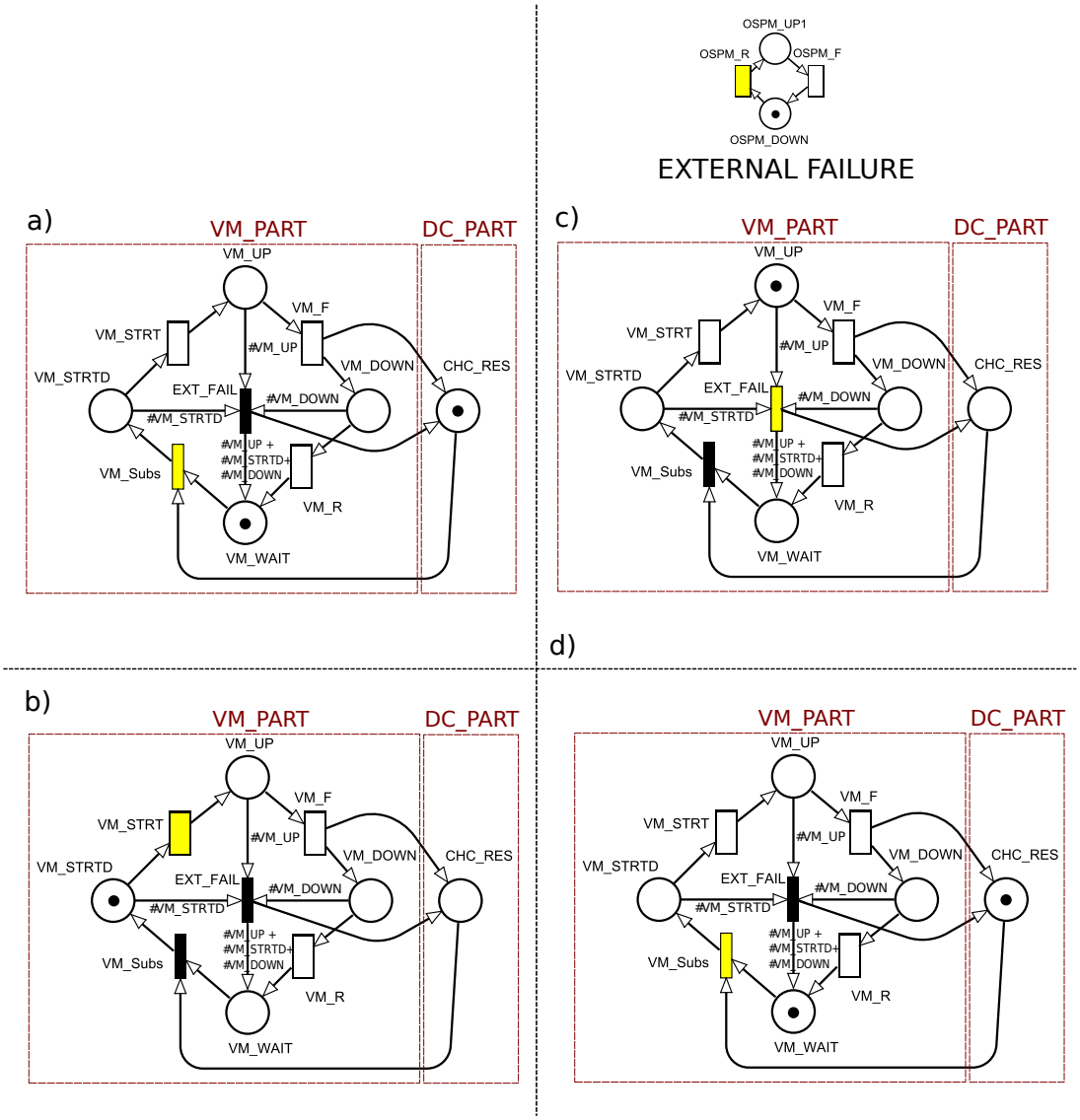
VM life-cycle with no external failure (Figure 4.8).

- **Initial configuration - Figure 4.8(a).** Initially, a VM is required to start (place *CHC_RES* is marked), and this particular server is able to instantiate a VM (*VM_WAIT* has one token). If the underlying infrastructure is operational (*VM_Subs* guard is evaluated as true), a new VM can be started and transition *VM_Subs* fires.
- **VM instantiation - Figure 4.8(b).** In this stage, the VM is instantiating (*VM_STRTD* is marked). When transition *VM_STRT* fires, the VM becomes operational. If the underlying infrastructure fails during the instantiation (e.g., a Disaster occurrence), *EXT_FAIL* fires and the process must be restarted.
- **VM execution - Figure 4.8(c).** In this situation, the VM is running (*VM_UP*) has

one token) and only stops in case of VM error or external failures. In this particular flow, the VM stops working due to an error in the VM itself (*VM_F* fires).

- **VM failed - Figure 4.8(d).** A VM repair activity is conducted in this phase. When *VM_R* fires, the component returns to the initial configuration (Figure 4.8(a)).

Figure 4.9: Token Flow Availability Component P2



Source: Made by author.

VM life-cycle with external failure (Figure 4.9).

- **Initial configuration - Figure 4.9(a).** Like the previous token flow, in the initial configuration of this component, a VM required to start (place *CHC_RES* is marked), and the server is able to start a new VM (*VM_WAIT* has one token).

- **VM instantiation - Figure 4.9(b).** After *VM_Subs* fires, the VM instantiation process starts. In this case, *VM_STRTD* gets a new token, and when *VM_STRT* fires the VM becomes operational.

- **VM execution - Figure 4.9(c).** When the VM is running (*VM_UP* is marked), an external failure happens. In this case, the underlying server is down (*OSPM_DOWN* is marked). In this situation the system returns to the initial condition. It is important to stress that other external event could lead to the VM failure (e.g., a disaster).

- **VM is waiting external repair - Figure 4.9(d).** In this configuration, an external device is failed (i.e., server). Therefore, a new VM can be started only when the underlying infrastructure is operational again. *VM_Subs* guard function enables a new VM to start only if all the related components are operational (See Table 4.1).

Formal Description of VM availability Component.

The SPN $\mathscr{B}_{dp} = (P_{dp}, T_{dp}, I_{dp}, O_{dp}, H_{dp}, \Pi_{dp}, M_{dp}, Atts_{dp})$ represents the VM availability component, in which:

- $P_{dp} = \{VM\_UP, VM\_DOWN, VM\_WAIT, VM\_STRTD, CHC\_RES\}$.

- $T_{dp} = \{VM\_F, VM\_R, VM\_Subs, VM\_STRT, EXT\_FAIL\}$.

- The model's structure can be represented by the matrix $C_{dp} = O_{dp} - I_{dp} =$

$$
\begin{array}{c}
 \\
VM\_UP \\
VM\_DOWN \\
VM\_WAIT \\
VM\_STRTD \\
CHC\_RES
\end{array}
\begin{array}{c}
\begin{array}{ccccc} VM\_F & VM\_R & VM\_Subs & VM\_STRT & EXT\_FAIL \end{array} \\
\left[\begin{array}{ccccc}
-1 & 0 & 0 & 1 & -m_1 \\
1 & -1 & 0 & 0 & -m2 \\
0 & 1 & -1 & 0 & m1+m2+m3 \\
0 & 0 & 1 & -1 & -m3 \\
1 & 0 & -1 & 0 & m1+m3
\end{array}\right]
\end{array}.
$$

  Where m_1, m_2, and m_3 are non negative integers that represent the marking dependent arc multiplicities of input and output arcs of *EXT_FAIL*.

- This component has no inhibitor arcs, i.e. H_{dp} is an empty matrix with five lines and five columns.

- All transitions have the same priority level: $\Pi_{dp} = (1,1,1,1,1)$.

- $M_{dp} = (0,0,m_x,0,0)$, where m_x represents the maximum number of VMs that can be executed at the same time on the underlying physical machine.

- $Atts_{dp} = (Dist_{dp}, W_{dp}, G_{dp}, Pol_{dp}, Con_{dp})^5$ corresponds to attributes for the model transitions:

- $Dist_{dp}$. These values come from high level model.
- W_{dp} corresponds to the weight function. In the case of the transition with exponential distributed firing delay, this weight model the rate λ_t for the transition.
- Pol_{dp} is *preemptive resume* for all transitions.
- Con_{dp} is single server *ss* for all transitions.

This component contains a P-invariant covering all places given by:

$$\mathscr{I}_{(dp)(1)} = \begin{bmatrix} \overset{VM\_UP}{dp_1} & \overset{VM\_DOWN}{dp_1} & \overset{VM\_WAIT}{dp_1} & \overset{VM\_STRTD}{dp_1} & \overset{CHC\_RES}{0} \end{bmatrix}^T.$$

$$\mathscr{I}_{(dp)(2)} = \begin{bmatrix} \overset{VM\_UP}{dp_2} & \overset{VM\_DOWN}{0} & \overset{VM\_WAIT}{0} & \overset{VM\_STRTD}{dp_2} & \overset{CHC\_RES}{dp_2} \end{bmatrix}^T.$$

$$\mathscr{I}_{(dp)(t)} = \begin{bmatrix} \overset{VM\_UP}{dp_2+dp_1} & \overset{VM\_DOWN}{dp_1} & \overset{VM\_WAIT}{dp_1} & \overset{VM\_STRTD}{dp_2+dp_1} & \overset{CHC\_RES}{dp_2} \end{bmatrix}^T.$$

Since, ${\mathscr{I}_{(dp)(t)}}^T \times C_{dp} = 0$ and $\mathscr{I}_{(dp)(t)} > 0$, this block is structurally conservative as well as structurally bounded.

4.3.3 SPN block: VM performance component

VM performance component is adopted in performance models and represent the incoming of VM requests as well as data center and server selection. Figure 4.10 presents the VM performance model, which is composed of three main parts: (i) *VM_PART* that represents the behavior of running VMs on a single machine; (ii) *DC_PART* which expresses the incoming requests to data center; and (iii) *CLOUD_PART* that models the requests generation.

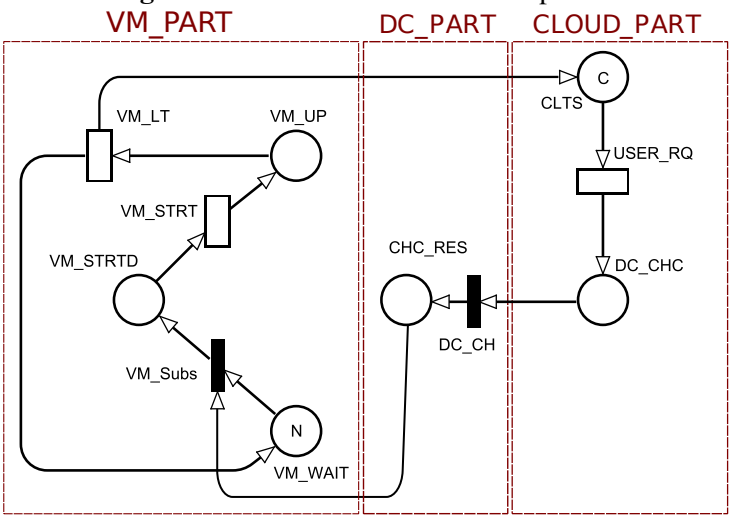
Figure 4.11 presents how performance behavior of VMs can be represented by using a VM performance component. Whenever performance models are created, the failure/repair behavior of VMs and other components are not represented (e.g., generic component for a server). Therefore, just the performance characteristics of VMs are represented.

Table 4.2: Guard Expressions for VM performance component.

Transition	Condition	Description
VM_Subs	no guard	start virtual machine
DC_CH	(*#VM_WAIT*>0)	request acceptance to data center

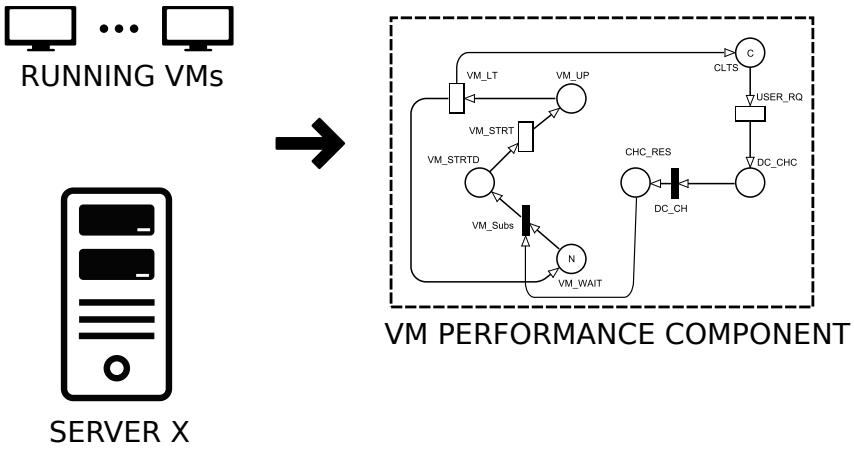
Whenever a VM is requested in *CLOUD_PART*, a data center is selected in *DC_PART*, and a VM is instantiated in *VM_PART* if the server is not full (i.e., *#VM_WAIT* > 0). In

Figure 4.10: VM Performance Component



Source: Made by author.

Figure 4.11: VM performance component representation



Source: Made by author.

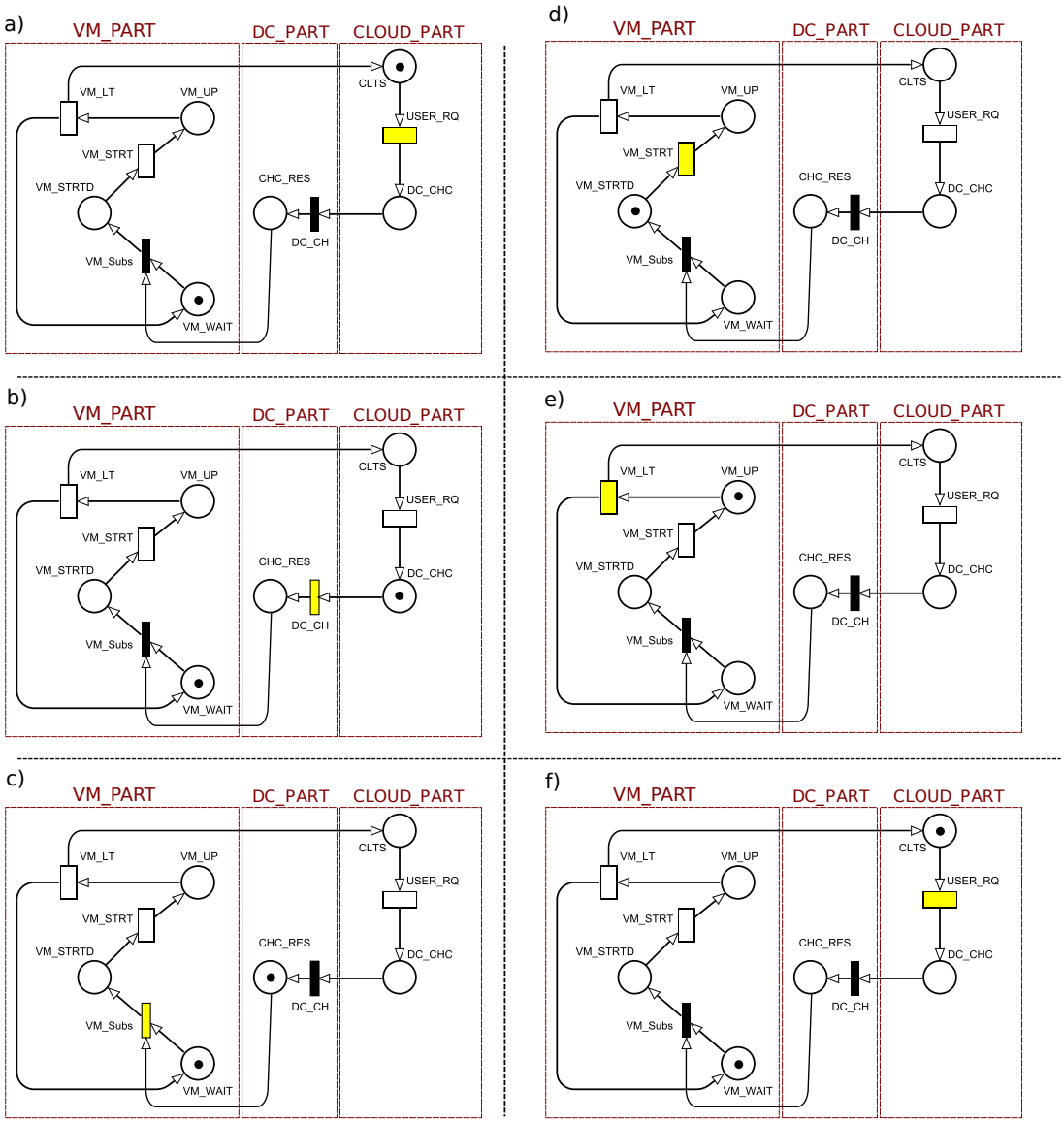
this component, failures and repairs of VMs are not modelled. All the remaining places and transitions present the same semantics and characteristics of VM availability component except *VM_Subs* and *DC_CH*. Table 4.2 details the guard expressions related to these transitions.

For this component, only one *DC_PART* is represented for each data center and just one *CLOUD_PART* is represented for the whole system. *DC_PART* of different server are merged if the machines are in the same data center and *CLOUD_PART* of all system servers are merged.

VM performance component: Metrics.

This component can be adopted to evaluate the following metric: (i) VM Utilization, and (ii) VM execution throughput. The expression shown in Equation 4.3 is utilized to evaluate the VM utilization. The VM execution throughput is given by Equation 4.5, where $E\{\#VM\_UP\}$ represents the expected number of executing VMs, and L_t is the VM execution mean time.

Figure 4.12: Token Flow of Performance Component



Source: Made by author.

$$V_{tp} = E\{\#VM\_UP\} \times (1/L_t) \tag{4.5}$$

Dynamic Behavior of VM Performance component.

The token flow that represents the performance characteristics of VMs in a disaster tolerant cloud system is represented in Figure 4.12. The behavior of this component is described as follows:

- **User request - Figure 4.12(a)**. When *CLTS* has a token, a user request should be performed to the system. The firing of transition *USER_RQ* represents a new VM request that was performed to the cloud system.

- **Data Center Selection - Figure 4.12(b)**. In this phase, the request is passed to one data center of the cloud. As there is just one data center in this configuration, the

request is passed to the current data center. However, if the cloud is composed of several data centers, there will be a *DC_CH* place associated with each data center. In this case, any data center can be selected (i.e., a *DC_CH* transitions will fire) if it is not overloaded (i.e., *VM_WAIT* has tokens).

- **VM Start Checking - Figure 4.12(c)**. Now, a new VM should be started in this data center as *CHC_RES* is marked and the current server has enough resources to start a new VM (*VM_WAIT* has one token). When *VM_Subs* fires, the VM instantiation process begins.

- **VM Instantiation - Figure 4.12(d)**. Just like VM availability component, in this configuration (*VM_STRTD* marked) a new VM is instantiating. When *VM_STRT* fires, the VM becomes fully operational.

- **VM Execution - Figure 4.12(e)**. Finally, in this component, a VM is no longer running when the user decide to stop its execution. Therefore, transition *VM_LT* fires and the component returns to its initial state Figure 4.12(f).

Formal Description of VM Performance component.

The VM performance component is modeled by a SPN $\mathscr{B}_{pf} = (P_{pf}, T_{pf}, I_{pf}, O_{pf}, H_{pf}, \Pi_{pf}, M_{pf}, Atts_{pf})$, in which:

- $P_{pf} = \{$*CLTS, DC_CHC, CHC_RES, WM_WAIT, VM_STRTD, VM_UP*$\}$.

- $T_{pf} = \{$*VM_LT, USER_RQ, DC_CH, VM_SUBS, VM_SRT*$\}$.

- The model's structure can be represented by the matrix $C_{pf} = O_{pf} - I_{pf} =$

$$
\begin{array}{c}
 \\
\textit{CLTS} \\
\textit{DC\_CHC} \\
\textit{CHC\_RES} \\
\textit{WM\_WAIT} \\
\textit{VM\_STRTD} \\
\textit{VM\_UP}
\end{array}
\begin{array}{c}
\begin{array}{ccccc}
\textit{VM\_LT} & \textit{USER\_RQ} & \textit{DC\_CH} & \textit{VM\_SUBS} & \textit{VM\_SRT}
\end{array} \\
\left[\begin{array}{ccccc}
1 & -1 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
1 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 \\
-1 & 0 & 0 & 0 & 1
\end{array}\right]
\end{array}
$$

- This component has no inhibitor arcs, i.e. H_{pf} is an empty matrix with six lines and five columns.

- All transitions have the same priority level. $\Pi_{pf} = (1,1,1,1,1)$.

- $M_{pf} = (0,0,0,m_x,0,0)$, where m_x represent the maximum number of VMs that can be executed at the same time on the underlying physical machine.
- $Atts_{pf} = (Dist_{pf}, W_{pf}, G_{pf}, Pol_{pf}, Con_{pf})$[5] corresponds to attributes for the model transitions:
  - $Dist_{pf}$ These values come from high level model.
  - W_{pf} corresponds to the weight function. In the case of the transition with exponential distributed firing delay, this weight model the rate λ_t for the transition.
  - Pol_{pf} is *preemptive resume* for all transitions.
  - Con_{pf} is single server *ss* for all transitions, except for *VM_LT* that is infinite server *is*.

This component contains a P-invariant covering all places given by:

$$\mathscr{I}_{(pf)(1)} = \begin{bmatrix} \overset{CLTS}{pf_1} & \overset{DC\_CHC}{pf_1} & \overset{CHC\_RES}{pf_1} & \overset{WM\_WAIT}{0} & \overset{VM\_STRTD}{pf_1} & \overset{VM\_UP}{pf_1} \end{bmatrix}^T.$$

$$\mathscr{I}_{(pf)(2)} = \begin{bmatrix} \overset{CLTS}{0} & \overset{DC\_CHC}{0} & \overset{CHC\_RES}{0} & \overset{WM\_WAIT}{pf_2} & \overset{VM\_STRTD}{pf_2} & \overset{VM\_UP}{pf_2} \end{bmatrix}^T.$$

$$\mathscr{I}_{(pf)(t)} = \begin{bmatrix} \overset{CLTS}{pf_1} & \overset{DC\_CHC}{pf_1} & \overset{CHC\_RES}{pf_1} & \overset{WM\_WAIT}{pf_2} & \overset{VM\_STRTD}{pf_1+pf_2} & \overset{VM\_UP}{pf_1+pf_2} \end{bmatrix}^T.$$

Conclusion Since, ${\mathscr{I}_{(pf)(t)}}^T \times C_{pf} = 0$ and $\mathscr{I}_{(pf)(t)} > 0$, this block is structurally conservative as well as structurally bounded.

4.3.4 SPN block: VM performability component

VM performability component is adopted in performability models and represents VM requests on a single physical machine considering failures and repairs on the underlying infrastructure. Whenever a user request is performed, a new VM is started (considering that the infrastructure is operational) and becomes available for some time. If the external infrastructure or the VM fail during the virtual machine execution, the VM should be migrated to another physical machine. If there is no available physical machine in the system, the task is rejected and a new request must be performed. As VM availability component, this component (Figure 4.13) interacts with three generic components: (i) *DC*, (ii) *NAS_NET* and (iii) *OSPM*. Similar to VM performance component, this SPN submodel is composed of three main parts: (i) *VM_PART*, (ii) *DC_PART* and (iii) *CLOUD_PART*.

Figure 4.13: VM performability model

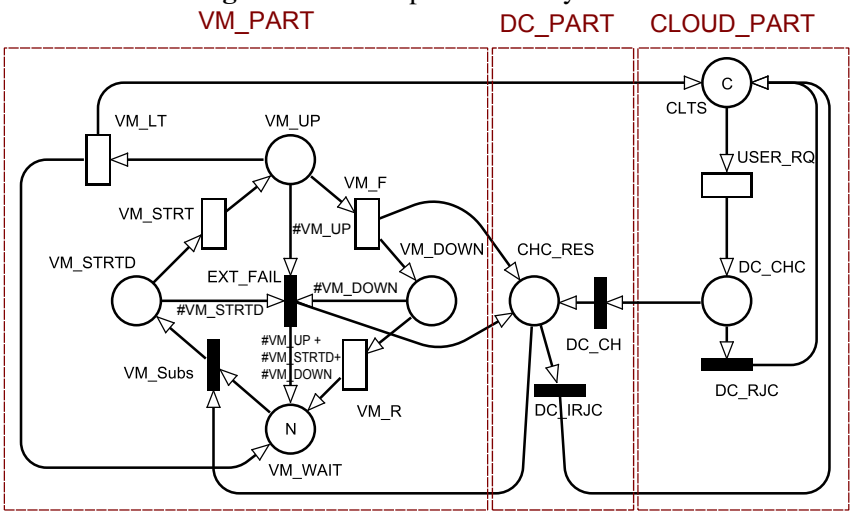
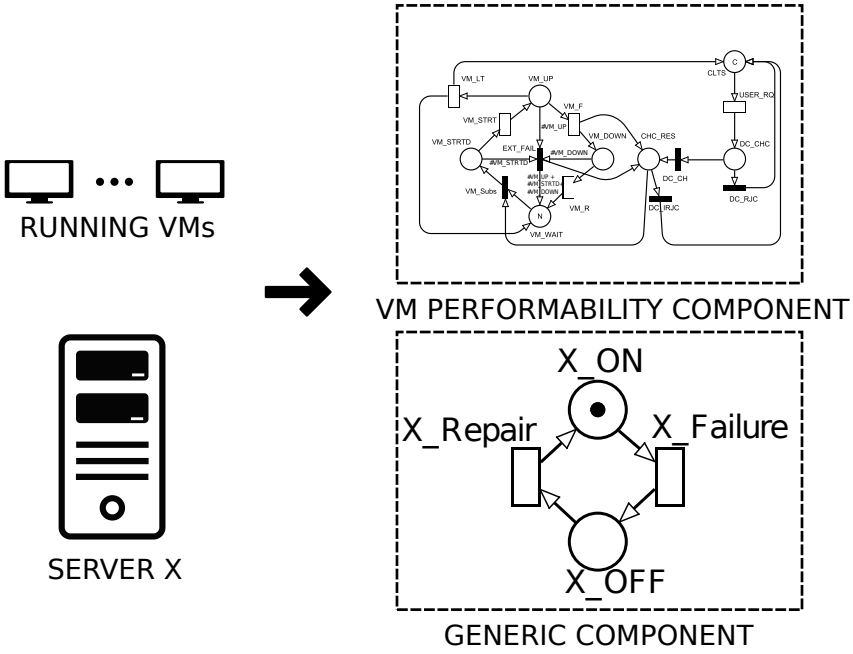


Figure 4.14: Example Convertion to VM performability component



Source: Made by author.

Figure 4.14 presents the representation of servers and VMs by adopting a VM performability component. Observe that this model is a combination of VM performance and dependability components to represent system performability characteristics.

Considering *VM_PART*, places and transitions are analogous to VM availability component, except *VM_LT* that represents the period in which the VM is operational. As the previous submodels, *DC_PART* of different VM performability components are merged if the respective physical machines are located in the same data center. Place *CHC_RES* represents the incoming requests to the current data center. In case of no available machine in the system, the request is cancelled. The task cancellation is represented by the transition *DC_IRJC*. A task is rejected whenever one of the following conditions is satisfied: the network infrastructure is

failed *(#NAS_NET_UP=0)*, a disaster happened *(#DC_UP=0)*, or the respective server cannot instantiate a new VM *(#VM_WAIT=0)*, the underlying physical machine is not operational *(#OSPM_UP=0)*. Table 4.3 presents the guard expressions related to this submodel. *DC_CH* represents the opposite idea, in the sense that a task is received only if the respective infrastructure is working.

CLOUD_PART represents the load generation and request rejection. The *CLOUD_PART* of different VM performability submodels are merged, just one *CLOUD_PART* is represented for the whole system.

Table 4.3: Guard Expressions for VM performability component.

Transition	Condition	Description
DC_IRJC	*(#NAS_NET_UP=0) OR (#DC_UP=0) OR (#VM_WAIT=0) OR (#OSPM_UP=0)*	task rejection
DC_CH	*(#NAS_NET_UP>0) AND (#DC_UP>0) AND (#VM_WAIT>0) AND (#OSPM_UP>0)*	request acceptance to data center
DC_RJC	*(#NAS_NET_UP=0) OR (#DC_UP=0) OR (#VM_WAIT=0) OR (#OSPM_UP=0)*	task rejection

CLTS and *DC_CH* model clients that are about to perform requests and the requests that entered into the system, respectively. It is assumed that each client requests a single VM. The transition *USER_RQ* means the request process. Finally, *DC_RJC* denotes the request rejection when all data centers are unavailable. A data center is unavailable if the underlying infrastructure is broken or the servers are full. It is important to stress that the guard expressions of *DC_IRJC*, *DC_CH* and *DC_RJC* can vary depending on the number of physical machines and the number of data centers. In this case, we assume just one physical machine in one data center. However, this approach is generic enough to consider several machines in multiple data centers.

VM performability component: Metrics.

The following metric can be assessed by adopting this component: (i) capacity oriented availability, (ii) VM utilization, (iii) VM execution throughput, (iv) probability to finish a request. The metrics i-iii can be evaluated by using the Equations 4.4, 4.3, and 4.5, respectively.

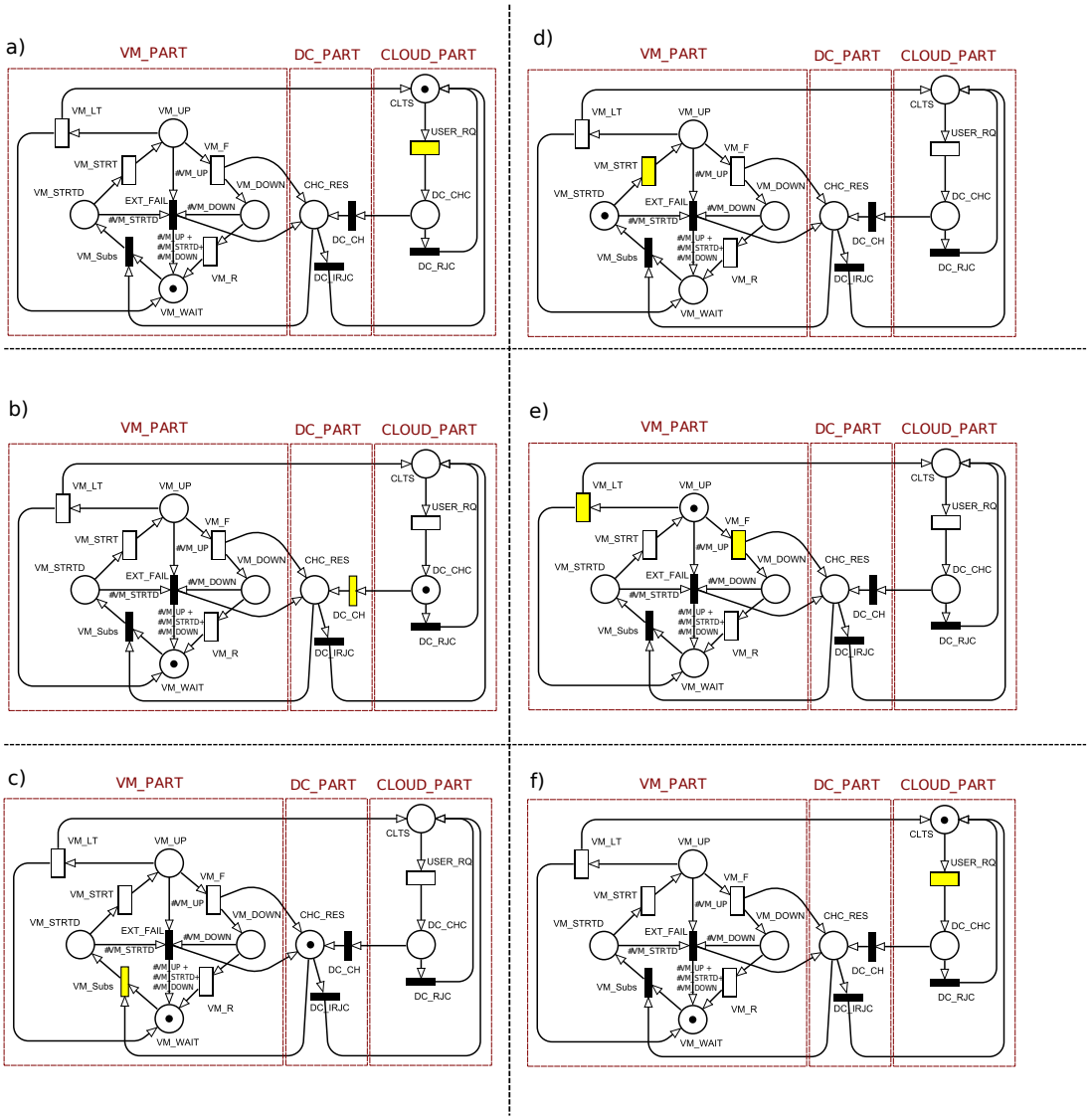
$$P_{fs} = \frac{E\{\#VM\_UP\} \times (1/L_t)}{P\{\#CLTS\ > 0\} \times (1/ReqTime)} \tag{4.6}$$

The probability to finish a request is given by Equation 4.6, where $E\{\#VM\_UP\}$ represents the expected number of executing VMs, L_t is the VM execution mean time, $P\{\#CLTS\ > 0\}$ is the probability of having VM requests, and *ReqTime* is the mean time between VM requests.

Dynamic behavior of VM performability component.

Figure 4.15 presents the token flow that represents the performance and availability

Figure 4.15: Token Flow of Performability Component



Source: Made by author.

characteristics of VMs in a disaster tolerant cloud computing system. The steps presented in Figures 4.15(a)-(e) are analogous to the steps presented in Figure 4.12(a)-(e). The main difference between the presented flows is that VM performability component allows component's failures/repairs, and external events (e.g., disasters) may affect VM behavior.

Formal Description of VM performability component.

The VM performability component is modeled by a SPN $\mathscr{B}_{pb} = (P_{pb}, T_{pb}, I_{pb}, O_{pb}, H_{pb}, \Pi_{pb}, M_{pb}, Atts_{pb})$, in which:

- $P_{pb} = \{CLTS, DC\_CHC, CHC\_RES, VM\_DOWN, VM\_WAIT, VM\_STRTD, VM\_UP\}$.
- $T_{pb} = \{USER\_RQ, DC\_RJC, DC\_CH, DC\_IRJC, VM\_R, VM\_SUBS, VM\_SRT,$

VM_LT, VM_F, EXT_FAIL}.

- The model's structure can be represented by the matrix $C_{pb} = O_{pb} - I_{pb} =$

	USER_RQ	*DC_RJC*	*DC_CH*	*DC_IRJC*	*VM_R*	*VM_SUBS*	*VM_STRT*	*VM_LT*	*VM_F*	*EXT_FAIL*
CLTS	-1	1	0	1	0	0	0	1	0	0
DC_CHC	1	-1	-1	0	0	0	0	0	0	0
CHC_RES	0	0	1	-1	0	-1	0	0	1	m_1+m_3
VM_DOWN	0	0	0	0	-1	0	0	0	1	$-m_2$
WM_WAIT	0	0	0	0	1	-1	0	1	0	$m1+m2+m3$
VM_STRTD	0	0	0	0	0	1	-1	0	0	$-m_3$
VM_UP	0	0	0	0	0	0	1	-1	-1	$-m_1$

  Where m_1, m_2, and m_3 are non negative integers that represent the marking dependent arc multiplicities indicated in C_{dp}.

- This component has no inhibitor arcs, i.e. H_{pb} is an empty matrix with seven lines and ten columns.

- All transitions have the same priority level. $\Pi_{pb} = (1,1,1,1,1,1,1,1,1,1)$.

- $M_{pb} = (0,0,0,0,m_x,0,0)$, where m_x represent the maximum number of VMs that can be executed at the same time on the underlying physical machine.

- $Atts_{pb} = (Dist_{pb}, W_{pb}, G_{pb}, Pol_{pb}, Con_{pb})^{10}$ corresponds to attributes for the model transitions:

  - $Dist_{pb}$ These values come from high level model.
  - W_{pb} corresponds to the weight function. In the case of the transition with exponential distributed firing delay, this weight model the rate λ_t for the transition.
  - Pol_{pb} is *preemptive resume* for all transitions.
  - Con_{pb} is single server *ss* for all transitions, except for *VM_LT* that is infinite server *is*.

This component contains a P-invariant covering all places given by:

$$\mathscr{I}_{(pb)(1)} = \begin{matrix} \textit{CLTS} & \textit{DC\_CHC} & \textit{CHC\_RES} & \textit{VM\_DOWN} & \textit{WM\_WAIT} & \textit{VM\_STRTD} & \textit{VM\_UP} \end{matrix}$$
$$\left[\begin{array}{ccccccc} 0 & 0 & 0 & pb_1 & pb_1 & pb_1 & pb_1 \end{array}\right]^T.$$

$$\mathscr{I}_{(pb)(2)} = \begin{matrix} \textit{CLTS} & \textit{DC\_CHC} & \textit{CHC\_RES} & \textit{VM\_DOWN} & \textit{WM\_WAIT} & \textit{VM\_STRTD} & \textit{VM\_UP} \end{matrix}$$
$$\left[\begin{array}{ccccccc} pb_2 & pb_2 & pb_2 & 0 & 0 & pb_2 & pb_2 \end{array}\right]^T.$$

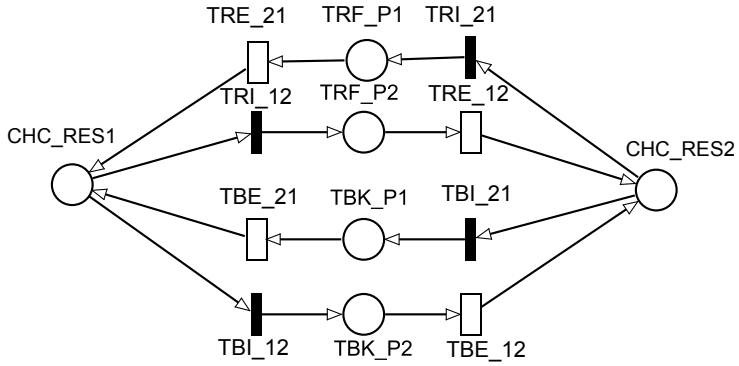
$$\mathscr{I}_{(pb)(t)} = \begin{matrix} \textit{CLTS} & \textit{DC\_CHC} & \textit{CHC\_RES} & \textit{VM\_DOWN} & \textit{WM\_WAIT} & \textit{VM\_STRTD} & \textit{VM\_UP} \end{matrix}$$
$$\left[\begin{array}{ccccccc} pb_2 & pb_2 & pb_2 & pb_1 & pb_1 & pb_1+pb_2 & pb_1+pb_2 \end{array}\right]^T.$$

Since, ${\mathscr{I}_{(pb)(t)}}^T \times C_{pb} = 0$ and $\mathscr{I}_{(pb)(t)} > 0$, this block is structurally conservative as well as structurally bounded.

4.3.5 SPN block: VM transmission component

A VM should migrate to another data center whenever the current data center is overloaded or the underlying infrastructure is failed. Moreover, Backup Server is responsible for transmitting VM images in case of disaster, network error and server failure. VM transmission component (Figure 4.16) has four transitions that represent the VM data transfer and four immediate transitions that depict the enabling of VM migrations. *TRE_21* represents the VM transmission between two generics data centers, in this case, Data Center 2 and Data Center 1; *TRE_12* characterizes the migration from Data Center 1 to Data Center 2; *TBE_21* corresponds to data transfer from Backup Server to Data Center 1, and *TBE_12* characterizes the data transfer from Backup Server to Data Center 2.

Figure 4.16: VM Transmission component



Source: Made by author.

Table 4.4: Guard Expressions for VM transmission component.

Transition	Condition
TRI_12	(#VM_WAIT1=0 OR #OSPM_UP1=0) AND (#VM_WAIT2=0 OR #OSPM_UP2=0) AND NOT(((#VM_WAIT3=0 OR #OSPM_UP3=0) AND (#VM_WAIT4=0 OR #OSPM_UP4=0)) OR #NAS_NET_UP2=0 OR #DC_UP2=0)
TRI_21	(#VM_WAIT3=0 OR #OSPM_UP3=0) AND (#VM_WAIT4=0 OR #OSPM_UP4=0) AND NOT(((#VM_WAIT1=0 OR #OSPM_UP1=0) AND (#VM_WAIT2=0 OR #OSPM_UP2=0)) OR #NAS_NET_UP1=0 OR #DC_UP1=0)
TBI_12	(#BKP_UP=1 AND #NAS_NET_UP1=0 OR #DC_UP1=0) AND NOT(((#VM_WAIT3=0 OR #OSPM_UP3=0) AND (#VM_WAIT4=0 OR #OSPM_UP4=0)) OR #NAS_NET_UP2=0 OR #DC_UP2=0)
TBI_21	(#BKP_UP=1 AND #NAS_NET_UP2=0 OR #DC_UP2=0) AND NOT(((#VM_WAIT1=0 OR #OSPM_UP1=0) AND (#VM_WAIT2=0 OR #OSPM_UP2=0)) OR #NAS_NET_UP1=0 OR #DC_UP1=0)

Table 4.4 presents the guard expressions of VM transmission component. It is assumed that Data Center 1 contains the physical machines *OSPM_UP 1* and *OSPM_UP2*, and Data Center 2 includes *OSPM_UP3* and *OSPM_UP4*. *Mean Time to Transmit (MTT)* represents the mean time to transfer one virtual machine from one location to another. *MTT* depends on the physical link speed, the distance between the data centers and the VM size (75). In this block, there are three *MTTs*: mean time to transmit a VM from the data center to another (*MTT_DCS*) and the mean times to transfer the VM image from Backup Server to Data Centers 1 and 2 (*MTT_BK1 and MTT_BK2*). The *MTT_DCS* parameter is associated to transitions *TRE_12* and *TRE_21*, while *MTT_BK1* and *MTT_BK2* are related to TBK_21 and TBK_12, respectively.

TRE_12 guard expression is set true whenever both physical machines of Data Center 1 are unavailable *((#VM_WAIT1=0 OR #OSPM_UP1=0) AND (#VM_WAIT2=0 OR #OSPM_UP2 =0))*, the machines of Data Center 2 are not failed *NOT((#VM_WAIT3 =0 OR #OSPM_UP3=0) AND (#VM_WAIT4 =0 OR #OSPM_UP4=0))*, the network is operational *NOT(#NAS_NET_UP2 =0)* and a disaster did not occur *NOT(#DC_UP2=0)*. The guard expression of *TBK_12* is evaluated as true whenever Backup Server is operational *(#BKP_UP =1)* and Data Center 1 is not accessible due to disasters *(#DC_UP1 =0)* or networking issues *(#NAS_NET_UP1=0)*. Additionally, the Data Center 2 must be operational i.e., *NOT(((#VM_WAIT3=0 OR #OSPM_UP3=0) AND (#VM_WAIT4=0 OR #OSPM_UP4=0)) OR #NAS_NET_UP2=0 OR #DC_UP2=0)*. The guard expressions related to *TRE_21* and *TBK_21* are analogous to *TRE_12* and *TBK_12*.

VM Transmission component: Metrics.

Two metrics can be accessed by using the VM Transmission component: (i) VM data center transmission throughput (ii) VM backup server transmission throughput. Equation 4.7 presents the VM data center transmission throughput (T_{p_dc}). This metric corresponds to the throughput of transmitted VMs due to failures or overloads in a given data center. In this expression, $E\{\#TRF\_P2\}$ corresponds to the expected number of transmitted VMs, and MTT_DC represents the mean time to transmit a VM from Data center 1 to Data center 2.

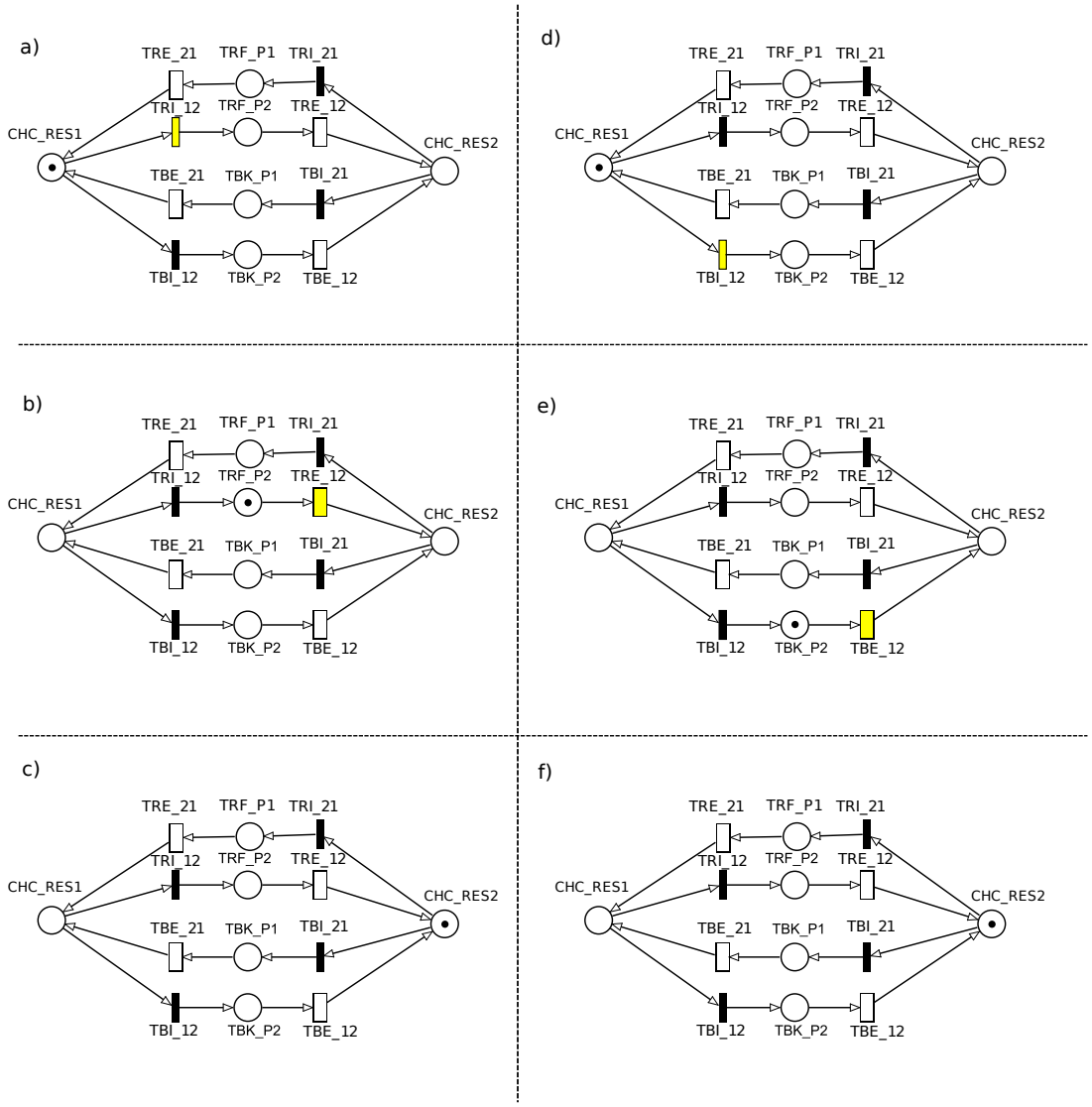
$$T_{p\_dc} = E\{\#TRF\_P2\} \times (1/MTT\_DC) \tag{4.7}$$

Equation 4.8 presents the VM backup server transmission throughput (T_{p_bs}). This metric is analogous to (T_{p_dc}), in which $E\{\#TBK\_P2\}$ corresponds to the expected number of transmitted VMs, and MTT_BS is the mean time to transmit a VM from backup server to Data center 2.

$$T_{p\_bs} = E\{\#TBK\_P2\} \times (1/MTT\_BS) \tag{4.8}$$

Dynamic Behavior of VM Transmission component.

Figure 4.17: Token Flow of Transmission Component



Source: Made by author.

The token flow that represents the VM transmission between cloud systems in a disaster tolerant cloud environment is represented in Figure 4.17. This token flow represents two situations: (i) all servers of a given data center are failed or overloaded (Figures 4.12(a)-(c)), and (ii) a disaster happened to a data center or it is inaccessible (Figures 4.12(d)-(f)). In the first situation, VMs are transmitted form Data center 1 to Data center 2. Considering the second scenario, VMs are transmitted from Backup Server to Data center 2. The behavior of this component is described as follows:

- **Servers of Data center 1 are overloaded or failed - Figure 4.17(a)**. In this situation, Data center 1 has no resources to run virtual machines. Therefore, VMs of Data center 1 (*CHC_RES1* tokens) are transmitted to Data center 2 (if it is operational). *TRI_12* guard function (see Table 4.4) ensures the VM transmission process only starts if Data center 2 is operational and has enough resources to start new VMs.

After *TRI_12* firing, the VM transmission should start.

- **VM transmission - Figure 4.17(b)**. In this phase, VMs transmitted from Data center 1 to Data center 2 are represented as *TRE_12* firings. As long as VMs are transmitted from Data center 1 to Data center 2, the VM instantiation process can be started on Data center 2 (Figure 4.17(c)).
- **VMs can be started on Data center 2 - Figure 4.17(c).** Transmitted VMs from Data center 1 can be started on Data center 2. Observe that, place *CHC_RES2* corresponds to an instance of *CHC_RES* place of VM availability, performance, or performability components. This place represents VM instance requests for a given data center.
- **A disaster happened to Data center 1 or it is inaccessible - Figure 4.17(d).** This picture represents the second way to utilize a VM transmission component. In this case, Data center 1 is not operational due to a disaster or network failure. In this case, VM backups should be transmitted from backup server to Data center 2 (if it is operational). Similar to Figure 4.17(b), a VM can only be sent to Data center 2 if this data center is able to receive VM requests. This behavior is represented by *TBI_12* guard function (see Table 4.4).
- **VMs can be started on Data center 2 - Figure 4.17(e)**. This phase is analogous to the presented in Figure 4.17(c).

Formal Description of VM transmission component.

The VM transmission component is modeled by a SPN $\mathscr{B}_{tr} = (P_{tr}, T_{tr}, I_{tr}, O_{tr}, H_{tr}, \Pi_{tr}, M_{tr}, Atts_{tr})$, in which:

- $P_{tr} = \{$*CHC_RES1, TRF_P1, TRF_P2, TBK_P1, TBK_P2, CHC_RES2*$\}$.
- $T_{tr} = \{$*TRE_21, TRI_21, TRE_12, TRI_12, TBE_21, TBI_21, TBE_12, TBI_12*$\}$.
- The model's structure can be represented by the matrix $C_{tr} = O_{tr} - I_{tr} =$

	TRE_21	*TRI_21*	*TRE_12*	*TRI_12*	*TBE_21*	*TBI_21*	*TBE_12*	*TBI_12*
CHC_RES1	1	0	0	−1	1	0	0	−1
TRF_P1	−1	1	0	0	0	0	0	0
TRF_P2	0	0	−1	1	0	0	0	0
TBK_P1	0	0	0	0	−1	1	0	0
TBK_P2	0	0	0	0	0	0	−1	1
CHC_RES2	0	−1	1	0	0	−1	1	0

- This component has no inhibitor arcs, i.e. H_{tr} is an empty matrix with six lines and eight columns.
- All transitions have the same priority level. $\Pi_{tr} = (1,1,1,1,1,1,1,1)$.
- $M_{tr} = (0,0,0,0,0,0)$.
- $Atts_{tr} = (Dist_{tr}, W_{tr}, G_{tr}, Pol_{tr}, Con_{tr})^8$ corresponds to attributes for the model transitions:
  - $Dist_{tr}$ These values come from high level model.
  - W_{tr} corresponds to the weight function. In the case of the transition with exponential distributed firing delay, this weight model the rate λ_t for the transition.
  - Pol_{tr} is *preemptive resume* for all transitions.
  - Con_{tr} is single server *ss* for all transitions.

This component contains a P-invariant covering all places given by:

$$\mathscr{I}_{(tr)} = \begin{bmatrix} \overset{CHC\_RES1}{tr} & \overset{TRF\_P1}{tr} & \overset{TRF\_P2}{tr} & \overset{TBK\_P1}{tr} & \overset{TBK\_P2}{tr} & \overset{CHC\_RES2}{tr} \end{bmatrix}^T.$$

Since, ${\mathscr{I}_{(tr)}}^T \times C_{tr} = 0$ and $\mathscr{I}_{(tr)} > 0$, this block is structurally conservative as well as structurally bounded.

4.4 Survivability Models

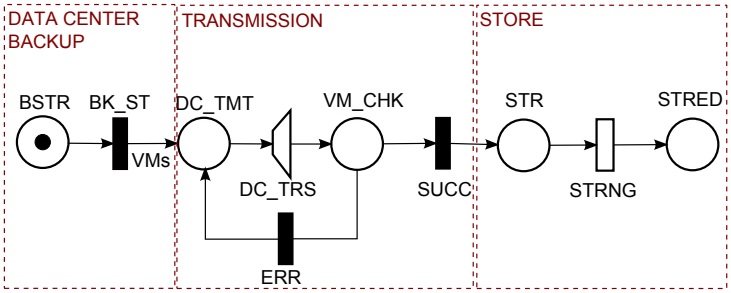
This section presents the adopted models to evaluate system survivability. The proposed approach (Figure 4.1) adopts SPN models for estimating survivability parameters in IaaS clouds. Although this work is focused on cloud computing systems, the approach is generic enough to be applied for other disaster recovery systems. Two models are proposed to evaluate system survivability. The first model (RPO Evaluation model) estimates the probability of finishing a backup at RPO time. The second survivability models evaluates the probability to recover a disaster tolerant IaaS cloud at RTO time.

It is important to emphasize that the performance, dependability and performability models are not combined with the survivability models (they represent distinct models). In other words, these models are evaluated in different times of cloud system assessment. Only performance, availability performability models are created by joining basic building blocks to create the final models. The survivability models do not have this characteristic in the sense that they always have the same structure.

4.4.1 RPO Evaluation Model

The RPO evaluation model is presented in Figure 4.20. It represents the backup process in which a data center transmits VM images to BS. This model is composed of three main sub-models: (i) Data Center Backup, (ii) Transmission and (iii) Store. The first sub-model represents the start of backup operation in a given data center. A token in *BSTR* means that the backup process should start and the firing of *BK_ST* leads to the creation of new tokens (VMs) representing new VM images to be backed up. The new tokens are stored in *DC_TMT*.

Figure 4.18: RPO Model



Source: Made by author.

Regarding Transmission submodel, *DC_TRS* represents the transmission of VMs from the data center to BS. It is important to stress that the moment matching approach (Section 3.4.2.4) can be adopted to represent expolinomial distributions for this transition. Once the VMs are transmitted, the data integrity of each VM is checked (*VM_CHK*). If the process presents errors, the process is restarted (*ERR*). In case of correct transmission, the VMs should be stored/replicated (Store submodel). Finally, the Store submodel represents the storing/replication process of the transmitted VMs. It is composed of two places, one that represents the VM images that are about to be stored (*STR*) and another one to model the saved images (*STRED*). The transition *STRNG* models the store/replication process. If the BS has not replication mechanisms, *SUCC* may be connected directly to *STRED*. In this case, STR and STRNG must be discarded.

In order to perform a disaster recovery evaluation, the following parameters are collected from the high level model: (i) the number of VMs must be periodically backed up, (ii) the backup period, (iii) the number of VMs that should be recovered to restore the service and (iv) the data center transmission characteristics. The system is evaluated using transient evaluation, which is adopted to observe the behavior of the disaster recovery mechanisms along the time (26). The system can be evaluated taking into account backup and recovery characteristics as detailed as follows.

Recovery Point Evaluation. To evaluate the system survivability in terms of RPO, a transient evaluation must be performed adopting the metric $P\{\#STRED = VMs\}$ in the time $RPO - B_p$. In other words, we are interested in evaluate the probability of finish the backup process in a specific time ($RPO - B_p$).

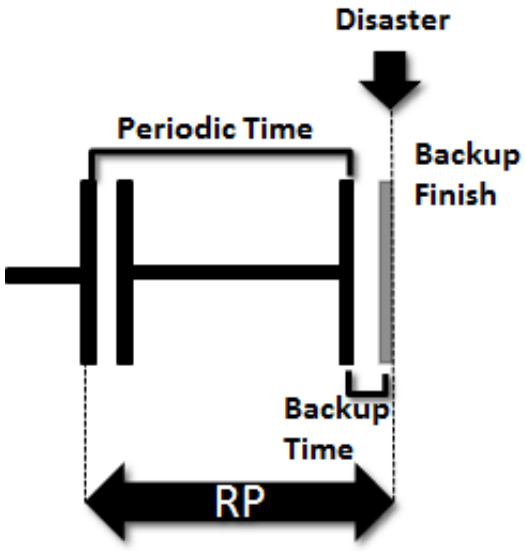
In this study, we are interested in evaluating the recovery point considering the worst-case

scenario (Figure 4.19). According to (76), this situation happens when the disaster occurs during the backup process. Observe that, in this case, the Recovery Point (R_p) is equal to the sum of the Backup Period (B_p) and the actual Backup Time (B_t). Consequently, the probability of meeting the RPO (P_{RPO}) in the worst-case scenario is given by:

$$P_{RPO} = P\{R_p \leq RPO\} = P\{B_t \leq RPO - B_p\}.$$

As B_p and the RPO are project decision parameters, the cloud designer must evaluate the behavior of the backup process to check the P_{RPO} metric. If the assessed probability is less than the user defined level, the system is not survival.

Figure 4.19: RPO worst case scenario.



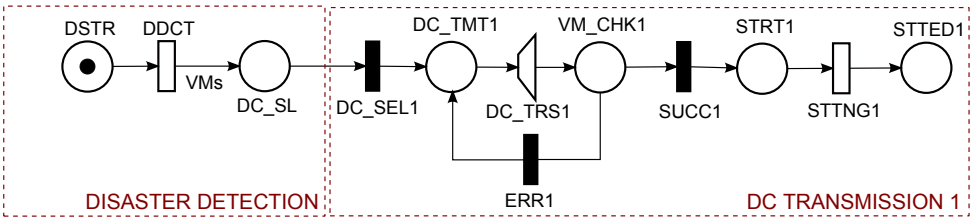
Source: Made by author.

4.4.2 RTO Evaluation Model

Figure 4.20 shows the RTO evaluation model which represents the recovery process immediately after the disaster occurrence. The model is composed of two basic submodels, one representing the disaster detection and other modeling the transmission of VM images to operational data centers. The first submodel is composed of a place that represents the start of recovery process (*DSTR*), a transition which models the disaster detection (*DDCT*) and a place that denotes the data center selection (*DC_SL*). The number of VM images that will be recovered is represented by VMs (arc multiplicity from *DDCT* to *DC_SL*). The other components of DC Transmission block are analogous to the components of RPO model (Figure 4.20) and will not be explained in details. The difference is that the three last components (STRT1, STTNG1 and STTED1) represents virtual machine instantiation instead of data store.

Recovery Time Evaluation. The process of checking the survivability in terms of RTO is similar to the recovery point evaluation. While the last considers the worst-case scenario, the

Figure 4.20: RTO Model



Source: Made by author.

recovery time evaluation is calculated directly. The probability of the recovery time meet the objective (P_{RTO}) is calculated as follows:

$$P_{RTO} = P\{R_t \leq RTO\}.$$

Where R_t denotes the recovery time. Similarly to RPO evaluation model, a transient evaluation should be conducted to assess the probability to recover the system respecting the RTO limit. The observed metric is *P{#STTED1 = VMs}* in time RTO. If the evaluated probability is less than the requirement, the system is not survival.

4.5 Final Remarks

This chapter presented the evaluation process and the models adopted in this work. Initially, the evaluation process was presented, and an example was provided to illustrate the proposed approach. Afterward, The high-level model was detailed. By using high-level models to represent the IaaS disaster tolerant clouds, users can describe precisely the system characteristics and interactions. Then, performance and availability models were described as well as its features. Finally, the survivability models were presented.

5

Mapping High-Level Models to Evaluation Models

This chapter describes the formal composition rules adopted in the proposed modeling approach for creating the final SPN model as well as model composition example. Initially, the mapping algorithm that translates HLM into the final SPN evaluation is presented. Next, modeling examples are given and, the adopted composition rules are demonstrated. Finally, The properties of the proposed operators are proved.

5.1 Mapping Algorithms

The proposed framework has two algorithms for SPN model generation, one for performability and availability models and other for performance models. As the survivability models do not adopt the composition of SPN components to create the final SPN model, there is no translation algorithm applied to these models. The presented algorithms are responsible for translating the high-level models into SPN evaluation models. The first algorithm translates high-level models into availability, and performability models, whereas the second generates performance models from high-level models.

Figure 5.1 presents the adopted algorithm for performability and availability SPN model generation, which creates a SPN model based on high level IaaS model. Firstly, a SPN model with no places and transitions is created (line 3). The result model is *incremented* with new places and transitions from new SPN blocks (e.g., increment function in line 5). The increment function corresponds to the net union operator, which merges two SPNs into a new SPN (Section 5.2.2). Let $\mathscr{N}_1$ and $\mathscr{N}_2$ be two SPNs, then $\mathscr{N}_1.increment(\mathscr{N}_2)$ corresponds to the net union of $\mathscr{N}_1$ and $\mathscr{N}_2$ and the result is saved in $\mathscr{N}_1$. The composition of all new SPN components is returned in line 19.

Function *getBackupServer* (line 4) returns the backup server element (bs). Next, the SPN model is incremented with a new generic component for representing the backup server. Function *parseGenericComponent* (line 5) creates a generic component. The first and second arguments ($T_{di}(bs)$, $T_{re}(bs)$) respectively represent probability distribution functions associated with failure and repair transitions of resulted SPN component.

For each data center element (d), a generic component is created for representing disaster and recovery events (line 7). A new loop is created (line 8) for representing the creation of SPN generic components for representing operating system, hardware and VM behavior of each server (lines 9 to 11). *parseVmBehaviorComponent* receives the parameters for creating VM

Figure 5.1: Performability or Availability SPN model generation Algorithm

```
01 SPN parseModel(G =< F_lt, T_di, T_re, MTT, C_VM, T_req, L_t, S_v >)
02 {
03   SPN result = EMPTY_SPN;
04   bs = getBackupServer(F_lt);
05   result.increment(parseGenericComponent(T_di(bs), T_re(bs)));
06   for each(d = < P_d, C_d > ∈ D){
07     result.increment(parseGenericComponent(T_di(d), T_re(d)));
08     for each(pm = < V_p, S_p, os, hw, m > ∈ P_d){
09       result.increment(parseGenericComponent(T_fr(os), T_rp(os)));
10       result.increment(parseGenericComponent(T_fr(hw), T_rp(hw)));
11       result.increment(parseVmBehaviorComponent(V_p, S_p, m, C_VM, T_req, L_t));
12     }
13     networkDependabilityParams := RBDEvaluation(C_d);
14     result.increment(parseGenericComponent(networkDependabilityParams));
15   }
16   for each ((f1, f2) | f1, f2 ∈ F_lt and f1 ≠ f2){
17     result.increment(parseDataCenterTransmission(f1, f2, MTT));
18   }
19  return result;
20 }
```

Source: Made by author.

availability or VM performability components. If the parameters C_{VM}, T_{req}, L_t are not assigned, a VM availability component is created. Otherwise, the function creates a VM performability component. Finally, a new transmission component for each pair of facilities (line 17). The MTT between facilities is updated and the result is incremented in the model.

Figure 5.2 shows the SPN generation algorithm for performance models. The algorithm is a shortened version of the performability/availability algorithm that creates VM performance components for all servers in the model (line 6). Only VM performance components are modeled in this algorithm as failures/repairs are not represented in performance models.

5.1.1 Mapping HLM to SPN: Example

In this section, we present an example that translates a high level model G_1 into a SPN evaluation model. Figure 5.3 presents the cloud system to be represented as a high level model and then translated to a SPN model. The high level model related to this example is $G_1 = (F_{lt}, T_{di}, T_{re}, MTT, C_{VM}, T_{req}, L_t, S_v)$ where:

- $F_{lt} = \{DC1, DC2, BS\}$ is a finite set of facilities. $DC1$ and $DC2$ are two data centers and BS represents the backup server.

- T_{di} means the disaster occurrence function, such that $T_{di}(DC1) = D_{D1}$, $T_{di}(DC2) =$

Figure 5.2: Performance SPN model generation algorithm

```
01 SPN parsePerformanceModel(G =< F_lt, T_di, T_re, MTT, C_VM, T_req, L_t, S_v >)
02 {
03   SPN result = EMPTY_SPN;
04   for each(d = < P_d, C_d > ∈ D){
05     for each(pm = < V_p, S_p, os, hw, m > ∈ P_d){
06       result.increment(parseVmBehaviorComponent(V_p, S_p, m, C_VM, T_req, L_t));
07     }
08   }
09  return result;
10 }
```

Source: Made by author.

D_{D2} and $T_{di}(BS) = D_{BS}$. D_{D1}, D_{D2} and D_{BS} represent the disaster probability distribution function related to $DC1$, $DC2$ and BS.

- T_{re} represents the disaster recovery function, where $T_{re}(DC1) = R_{D1}$, $T_{re}(DC2) = R_{D2}$ and $T_{re}(BS) = R_{BS}$. R_{D1}, R_{D2} and R_{BS} represent the recovery probability distribution function related to $DC1$, $DC2$ and BS.

- MTT denotes the VM transmission function. $MTT(DC1, DC2) = T_{12}$, $MTT(BS, DC1) = T_{B1}$, $MTT(BS, DC2) = T_{B2}$. T_{12}, T_{B1} and T_{B2} represent probability distribution function related to transmitting VMs between DC1 and DC2; BS and DC1; and BS and DC2.

- $C_{VM} = C$ means the maximum number of requested VMs at a given time.

- $T_{req} = Req$ corresponds to the request time probability function.

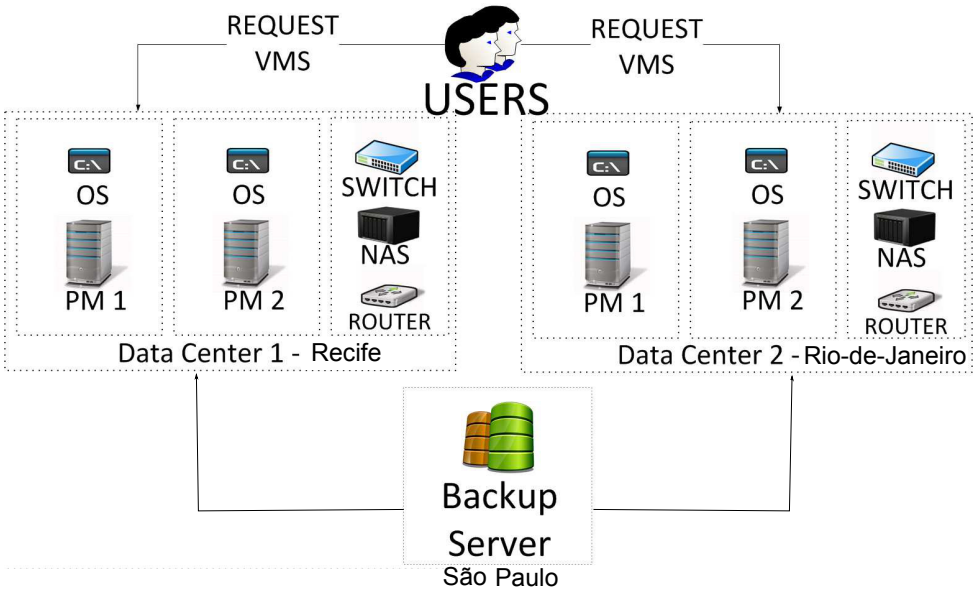
- $L_t = Lt$ is the VM life-time probability function that corresponds to a PDF that relates each VM duration time with a probability.

- $S_v = (v_{bd}, b_{pd}, r_{vd})$ corresponds to survivability parameters, where v_{bd} is the number of VMs must be periodically backed up. b_{pd} represents the backup period and r_{vd} is the number of VMs that should be recovered to restore the service.

$DC1 = (PM1, PM2, NAS1, ROUT1, SWT1)$ is composed of two physical machines $PM1$ and $PM2$. $NAS1, ROUT1, SWT1$ are network components and denote the NAS, router and switch related to $DC1$. $DC2 = (PM3, PM4, NAS2, ROUT2, SWT2)$ has an analogous structure of $DC1$ and will not be explained in details.

$PM1 = (\emptyset, S_{p1}, os1, hw1, N)$ is the tuple that represents the components associated with PM1. This physical machine has no VMs started at system startup. S_{p1} denotes the virtual machine set up time probability distribution function. $os1$ and $hw1$ represent the PM1's software

Figure 5.3: Motivational Architecture.

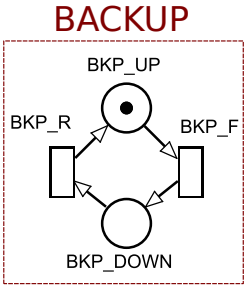


Source: Made by author.

and hardware. N is maximum number of VMs that PM1 can execute. The others physical machines of this system have analogous structures. The following lines presents a step-by-step demonstration to explain how the final SPN is obtained.

Step 1: Backup Server Creation.

Figure 5.4: Step 1: SPN Generation Example



Source: Made by author.

This step concerns the creation of a generic component to represent failure/repair behavior of backup server. Line 4 of *parseModel* algorithm (Figure 5.1) is responsible for obtaining the HLM element that represents the backup server. In line 5, a new generic component is created to represent the backup server, and this component is incremented to the final SPN model. Functions T_{di} and T_{di} are adopted to provide the generic component parameters related failure and repair of backup server (D_{BS} and R_{BS}). Figure 5.4 presents the result SPN model for this algorithm step.

Step 2: Disaster Representation.

In the second step, a generic component is created to represent disasters for data center 1. Line 7 of *parseModel* algorithm is adopted to create a new generic component to represent disasters and recoveries on data center 1. The disaster and recovery parameters are obtained by T_{di} and T_{re} functions (D_{D1} and R_{D1}). The final SPN model is presented in Figure 5.5

Figure 5.5: Step 2: SPN Generation Example

Source: Made by author.

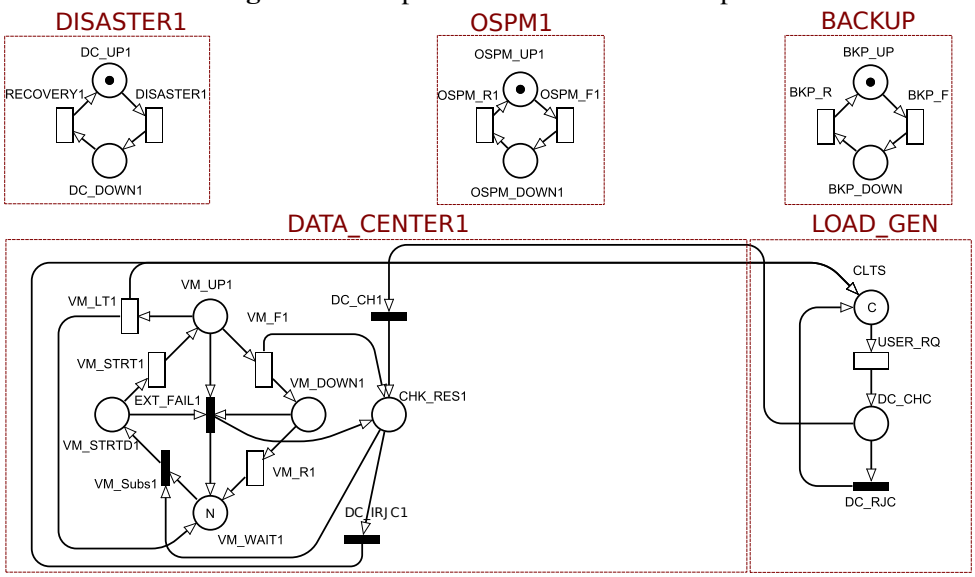
Step 3: Server 1 Representation.

In this phase, two component are incremented to the final model: (i) a generic component to represent the server (*OSPM1*), and (ii) a VM availability or performability component to model the behavior of related VMs (in this case, performability component). Lines 9 and 10 of *parseModel* algorithm create two generic components, one for representing the server's operating system and other to model the server's hardware.

For this particular example, we consider the servers' failure/repair events exponentially distributed. Therefore, an intermediate RBD evaluation is performed (not shown in the algorithm) to create a new SPN sub-model to represent the server's hardware and operating system in a single generic component (i.e., *OSPM1*). The intermediate RBD model is composed of server's hardware and operating system in a series configuration. The RBD evaluation assesses the mean time to failure (MTTF) and mean time to repair (MTTR) related to the intermediate RBD model, and these values are assigned to the *OSPM1* component (see Figure 5.6).

Therefore, instead of having two generics components (hardware and operating system), the resulting model considers just one generic component to represent the server. It is important to stress that this simplification is optional, and users can adopt two distinct generic components to represent hardware and operating system of a given server. Line 11 of *parseModel* algorithm is adopted to create a VM performability to represent the life-cycle of VMs on *OSPM1* server. The SPN representation of result model related to this step is presented in Figure 5.6.

Figure 5.6: Step 3: SPN Generation Example



Source: Made by author.

Step 4: Server 2 Representation.

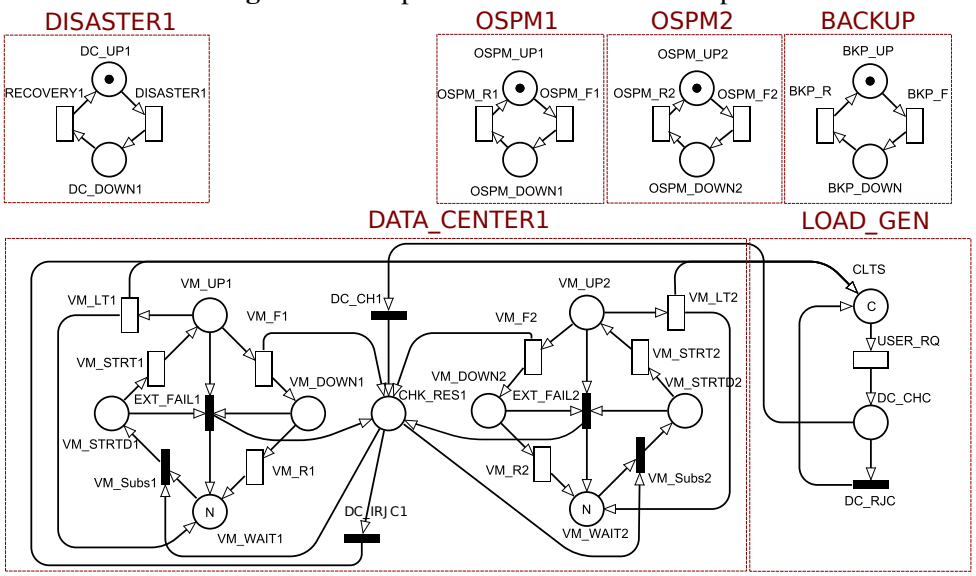
This step aims at representing the second server of data center 2. Lines 9-11 of *parseModel* algorithm will be repeated for each server in a given data center. As this step is analogous to Step 3, it will not be presented in details. However, it is important to observe how VM performability components are combined to represent the possibility to instantiate a VM in any server of the data center. As place *CHC_RES1* is merged for VM performability components of Servers 1 and 2, a VM request can be served by any server of this data center. It is also important to note that the *LOAD_GEN* part is unique for the whole system, and whenever a new VM performability component is created the respective *LOAD_GEN* part is merged with the existing one in the final model (Figure 5.7).

Step 5: Network components Representation.

In this phase, the network components representation is added to the final model (Lines 13-14 of *parseModel* algorithm). This step is analogous to the Step 3, in the sense that an intermediate RBD evaluation is performed to decrease the model complexity. Initially, an RBD evaluation is conducted to assess the availability parameters of the network infrastructure (as a single component). Then, a generic component (*NAS_NET1*) is added to the final model (Figure 5.8) to represent the network failure/repair characteristics.

Step 6: Data Center 2 Representation.

Figure 5.7: Step 4: SPN Generation Example



Source: Made by author.

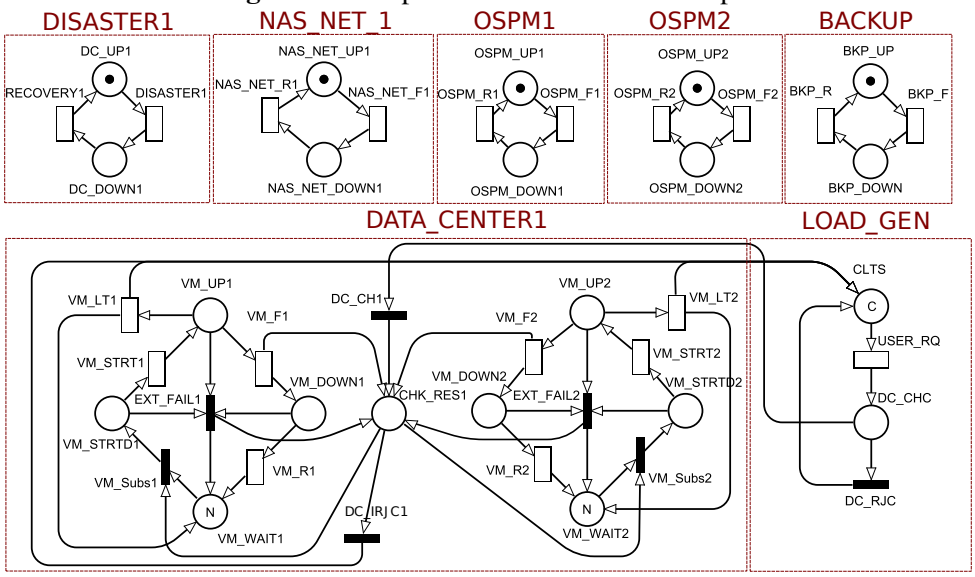
This step aims at representing the second data center of the disaster tolerant cloud system infrastructure. Although this step is represented as just one step, it represents a replication of Steps 2-4 for data center 2. The algorithm steps are analogous. Therefore, this step will not be detailed. Figure 5.9 presents the resulting SPN for this step. Observe that just one *LOAD_GEN* part is represented for the whole system, and a new *CHK_RES* place is created for representing VM requests of data center 2.

Step 7: VM Transmission Representation.

This is the last phase of the SPN result model creation. It involves the creation of VM transmission components for representing the transmission of VMs in case of disasters of data center overloads. For each pair of data centers, a new VM transmission component is created (Line 17 of *parseModel* algorithm). The final SPN model can be observed in Figure 5.10.

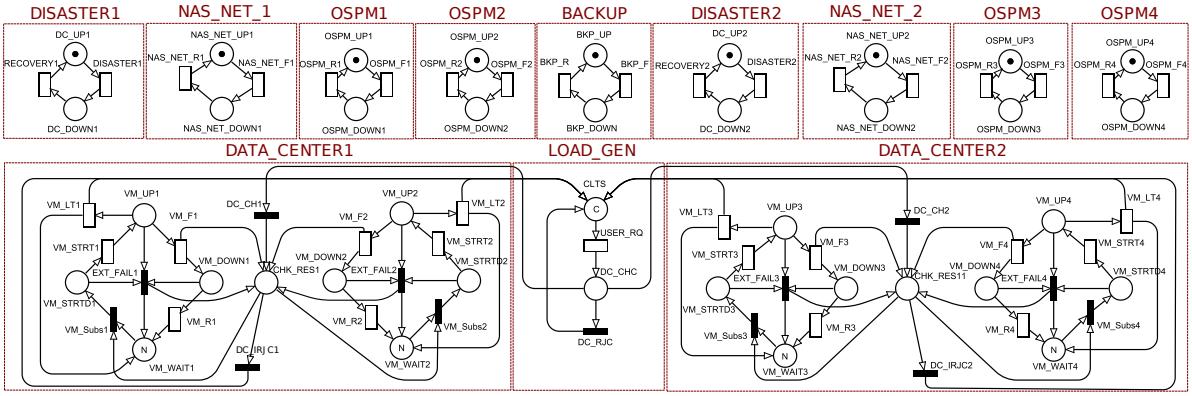
The process to create availability and performance models is similar to one presented previously. The main difference is that performance models adopt the algorithm presented in Figure 5.2. As failures, repairs, and disasters are not represented, generic and transmission components are not represented for performance models. Figures 5.11 and 5.12 show analogous output models of availability and performance of a cloud system with two data centers with two servers each, respectively.

Figure 5.8: Step 5: SPN Generation Example



Source: Made by author.

Figure 5.9: Step 6: SPN Generation Example



Source: Made by author.

5.2 Model Composition Rules

This section presents the formal composition rules adopted in this work to compose the proposed SPN components for creating the final SPN model. In this section, the composition rules will be shown and the respective properties are proved. In this section, we prove that by using the proposed SPN models and the composition rules, the final model will be always structurally bounded as well as conservative.

The proposed modeling approach adopts two operators to compose SPN basic component (e.g., SPN generic component): (i) place renaming, and (ii) net union. Place renaming is an operator that renames a subset of places of a given SPN. This operator represents an auxiliary operator for net union operator that generates a new SPN by merging two existing SPNs. In this section, place renaming is presented, and, next, net union is detailed.

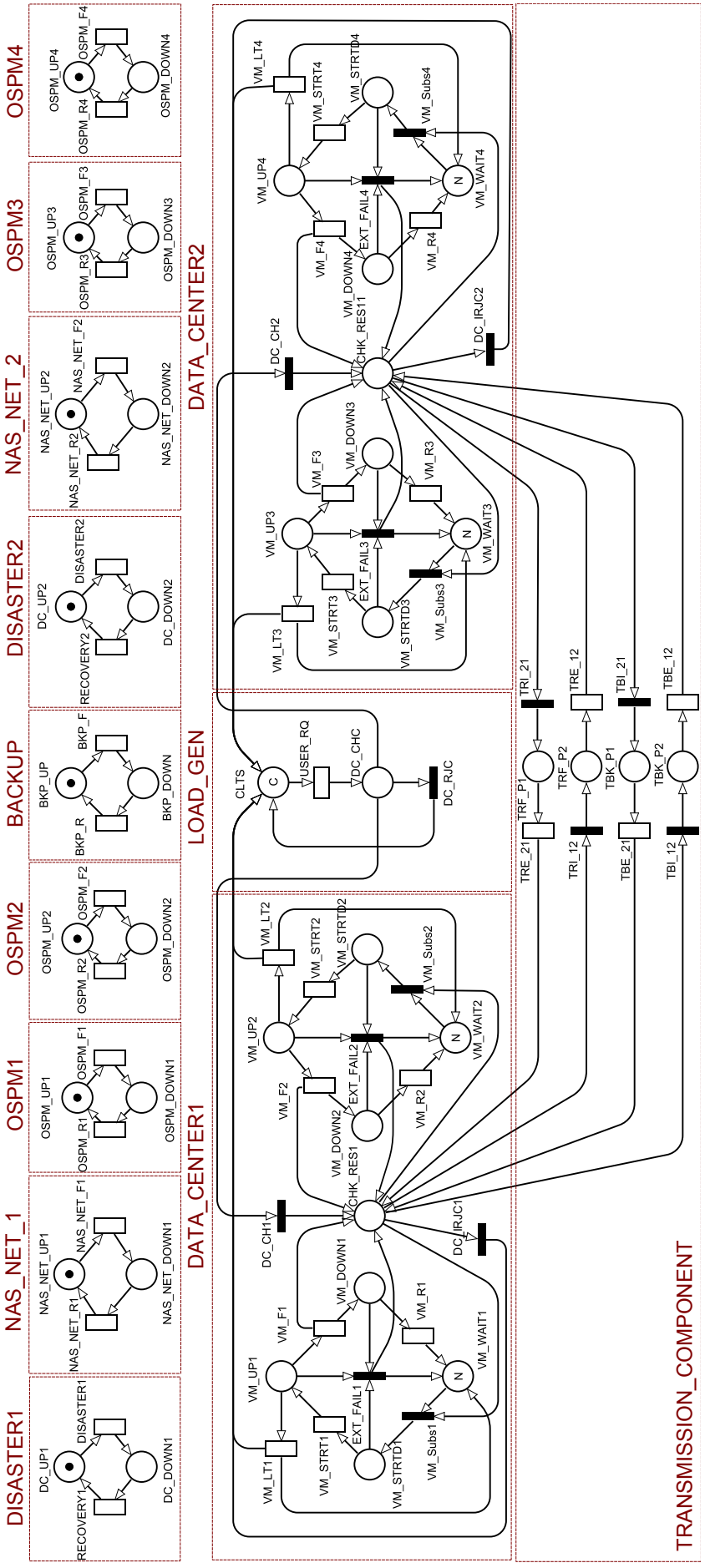


Figure 5.10: Step 7: Performability Model for two data centers

Source: Made by author.

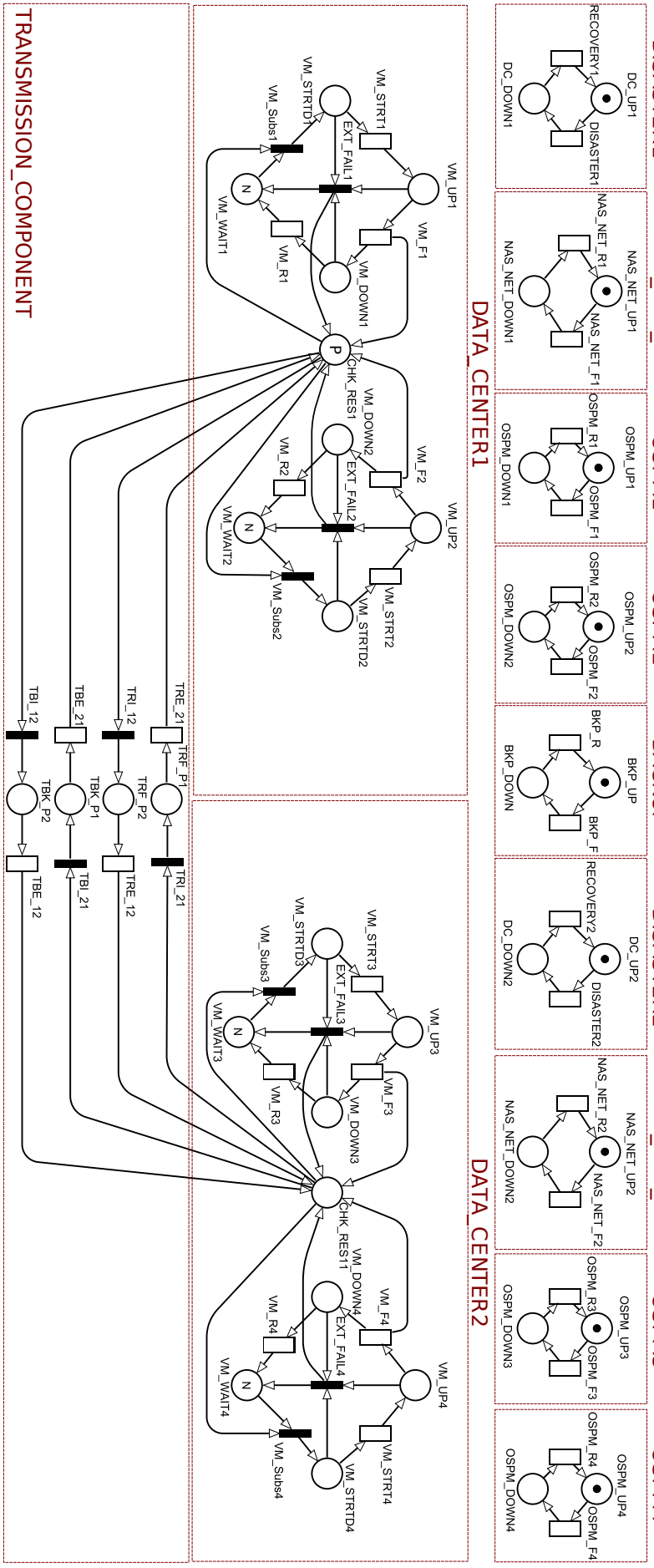


Figure 5.11: Availability Model for two data centers

Source: Made by author.

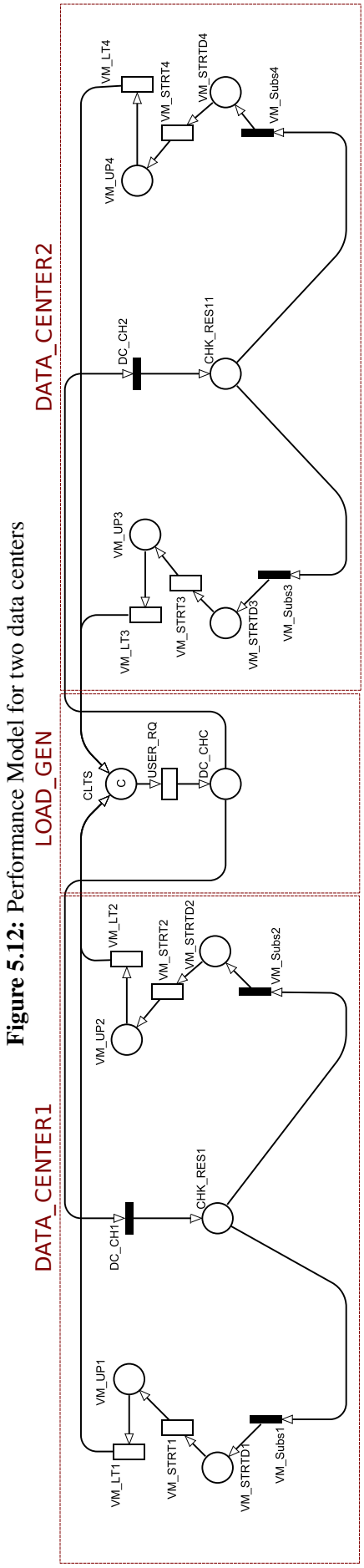


Figure 5.12: Performance Model for two data centers

Source: Made by author.

5.2.1 Place Renaming

This section defines the place renaming operator and provides an example. Before presenting the operator, a notation is provided to simplify the SPN representation. Let $\mathscr{N}' = (P', T', I', O', H', \Pi', M_0', Atts')$ be a SPN, the following notation is adopted:

- $\mathscr{I}_{\mathscr{N}'}(p,t)$ is equivalent to $I'[ordp_{\mathscr{N}'}(p), ordt_{\mathscr{N}'}(t)]$.
- $\mathscr{O}_{\mathscr{N}'}(p,t)$ represents $O'[ordp_{\mathscr{N}'}(p), ordt_{\mathscr{N}'}(t)]$.
- $\mathscr{H}_{\mathscr{N}'}(p,t)$ is adopted to represent $H'[ordp_{\mathscr{N}'}(p), ordt_{\mathscr{N}'}(t)]$.
- $\Pi_{\mathscr{N}'}(t)$ is equivalent to $\Pi'[ordt_{\mathscr{N}'}(t)]$.
- $\mathscr{M}_{0\mathscr{N}'}(p)$ represents $M_0'[ordp_{\mathscr{N}'}(p)]$.
- $Atts_{\mathscr{N}'}(t)$ is equivalent to $Atts'[ordt_{\mathscr{N}'}(t)]$.

$ordp_{\mathscr{N}'}$ and $ordt_{\mathscr{N}'}$ are the mapping functions presented in SPN definition (Section 3.4.2). The Following sets are adopted to define the the domain and codomain of net union/place renaming operators.

Definition 4. ***Set of all SPNs - SSPN.***
Let SSPN = $\{\mathscr{N}_i \mid i \in \mathbb{N}^*\}$, in which $\mathscr{N}_i = (P_i, T_i, I_i, O_i, H_i, \Pi_i, M_{0_i}, Atts_i)$. SSPN corresponds to the set of all stochastic Petri nets (SPNs).

Definition 5. ***Restricted RSSPN - RSSPN***
Let RSSPN $\subset$ SSPN be a restricted set of SPNs, in which $\forall \mathscr{N}_i, \mathscr{N}_n \in RSSPN$,

1. $(T_i \cap T_n = \emptyset \wedge P_i \cap P_n = \emptyset) \vee$
2. $(T_i \cap T_n = \emptyset \wedge P_i \cap P_n \neq \emptyset \rightarrow \forall p \in P_i \cap P_n, \mathscr{M}_{0\mathscr{N}_i}(p) = \mathscr{M}_{0\mathscr{N}_i}(p)) \vee$
3. $(T_i \cap T_n \neq \emptyset \wedge P_i \cap P_n \neq \emptyset \rightarrow \forall p \in P_i \cap P_n \wedge \forall t \in T_i \cap T_n, \mathscr{I}_{\mathscr{N}_i}(p,t) = \mathscr{I}_{\mathscr{N}_n}(p,t), \mathscr{O}_{\mathscr{N}_i}(p,t) = \mathscr{O}_{\mathscr{N}_n}(p,t), \mathscr{H}_{\mathscr{N}_i}(p,t) = \mathscr{H}_{\mathscr{N}_n}(p,t), \Pi_{\mathscr{N}_i}(t) = \Pi_{\mathscr{N}_n}(t), \mathscr{M}_{0\mathscr{N}_i}(p) = \mathscr{M}_{0\mathscr{N}_n}(p), Atts_{\mathscr{N}_i}(t) = Atts_{\mathscr{N}_n}(t))$

RSSPN is a subset of SSPN considering the following situations.

1. Disjoint SPNs.
2. SPNs that have no common transition, but with common places. In this case, the common places must have the same initial Marking ($\mathscr{M}_0$).
3. SPNs with common transitions and places. The common transitions and places must have the following characteristics, i.e.:
   - 3.1 Same input arcs ($\mathscr{I}_{\mathscr{N}_i}(p,t) = \mathscr{I}_{\mathscr{N}_n}(p,t)$).
   - 3.2 Same output arcs ($\mathscr{O}_{\mathscr{N}_i}(p,t) = \mathscr{O}_{\mathscr{N}_n}(p,t)$).
   - 3.3 Same inhibitor arcs ($\mathscr{H}_{\mathscr{N}_i}(p,t) = \mathscr{H}_{\mathscr{N}_n}(p,t)$).

3.4 Same initial marking ($\mathscr{M}_{0\mathscr{N}_i}(p) = \mathscr{M}_{0\mathscr{N}_n}(p)$).

3.5 Same transition attributes ($Atts_{\mathscr{N}_i}(t) = Atts_{\mathscr{N}_n}(t)$).

In this work, we identify elements of a given SPN by its name. Therefore, when two elements have the same name, they actually represent the same entity.

Definition 6. ***Place Renaming.*** *Place renaming is a function denoted by*

$$\rho : (P \cup T) \rightarrow \bigcup_{\mathscr{N} \in SSPN} \{P_{\mathscr{N}} \cup T_{\mathscr{N}}\},$$

that renames places of a SPN $\mathscr{N} = (P, T, I, O, H, \Pi, M_0, Atts)$*, such that a new isomorphic SPN* $\mathscr{N}' = (P', T', I', O', H', \Pi', M'_0, Atts')$ *is obtained.*

- P' = $\{\rho(p) \mid \forall p \in P\}$;
- T' = $\{\rho(t) \mid \forall t \in T\}$;
- $I' \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ is a matrix of input arcs where each entry $i_{jk} \in I'$ corresponds to the arc multiplicity (possibly marking-dependent) from place $\rho(p_j)$ to transition $\rho(t_k)$.
- $O' \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ is a matrix of output arcs where each entry $o_{jk} \in O'$ corresponds to the arc multiplicity (possibly marking-dependent) from transition $\rho(t_j)$ to place $\rho(p_k)$.
- $H' \in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ is a matrix of inhibitor arcs where each entry $h_{jk} \in H'$ corresponds to the possibly marking-dependent arc multiplicity of an inhibitor arc from place $\rho(p_j)$ to transition $\rho(t_k)$.
- $\forall t \in T, \Pi'(\rho(t)) = \Pi(t)$;
- $\forall p \in P, M'_0(\rho(p)) = M_0(p)$;
- $\forall t \in T, Atts'(\rho(t)) = Atts(t)$;

The notation adopted is $\mathscr{N}$' = $\mathscr{N}/\rho$, which represents the application of function ρ in $\mathscr{N}$, obtaining as result $\mathscr{N}$'. In this work, the place renaming function (ρ) is bijective. Therefore, the construction of a respective inverse operation is allowed.

Place Renaming: Example

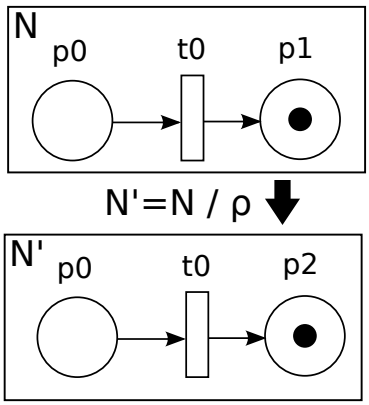
The following renaming function (ρ_1) is adopted to convert the SPN $\mathscr{N}$ into $\mathscr{N}'$. The following renaming function is adopted:

$\forall p_x \in P_{\mathscr{N}}$

$$\rho_1(p_x) = \begin{cases} p_2, & if\ (p_x = p_1) \\ p_x, & otherwise \end{cases}$$

The result of $\mathscr{N}'$ = $\mathscr{N}$ / ρ_1 is depicted in Figure 5.13.

Figure 5.13: Place renaming example



Source: Made by author.

5.2.2 Net Union

This section describes the net union operator, which merges places and transitions (if exist) of the SPNs. The proposed operator considers the characteristics of the adopted SPN components to create new SPNs. Moreover, a mathematical proof is presented to demonstrate that this operator conserves properties of original nets (e.g., structural conservation).

Definition 7. ***Net Union Operator.*** *Net union is a function represented by* $\sqcup$: *RSSPN* $\times$ *RSSPN* $\rightarrow$ *RSSPN that merges two RSSPN. Let* $\mathcal{N}_1 = (P_1, T_1, I_1, O_1, H_1, \Pi_1, M_{01}, Atts_1)$ *and* $\mathcal{N}_2 = (P_2, T_2, I_2, O_2, H_2, \Pi_2, M_{02}, Atts_2) \in$ *RSSPN.* $\mathcal{N}_3$=$(P_3, T_3, I_3, O_3, H_3, \Pi_3, M_{03}, Atts_3)$ *is obtained by* $\mathcal{N}_3 = \mathcal{N}_1 \sqcup \mathcal{N}_2$*, such that:*

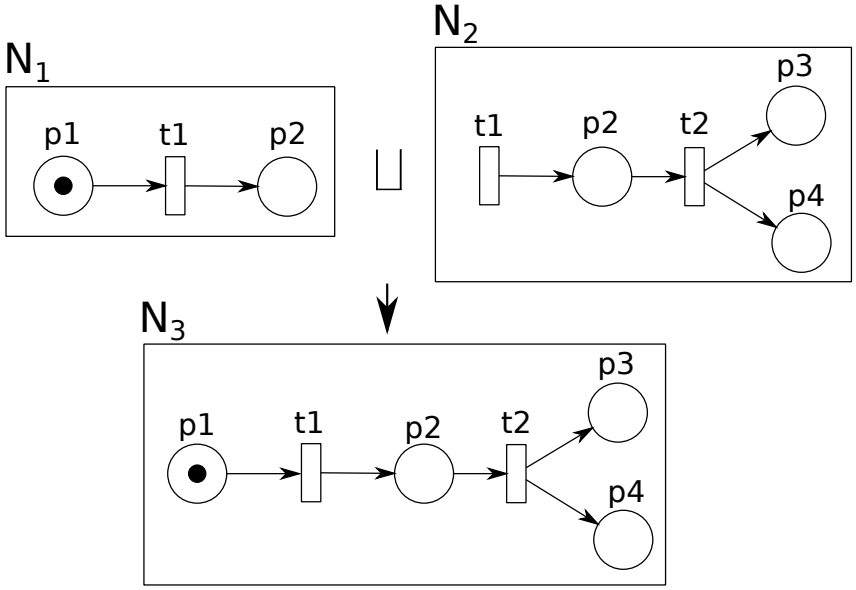
$\forall p_i \in P_3 \wedge t_i \cup T_3$:

- $P_3 = P_2 \cup P_1$
- $T_3 = T_2 \cup T_1$
- $\mathscr{I}_{\mathcal{N}_3}(p_i, t_t) = \begin{cases} \mathscr{I}_{\mathcal{N}_1}(p_i, t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{I}_{\mathcal{N}_2}(p_i, t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\mathscr{O}_{\mathcal{N}_3}(p_i, t_t) = \begin{cases} \mathscr{O}_{\mathcal{N}_1}(p_i, t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{O}_{\mathcal{N}_2}(p_i, t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\mathscr{H}_{\mathcal{N}_3}(p_i, t_t) = \begin{cases} \mathscr{H}_{\mathcal{N}_1}(p_i, t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathcal{N}_2}(p_i, t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\Pi_{\mathcal{N}_3}(t_i) = \begin{cases} \Pi_{\mathcal{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathcal{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$

- $\mathscr{M}_{0_{\mathscr{N}_3}}(p_i) = \begin{cases} \mathscr{M}_{0_{\mathscr{N}_1}}(p_i), & if\ p_i \in P_1 \\ \mathscr{M}_{0_{\mathscr{N}_2}}(p_i), & if\ p_i \in P_2 \end{cases}$
- $Atts_{\mathscr{N}_3}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$

For the sake of readability, the infix notation of $\sqcup$ is adopted (as presented before). An example of net union operator utilization is shown in Figure 5.14. In this example, two RSSPN nets are merged ($\mathscr{N}_1$ and $\mathscr{N}_2$) to generate a new RSSPN $\mathscr{N}_3$. The Appendix B presents properties of net union operator. These properties are adopted to demonstrate that all generated models, using net union operator and the proposed building block models, are structurally bounded and conservative.

Figure 5.14: Net Union Example



Source: Made by author.

5.3 Basic Models Combination Proof

This section presents the proofs that all SPN generated by using net union operator and the proposed basic components are structurally bounded and conservative. As presented in Section 4.3, the created SPN models can be divided into three categories: (i) availability, (ii) performance, and (iii) performability models. As the demonstrations are analogous, just the properties of the composition of availability models will be proved. The demonstrations for performance and performability model follow the same steps.

5.3.1 Availability Models: Structural Properties

The proposed availability models are structurally bounded and conservative. The corresponding proof is composed of two steps. Initially, the set of all generated availability models by

using the net union operator is defined. Then, an inductive proof is presented to demonstrate the aforementioned properties. The same steps are conducted to the proposed performance and performability models.

Definition 8. ***Set of All availability models generated from proposed SPN components and net union operator - $\mathscr{D}^*$.***

Let $\mathscr{G}_e$ the set of all existing generic components, $\mathscr{D}_e$ the set of all VM availability components, and $\mathscr{T}_r$ the set of all transmission components. Then, $\mathscr{D} = \mathscr{G}_e \cup \mathscr{D}_e \cup \mathscr{T}_r$ represents all the possible components that can be adopted to create the proposed availability models. Let $\mathscr{N}_\emptyset$ be the identity element, $\mathscr{D}^*$ represents the set of all proposed availability models using net union operator and $\mathscr{D}$. $\mathscr{D}^* = \bigcup_{i \in \mathbb{N}} \mathscr{D}_i = \mathscr{D}_0 \cup \mathscr{D}_1 \cup \mathscr{D}_2 \cup ...$, in which $\mathscr{D}_0 = \mathscr{N}_\emptyset$, $\mathscr{D}_{i+1} = \{a \sqcup b | a \in \mathscr{D}_i \wedge b \in \mathscr{D}\}$ and $i \geq 0$.

Theorem 1. ***$\mathscr{D}^*$ is Composed of Structurally Conservative and Bounded Models.***

Let $\mathscr{N}$ be a SPN $\in \mathscr{D}^*$, then $\mathscr{N}$ is structurally conservative as well as structurally bounded.

Proof. Base Cases:

1. $\mathscr{N} \in \mathscr{D}_0$. As $\mathscr{D}_0 = \{\mathscr{N}_\emptyset\}$, $\mathscr{N}$ corresponds to the identity element, which has no state space. Therefore, $\mathscr{N}$ is structurally conservative as well as structurally bounded.

2. $\mathscr{N} \in \mathscr{D}_1$. $\mathscr{D}_1 = \{\mathscr{N}_a \sqcup \mathscr{N}_b | \mathscr{N}_a \in \mathscr{D}_0 \wedge \mathscr{N}_b \in \mathscr{D}\}$. Since $\mathscr{N} \sqcup \mathscr{N}_\emptyset = \mathscr{N}$, $\mathscr{D}_1$ is only composed of the proposed SPN components which are structurally conservative as well as structurally bounded models (see Section 4.3).

3. $\mathscr{N} \in \mathscr{D}_2$. $\mathscr{D}_2 = \{\mathscr{N}_a \sqcup \mathscr{N}_b | \mathscr{N}_a \in \mathscr{D}_1 \wedge \mathscr{N}_a \in \mathscr{D}\}$. This base case involves all the possible unions of $\mathscr{D}$ elements. The possible combinations are shown as follows.

   i $\mathscr{N}_c = \mathscr{N}_d \sqcup \mathscr{N}_e$, in which $d = e$. $\mathscr{N}_c$ is structurally bounded as well as structurally conservative, since the net union of a building block with itself results in the same building block. As presented previously, all building blocks are structurally bounded and conservative.

   ii $\mathscr{N}_c = \mathscr{N}_d \sqcup \mathscr{N}_e$, such that $d \neq e \wedge P_d \cap P_e = \emptyset \wedge T_d \cap T_e = \emptyset$. N_c is structurally bounded and conservative, since the net union of disjoint nets preserves the P-invariants of each subnet in the generated model.

   iii $\mathscr{N}_c = \mathscr{N}_d \sqcup \mathscr{N}_e$, in which $d \neq e \wedge (P_d \cap P_e \neq \emptyset \vee T_d \cap T_e = \emptyset)$. $\mathscr{N}_c$ is structurally bounded and conservative, since all possible mergings of two components to create availability models result in structurally bounded and conservative nets. Possible combinations are shown in Appendix A.1.1 and A.1.2.

Inductive Step: $\mathscr{N}_x \in \mathscr{D}_{i+1}$ is structurally conservative and structurally bounded ($\mathscr{I}_x{}^T \times A_x = 0$, in which A_x is the respective incidence matrix and $\mathscr{I}_x{}^T$ is a vector of positive integers).

The recursive definition of $\mathscr{D}^*$ indicates that each $\mathscr{N} \in \mathscr{D}_i$ is composed using one or more base cases presented previously: $\mathscr{D}^* = \bigcup_{i \in \mathbb{N}} \mathscr{D}_i = \mathscr{D}_0 \cup \mathscr{D}_1 \cup \mathscr{D}_2 \cup \mathscr{D}_3 \cup ...$, such that

- $\mathscr{D}_1 = \{a \sqcup b | a \in \mathscr{D}_0 \wedge b \in \mathscr{D}\}$
- $\mathscr{D}_2 = \{a \sqcup b | a \in \mathscr{D}_1 \wedge b \in \mathscr{D}\} = \{(b_0 \sqcup b_1) \sqcup b | b_0 \in \mathscr{D}_0 \wedge b_1, b \in \mathscr{D}\}$
- $\mathscr{D}_i = \{a \sqcup b | a \in \mathscr{D}_i \wedge b \in \mathscr{D}\} = \{((((b_0 \sqcup b_1)...) \sqcup b_{i-1}) \sqcup b | b_0 \in \mathscr{D}_0 \wedge b_1, ..., b_{i-1}, b \in \mathscr{D}\}$

As all base cases do generate structurally conservative as well as structurally bounded nets (the conservative components are preserved and, also, the places that allow the fusion with other building blocks); and net union operator is a commutative monoid (the composition order does not matter), $\mathscr{N}$ is a structurally bounded net. Since $\mathscr{D}_{i+1} = \{((((b_0) \sqcup b_1)...) \sqcup b_{i-1}) \sqcup b_i) \sqcup b | b_0 \in \mathscr{D}_0 \wedge b_1, ..., b_{i-1}, b_i, b \in \mathscr{D}\})$ also utilizes the base cases and net union operator, $\mathscr{N}_x \in \mathscr{D}_{i+1}$ is structurally bounded as well as conservative. Moreover, all possible mergings of components to create availability models result in structurally bounded and conservative nets. Possible combinations are shown in Appendix A.1.1 and A.1.2. Thus, for any $\mathscr{N}_y \in \mathscr{D}^*$, $\mathscr{N}_y$ is structurally bounded and conservative. This concludes the proof $\square$.

A similar demonstration method is adopted for performability and performance models. By using the previous steps, it is possible to demonstrate that all generated performability and performance models are structurally bounded and conservative nets. In this case, the remainder demonstrations presented in Appendix A.1 for base cases and Appendix A.2 for inductive steps are adopted.

5.4 Final Remarks

This chapter described the main algorithm of this thesis and the formal composition rules adopted in the proposed modeling approach. An example was presented to illustrate the translation of an HLM into a final SPN model. Additionally, The structural properties of the generated models by using the proposed composition rules were proved. Therefore, it is possible to state that the generated models will not suffer from state space explosion problem (26). Then, no matter how big the evaluated system is, the final model will always result in a structural bounded and conservative model.

6 Evaluation Environment

This chapter presents an overview of the proposed tools to evaluate disaster tolerant cloud computing systems. Two tools are presented in this chapter: (i) Geoclouds Modcs, and (ii) Eucabomber 2.0. GeoClouds Modcs is an environment for providing support to automatic model generation and evaluation of geographically distributed cloud computing systems. Eucabomber 2.0 supports dependability studies by injecting fault/repair events into the components of cloud systems. Geoclouds Modus supports the proposed modeling approach (Section 4.1) by providing a user-friendly way to create the evaluation models and assess the user metrics. On the other hand, Eucalyptus 2.0 is adopted to validate the generated models by conducting experiments in real-world environments.Initially, Geoclouds Modcs' architecture and data flow are detailed. Then, Eucabomber's architecture, tool's layers and usability overview are presented.

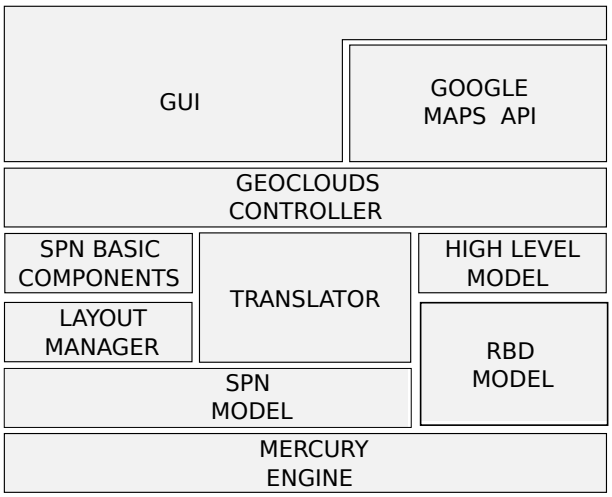
6.1 GeoClouds Modcs

GeoClouds Modcs is an integrated environment for performance, availability and survivability evaluation of geographically distributed IaaS systems taking into account disaster tolerance. The presented environment adopts RBD and SPN models for performability and survivability assessment. By using the tool, cloud system designers can create, edit and evaluate SPN and RBD models through a high-level user interface. Therefore, users need to provide only structural and failure/repair data to perform an evaluation and details related to SPN model composition are not required. In addition to this, the proposed environment exports the generated models to external evaluation tools such as Mercury (77) or TimeNET (72). Then, advanced users can modify exported models to consider a customized evaluation.

Figure 6.1 presents the environment's structure. The framework is written in Java® (78), therefore it is portable to any operating system that supports Java Virtual Machine (79). The tool's Graphical User Interface (GUI) adopts Google Maps API® (80) to estimate the transmission parameters and allow users indicate data center locations. The high level model updating process follows the model view controller design pattern, consequently the SPN models are automatically updated whenever the user performs GUI changes.

GeoClouds Controller manages the translation process that is responsible for creating SPN basic components based on High Level Model. Translator creates intermediate RBD models and combines SPN basic components to generate the final SPN model. Translator utilizes a Layout Manager to organize the graphical positions of places and transitions of final SPN model. Hence, experienced users can edit the SPN final model according to their needs. Generated SPN models can be exported to Mercury or TimeNET. A tool screenshot is presented in Figure 6.2,

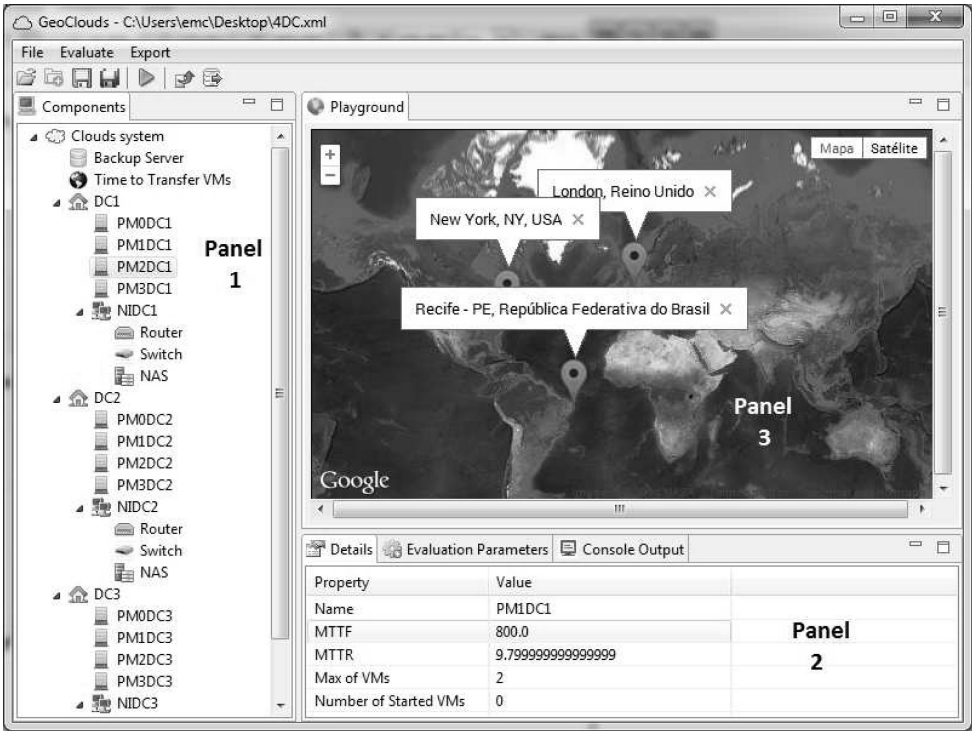
Figure 6.1: System Structure



Source: Made by author.

which is composed of three main Panels. Panel 1 - for representing the IaaS infrastructures and subcomponents. Panel 2 - in which architecture and evaluation parameters are detailed, and Panel 3 - where users can select data center positions.

Figure 6.2: GeoClouds Tool's Screenshot

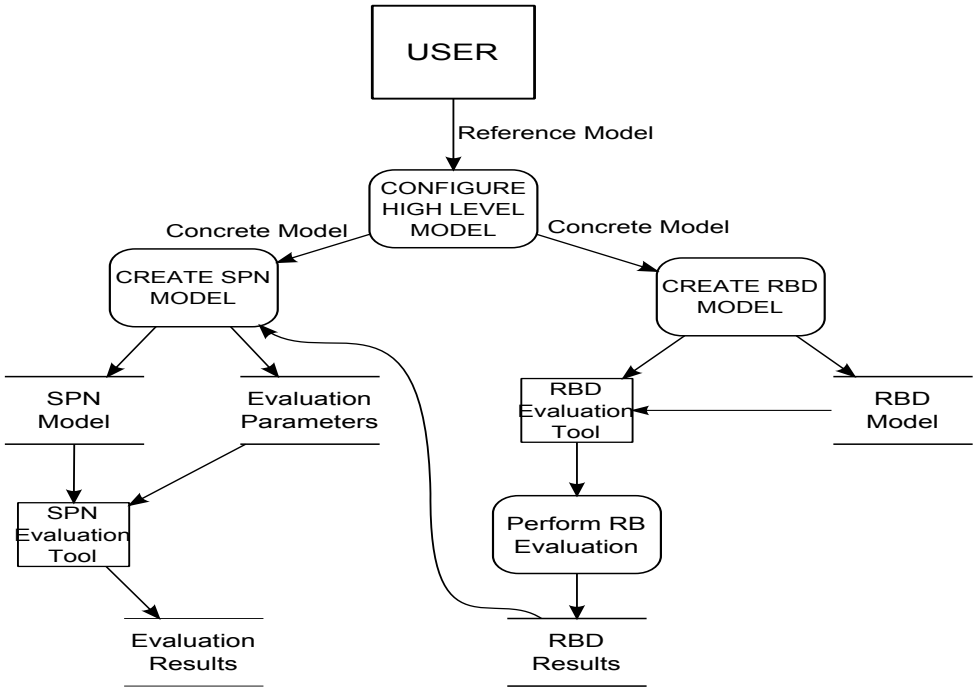


Source: Made by author.

For a better understanding of the tool's internal evaluation process, Figure 6.3 depicts the tool data flow. Basically, this picture summarizes how the models are created and evaluated considering the tool internal steps. Firstly, the user configures the High Level IaaS model by providing the parameters and the evaluation details. After the user's evaluation request, the tool

creates and evaluates an internal RBD model. The RBD results are utilized in the SPN model to configure the servers and network infrastructures. Finally, the SPN model is evaluated and the results are provided.

Figure 6.3: Data Flow Model



Source: Made by author.

6.1.1 MTT estimation

The default approach adopted to assess the network throughput considers the distance between the communication nodes (81) (75). Nevertheless, other methods may also be adopted, for instance, experiments can be conducted to estimate the available bandwidth between sites. Expressions 6.1, 6.2 and 6.3 estimate an upper bound on the transfer rate.

$$Rate < (MSS/RTT) \times (1/\sqrt{p}), \tag{6.1}$$

where *Rate* (kbps) is the TCP transfer rate, *MSS* (bytes) is the maximum segment size per package, *RTT* (ms) is the round trip time, and p is the packet loss ratio. To estimate the RTT, the following equation is adopted:

$$RTT = \frac{Dist}{\alpha \times 100}, \tag{6.2}$$

where *Dist* means the distance in kilometers. $\alpha \in (0,1]$ represents the network directness. α is the ratio of the hop distance (HD) and the end-to-end distance (EED) of two nodes, where EED $\leq$ HP. α values close to one mean the path between the hosts follows a direct path. Values much smaller than one mean the path is very indirect. The values of α, *MSS* and p are user configurable. The distance (*Dist*) is taken from Google Maps API.

$$\alpha = \frac{EDD}{HD} \tag{6.3}$$

6.2 Eucabomber 2.0

This section presents an overview of Eucabomber 2.0. This tools has been developed in this work to support dependability model estimation through fault injection techniques. The fault injector focuses on high-level components such as VMs, server's hardware, and cloud management applications. The following sections present Eucabomber's architecture, tool's layers and usability overview.

6.2.1 Operation and Methodology

This section discusses Eucabomber's features and implementation. Considering the proposed tool, failure and repair commands are injected into the target system at runtime. During system execution, failures and repair are injected by using software commands (e.g., process kill) and the system behavior is analysed from a user point of view (operational VMs). In this work, the system is considered working if the requested VMs by the user are operational. Then, even if parts of the system are not operational, whenever the user requested VMs are running, users consider the system is working.

To compute the time between failures and repairs, a clock records downtime and uptime of each component during system execution. System behavior is observed considering two possible states: UP when it is working properly and DOWN, when it is not.

The time between failures and repairs is generated by the FlexLoadGenerator package (Section 6.2.3). Eucabomber tool also offers a textual interface and the possibility to customize the test environment by using XML files. The library XML Stream for Java (XStream) (82) is adopted to describe the system components and its dependencies. By using this library, users are able to determine the Eucabomber experiments parameters, that includes:

- **Experiment Time.** This parameter determines the experiment duration time in milliseconds.
- **Components list.** This parameter corresponds to the list of parameters that will be adopted in this experiment.
- **IP address of each component.** This parameter assigns an IP address for each component.
- **Relationship between components.** For each component, a dependency list is created. If any component of this list fails, the dependent component fails.
- **Failure/Repair characteristics of each component.** For each component, the failure/repair probability distribution function and its parameters are updated.

Figure 6.4: EucaBomber Textual Menu.

```
- Welcome to EucaBomber's Menu -
1 - Load XML configuration
2 - Create XML configuration
3 - Design Test Architecture
4 - Run Experiment
5 - Clear Loaded data
6 - Exit
```

Source: Made by author.

Eucabomber presents a simple textual interface as illustrated in Figure 6.4. The first two Eucabomber options are related to loading and creating XML configuration files. The third option is to design the experiment. For this purpose, the tester is required to input the total time (in seconds) of the experiment. Then, IP addresses of each machine (physical and virtual) is set. Next, all the necessary data to access the host is provided (e.g., passwords) and the probability distribution data for each component is parametrized. The last three options allows running the experiment, cleaning previous experiment data, and exiting the Eucabomber menu.

6.2.2 Architecture Model

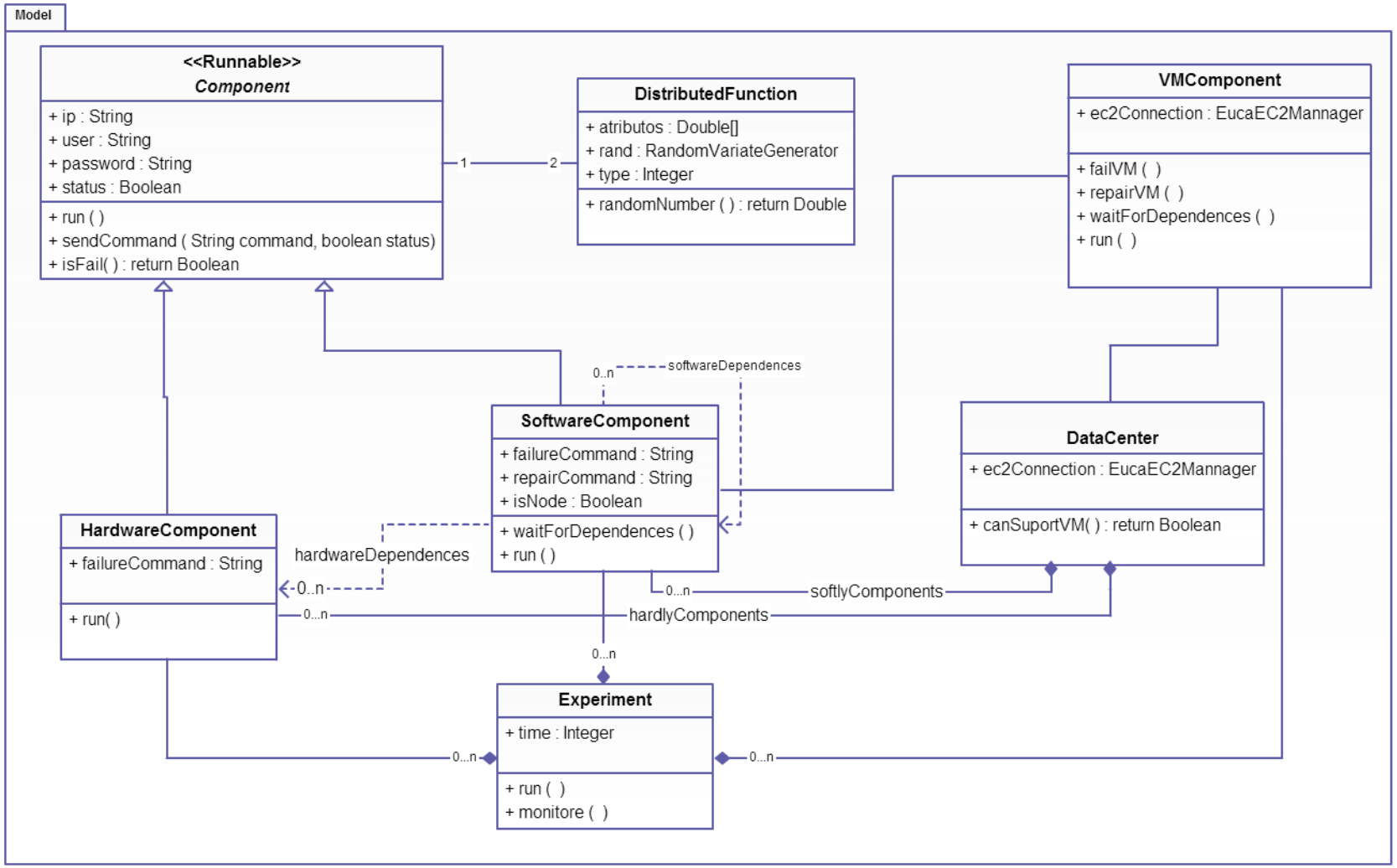
Eucabomber 2.0 presents a new architecture when compared to the previous version (45). The internal tool's infrastructure needed to be redesigned to support dependency between components. For instance, whenever a physical machine fails, all software running on that machine fails too and the new Eucabomber's architecture supports these characteristics.

The Eucabomber 2.0 architecture is presented in Figure 6.5. This picture presents a class diagram that illustrates the Eucabomber 2.0 internal Java classes and the interaction between them. *Component* class represents a high-level entity in the real system that can either fail or repair and can be implemented as *HardwareComponent* or *SoftwareComponent*. For instance, a server is instantiated as a *HardwareComponent* and it is a special case of *Component* class. Each component is associated with a *DistributedFunction* which controls event generation times. This component is responsible for generating the failure/repair events to the components based on the probability distribution function assigned by users.

HardwareComponent represents hardware components and is independent of other components. Different from software components that may be affected by external failures, the failure of any other component does not affect the functioning of the hardware component. *SoftwareComponent*, on the other hand, requires a physical infrastructure and possibly other software to run properly, and therefore depends on *HardwareComponents* and/or other *SoftwareComponents*. *VMComponent* has specific relationships with the cloud manager tool (e.g., Eucalyptus Cloud Controller (47)) of each IaaS infrastructure because it deals directly with the VM. To achieve this, it extends the *SoftwareComponent* by expanding its attributes and associating with a *DataCenter*. *DataCenter* corresponds to an entity that hosts the API Eucaplytus interface.

HardwareComponents, *SoftwareComponents* and *VMComponents* are related to one

Figure 6.5: EucaBomber Class Diagram.



Source: Made by author.

entity called Experiment. This component manages the behavior of the experiment, for example by controlling timers and threads of Eucabomber's execution.

6.2.3 Flex Load Generator Package

The Eucabomber tool adopts FlexLoadGenerator (83) for generating failure and repair random times following a user-specified probability distribution. Although there are many support functions included in FlexLoadGenerator framework, this section presents only the ones employed in Eucabomber core functions.

In order to generate the fault/repair events, a Connection module is adopted for providing network communication between the host running the fault injector and the hosts that will be affected by events. This module employs the SSH2 protocol (84) supported by a Java package which enables the execution of remote commands (85).

The Random Variate Generation module generates random numbers given by a probability function. The FlexLoadGenerator supports the following probability distribution functions: Erlang, Exponential, Geometric, Lognormal, Normal, Pareto, Poisson, Triangular, Uniform, and Weibull.

6.2.4 EucaBomber's Tool Core

As previously explained, the Eucabomber tool was designed to generate events that emulate faults/repairs in high-level cloud components. It is important to state that Eucabomber is only capable of executing repair events in the target environment if the previous fault event was generated by Eucabomber. To simulate real system behavior, the events of fault and repair follow random distributed times supported by the FlexLoadGenerator package (83). Eucabomber supports fault and repair events in the following categories:

- **Infrastructure.** The events generated in this category simulate hardware failures. These failures are implemented by hibernation commands to the operating system, which actually stop the activity of the machine for a limited period. When the hibernation time is over, the machine previous state is restored, thereby performing a repair. Other alternative presented in Eucabomber 2.0 to simulate physical machine failures and repairs corresponds to: (i) shutdown events to emulate failures and (ii) wake on lan (network restart) procedure (86) to emulate repairs.

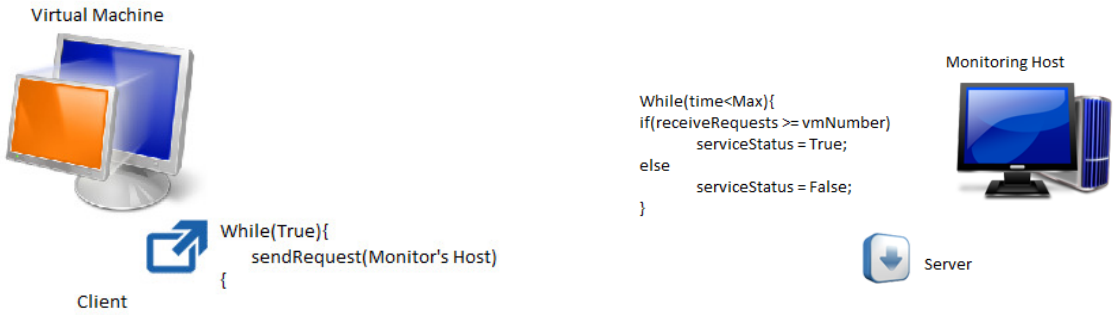
- **Cloud Infrastructure Components.** This mode is focused on interacting with the high-level cloud components (i.e., servers, VMs, and cloud management applications). Fault events of this type are activated by using service commands (e.g., VM start or kill) for Linux distributions and the SSH2 protocol.

6.2.5 Monitoring and Data analysis

The Eucabomber's core has no mechanism to monitor the experiment since the project scope does not include this feature. However, Unix scripts are adopted to monitor the VM state during system execution. These scripts were implemented to periodically check the state of VMs. The scripts run on VMs and a monitoring server machine to verify the status of running VMs.

To monitor the service status, the server script need to check if the amount of operational virtual machines is higher than the number of required VMs to provide the user service. From this point of view, the service availability will be measured adopting a user perspective (number of active VMs). Figures 6.7(a) and 6.7(b) illustrate how the monitoring agent works. First, a new VM image is created and loaded into the cloud system (87). A VM image was prepared with the client monitoring daemon as illustrated in Figure 6.7(a). The VM client sends a message to the server informing that the it is operational. On the server side (Figure 6.7(b)), the program loops around and periodically checks each message sent from the user's VMs to calculate the number of available VMs. If the server receives the message from the VM, the VM is considered operational (UP), and this event is registered in the system log. Otherwise, a DOWN event for this VM is registered in the system log. The monitoring mechanisms is replicated to all new VMs, and each new VM knows in advance the monitoring server IP address.

Figure 6.6: Monitoring agents.



(a) Client inserted in the VM image. (b) Previous configured host server.

Source: Made by author.

By applying this process, at the end of the experiment there will be a log containing an assortment of UP and DOWN strings. Each one these entries is a measure of server status at a given time. Therefore, it is possible to approximate this discontinuous data to continuous data by normalizing the values between points, and finally estimate the confidence interval of availability. By using Equation 3.15, it is possible to calculate the ρ estimator from the number of UP (uptime or S_n) and DOWN (downtime or Y_n) entries in the log file.

The number of errors (n) is easily calculated by comparing the number of UP and DOWN states. Adopting the ρ estimator, the n number of faults, and the confidence level (α), the availability can be computed with the Equations 3.16 and 3.17.

6.3 Final Remarks

This chapter presented the proposed tools to support cloud system evaluation. The first tool (GeoClouds Modcs) is adopted for creating and evaluating IaaS disaster tolerant systems by composing basic submodels. By using these tools, modeling details are hidden to the final users. Then, even users that have no skills on SPN-RBD modeling will be able to perform a system evaluation. However, he/she should provide availability and performance parameters to perform the system evaluation.

The second tool was presented (Eucabomber 2.0) is adopted to perform availability evaluation for IaaS cloud systems. The tool utilizes fault injection and monitoring scripts to perform system evaluation.

7 Case studies

To illustrate the feasibility of the proposed work, we present six case studies considering survivability, performance and dependability metrics to evaluate geographically distributed disaster tolerant cloud computing systems. The purpose of each case study is summarized as follows:

- *Case Study I.* This case study corresponds to a simple availability evaluation example to illustrate how IaaS system can be evaluated by using our proposed framework.
- *Case Study II.* The second case study presents a disaster tolerant cloud system evaluation taking into account dependability and performance metrics in a single model (performability model).
- *Case Study III.* The third case study shows an availability evaluation considering large cloud computing systems. The central idea of this case study is to demonstrate how the proposed models can scale to represent large cloud computing systems.
- *Case Study IV.* This case study is proposed to show how the proposed framework can be adopted to evaluate how IaaS systems behave in case of disasters. A survivability evaluation is performed by using the proposed models.
- *Case Study V.* This case study presents a comparison between the results of the proposed availability models and fault injection experiments performed by Eucabomber 2.0.
- *Case Study VI.* The last case study provides a performance evaluation of a small cloud computing system. The evaluation results are compared to experiments performed in a real test-bed environment.

7.1 Case Study I

This section presents a case study that illustrates how the proposed framework can be adopted to estimate availability metrics in cloud computing systems. The evaluated infrastructure is composed of two data centers and a Backup Server. The data centers are respectively located in Paris-France and Frankfurt-Germany. The designer has two location options for Backup Server (London-England or Brussels-Belgium). Distances between data centers and possible backup servers are presented in Table 7.1. The distances between locations for all case studies were taken from (88).

Table 7.1: Distance of facilities.

Location 1	Location II	Distance (Km)
Paris	Frankfurt	478.4
Paris	Brussels	264.2
Paris	London	343.9
Frankfurt	Brussels	317.5
Frankfurt	London	771.1

Each data center is composed of servers and a network infrastructure. The servers can execute up to two virtual machines. In order to consider the cloud system available, at least two virtual machines must be operational.

Table 7.2: Case study scenarios.

Scenario	Disaster Mean Time (months)	Number of servers (total)	BS_location
A1	6	2	London
A2	6	2	Brussels
A3	6	4	London
A4	6	4	Brussels
A5	12	2	London
A6	12	2	Brussels
A7	12	4	London
A8	12	4	Brussels

As mentioned in Section 3.1, VMs can be instantiated in another physical machine or data center in case of failure. For this example, eight scenarios will be evaluated considering diverse disaster mean time, total number of servers and Backup Server locations (Table 7.2). For this example, we assume the number of servers is the same for each data center.

For the following case studies (except Case Study IV), we considered the approach presented in (81) to estimate the *MTT* value. This approach assesses network throughput based on the distance between the communication nodes. As stated before (Section 6.1.1), this approach corresponds to the default approach adopted to evaluate MTT values. However, another approach can be utilized for estimating the time to transmit VM data. For instance, actual measurements can be performed to estimate MTT. Real experiments to estimate data transmission rates were conducted in Case Study IV. The equation associates a constant α with the network speed, which can vary from 0 (no connection) up to 1.0 (fastest connection). We assume the package loss ratio 1%, MSS 1460 bytes, α as 0.45 and the size of VMs as 4GB. For the next case studies, we considered the mean time between disaster to be 100 years and the data center to take 30 days to be repaired. The mean time to start a VM is five minutes.

To evaluate the system, the designer must provide architecture parameters, such as the data center and Backup Server locations as well as the dependability parameters (i.e., failure/repair rates) for each component (see Table 7.3). Table 7.4 presents the results associated with this example. Scenario A8 presents the highest availability and consequently the lowest

Table 7.3: Dependability parameters for Case study I.

Component	MTTF(h)	MTTR(h)
Server Operating System	4000	1
Server Hardware	1000	12
Switch	43000	4
Router	14077	4
NAS	20000	2
VM	2880	0.5
Backup Server	50000	0.5

Table 7.4: Case study I results.

Scenario	Availability	Annual downtime(h)
A1	0.97103	253.7
A2	0.97217	243.7
A3	0.97644	206.3
A4	0.97694	202.0
A5	0.99404	52.2
A6	0.99411	51.5
A7	0.99739	22.8
A8	0.99742	22.6

annual downtime. This result is not a surprise since it presents the highest disaster mean time, number of servers, and nearest BS location in relation to the data centers. Scenario 1 shows analogous results, but considering low availability results. Although these results are relatively straightforward, larger and complex IaaS cloud infrastructures may present non-trivial results.

7.2 Case Study II

In order to present how performability models can be evaluated by using the proposed approach, this case study evaluates a set of cloud system scenarios in which IaaS cloud systems are deployed into two different data centers. Each data center is composed of two physical machines and each physical machine can execute up to two VMs. Therefore, the maximum number of running VMs is eight.

We have conducted an availability evaluation considering distance between data centers and disaster mean time. We assume data centers located in the following pairs of cities: Rio de Janeiro (Brazil)-Brasilia (Brazil), Rio de Janeiro-Recife (Brazil), Rio de Janeiro-NewYork (USA), Rio de Janeiro-Calcutta (India) and Rio de Janeiro-Tokyo (Japan). We assume that the Backup Server is located in São Paulo (Brazil). Table 7.5 presents the distances between the adopted locations to place data centers or backup servers.

We assume the mean time between disaster to be 100 years and the data center to take 30 days to be repaired. Moreover, a VM takes five minutes to start and the mean time between requests is half an hour. The mean time for using a VM is 720 hours. Previous work (89) presents

Table 7.5: Distance of facilities.

Location 1	Location II	Distance (Km)
Rio	Brasilia	1335.1
Rio	Recife	1875.5
Rio	New York	7767.7
Rio	Calcuta	15085.34
Rio	Tokyo	18587.80
São Paulo	Rio	224.4
São Paulo	Brasilia	1176.8
São Paulo	Recife	2132.1
São Paulo	New York	7694.2
São Paulo	Calcuta	15446.1
São Paulo	Tokyo	18554.9

the dependability parameters associated with the devices. GeoClouds Modcs Framework was adopted to perform the evaluation. To perform system evaluation we created a performability model (see Section 4.3.4) with metrics oriented to providers and clients.

The provider-oriented metric is the machine utilization in terms of maximum number of VMs (U) and the user-oriented metric is the probability of a task to be completed without error (P). U is calculated as follows:

$$U = \frac{\sum_{j=1}^{M} E\{\#VM\_UPj\}}{M \times N} \tag{7.1}$$

.

Where M corresponds to the number of PMs and N is the maximum number of VMs per physical machine. P corresponds to:

$$P = \frac{(\sum_{j=1}^{Pm} E\{\#VM\_UPj\}) \times (1/T)}{P\{\#CLTS > 0\} \times (1/R)} \tag{7.2}$$

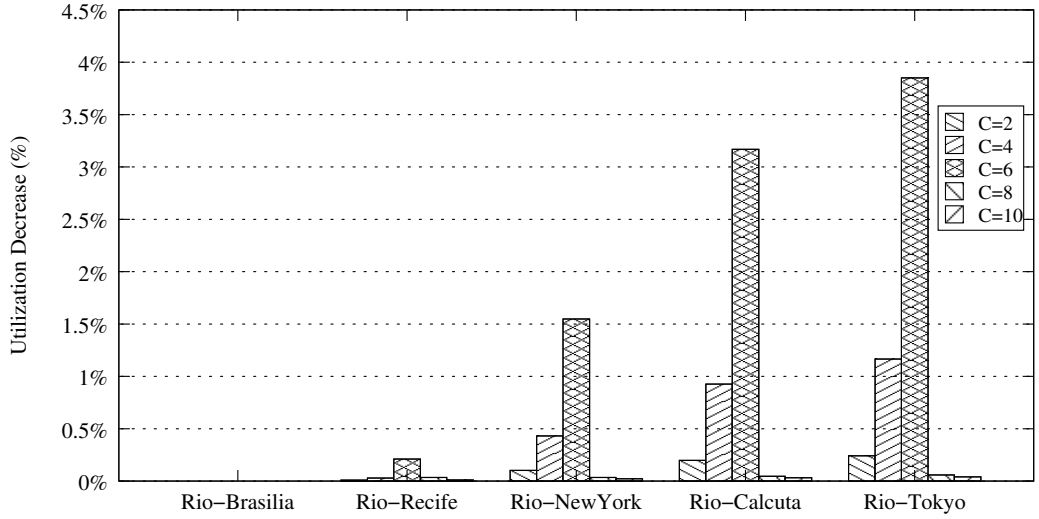
.

In which T and R correspond to the times associated with transitions *VM_LT* and *USER_REQ*. This expression divides the throughput of completed requests by the incoming requests throughput to compute the percentage of completed requests.

Figure 7.1 shows that utilization percentage decreases for each different pair of cities and number of clients (C). The results show that the utilization decreases with the distance if the system is not stressed. In case of high system load ($C > 6$), the effect of distance on the utilization is reversed. In this case, the distance variation has no effect on system utilization as the system is overloaded.

A task rejection considering the proposed approach happens if the infrastructure is broken or the servers are full. Figure 7.2 presents the P percentage increase considering the same set of scenarios. It is possible to observe that, in this particular case, P rises with the distance between the data centers (until $C = 6$). With the growth of system load ($C > 6$), P decreases depending on

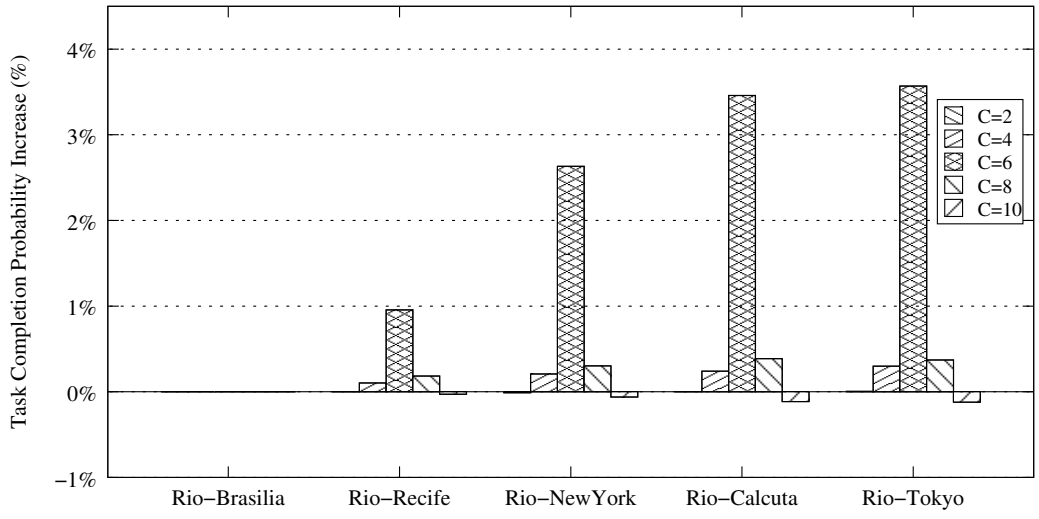
Figure 7.1: Utilization decrease of different distributed cloud configurations



Source: Made by author.

the data centers distance. Therefore, for this particular system we can conclude that P is highly impacted by server utilization.

Figure 7.2: P increase for different cloud configurations



Source: Made by author.

Table 7.6: P and U for the baseline architecture (Rio de Janeiro-Brasilia)

Client load	U	P
C=2	0.2497989	0.9999299
C=4	0.4989833	0.9952341
C=6	0.7471487	0.9338905
C=8	0.9861435	0.4542866
C=10	0.9867157	0.2934411

Table 7.6 presents the values for U and P of the baseline architecture (Rio de Janeiro-

Brasilia) considering different user loads.

7.3 Case Study III

In order to demonstrate the scalability of the proposed framework, we evaluated a set of cloud system scenarios with a high number of servers. As the previous case study, the proposed scenarios consider a cloud system with two data center. We assume that the Backup Server is located in London (UK). The following pairs of cities were considered to place the data centers: Budapest (Hungary)-Barcelona(Spain), Paris(France)-Amsterdam(Netherlands), Crawley(UK)-Basildon(UK), Bristol-Nottingham(UK). This configuration was selected because each pair of cities have data centers with similar distances to the backup server. Table 7.7 presents the distances between the adopted locations to place data centers or backup servers.

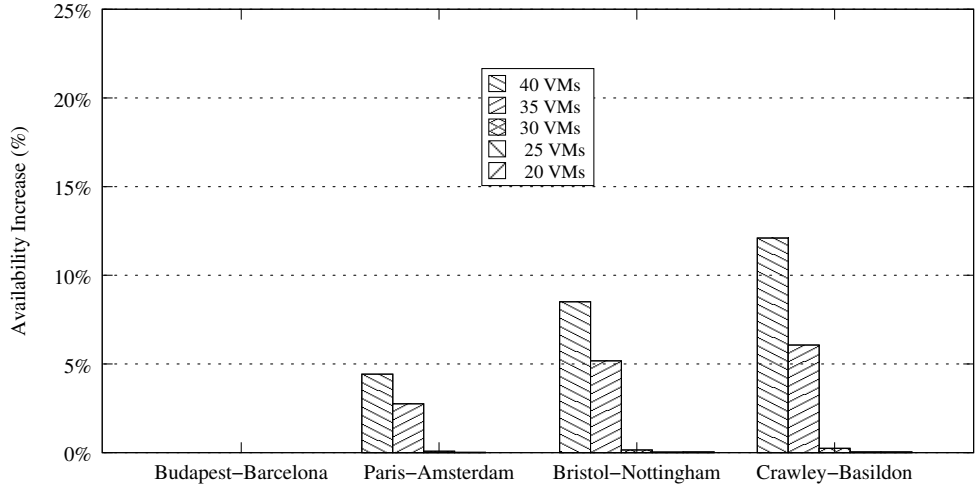
Table 7.7: Distance of facilities.

Location 1	Location II	Distance (Km)
Budapest	Barcelona	1499.1
Paris	Amsterdam	430.3
Bristol	Nottingham	193.3
Crawley	Basildon	70.1
London	Budapest	1450.8
London	Barcelona	1140.3
London	Amsterdam	357.7
London	Paris	343.9
London	Nottingham	175.7
London	Bristol	170.6
London	Crawley	44.5
London	Basildon	43.3

This case study evaluates the cloud system in terms of system availability by using the proposed availability model (see Section 4.3.2). Each data center is composed of 40 servers and each one can execute up to two VMs. As the previous case study, we estimate the *MTT* value by adopting the approach presented in (81) with the same parameters. The dependability and disaster parameters are the same of the previous case study.

Figure 7.3 presents the availability increasing for the different pair of cities. The configuration with data centers located in Budapest and Barcelona was adopted as baseline architecture. For each scenario, we changed the required number of VMs to consider the system operational (20, 25, ..., 40). It is possible to observe that, for this particular example, the system availability increases as the distance decreases. However, when the number of required VMs to consider system operational decreases, the distance of data centers becomes less significant. Table 7.8 presents the availability values for the baseline infrastructure. As expected, the availability increases as the number of required VMs decreases.

Figure 7.3: Availability for different cloud configurations.



Source: Made by author.

Table 7.8: Availability values for the baseline architecture (Budapest-Barcelona)

Number of required VMs	Availability (%)
40	96.956
35	97.683
30	98.152
25	98.173
20	98.178

7.4 Case Study IV

This case study is proposed to demonstrate how survivability metrics can be evaluated in cloud computing systems by using the proposed approach. The environment is composed of five data centers and a BS. The data centers are located in the following cities: (i) New York (USA), (ii) Rio de Janeiro (Brazil), (iii) Zurich (Switzerland), (iv) Vienna (Austria) and Sydney (Australia). The backup server is located in Ilmenau (Germany). Table 7.9 presents the distances between the adopted locations to place data centers or backup servers.

Table 7.9: Distance of facilities.

Location 1	Location II	Distance (Km)
Ilmenau	Zurich	407.1
Ilmenau	Vienna	480.9
Ilmenau	New York	6326.7
Ilmenau	Rio de Janeiro	9756.5
Ilmenau	Sydney	16330.5

This experiment evaluates the recovery and backup process by using the modelling approach presented in Section 4.4. In this case study, we assume that 512 MB should be transmitted to BS to synchronize the VM data (backup process) and each VM image has 4

GB (recovery process). To estimate the time to transmit the VM data between the data centers and BS, a measurement process has been conducted to characterize the transfer rate between the backup server and the data centers. Mercury-ASTRO (77) and TimeNET (72) tools have been adopted to perform the evaluation. The transfer rates between BS and each data center is presented in Table 7.10.

Table 7.10: Transfer rates between BS and each data center.

Data Center Location	Rate (MB/s)	Standard Deviation (MB/s)
Zurich	2.0701	0.4019
Vienna	1.7412	0.3270
New York	1.1253	0.2126
Rio de Janeiro	0.6859	0.1351
Sydney	0.5659	0.1088

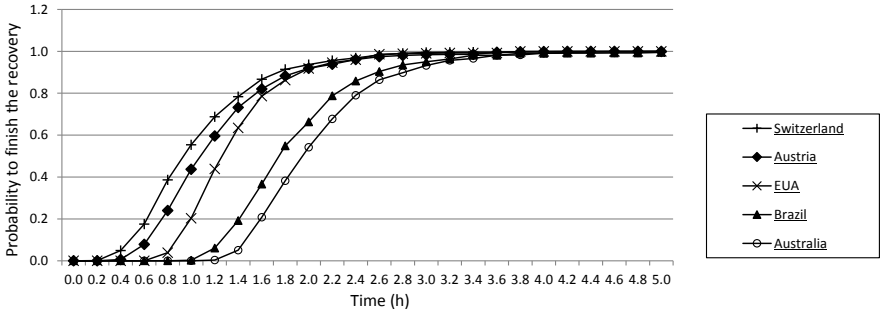
To evaluate the recovery process, the behavior of each data center to receive and instantiate five VM images (4 GB each) is considered. The mean time to detect the disaster is 30 minutes and the mean instantiation time is five minutes. The transmission success probability considered is 99.9%. For this particular experiment, the transition *DC_TRS* was converted to Hypoexponential subnets for all data centers (Section 3.4.2.4). The evaluation results for each data center are presented in Figure 7.4 and some important points are summarized in Table 7.11. For instance, considering that the RTO is two hours, the data center located in Zurich can be a good option to recover the service if we assume that probability to recovery should be higher than 93%. On the other hand, if the RTO is four hours and the minimum probability to recovery is 99%, all data centers can be adopted to restart the affected VMs.

Table 7.11: Probability to recover the service for different data centers

Time(h)	Zurich	Vienna	New York	Rio de Janeiro	Sydney
1.0	0.5539	0.4370	0.2028	0.0027	0.0003
2.0	0.9369	0.9187	0.9183	0.6636	0.5424
3.0	0.9946	0.9909	0.9837	0.9501	0.9326
4.0	0.9998	0.9989	0.9973	0.9913	0.9903

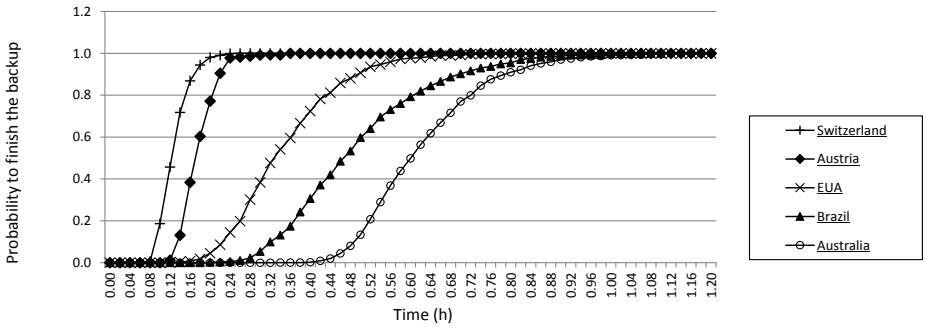
A similar evaluation was performed considering the backup process. In this case, each data center synchronizes the data of five VMs (512 MB each) to BS and the replication process takes one minute. Figure 7.5 presents the evaluation results. Additionally, Table 7.12 shows important points considered in this evaluation. For the worst case scenario, if the difference between the RPO and the backup period is 0.2 hours and minimum probability to backup the VMs is equal to 0.98, only the data center of Zurich can be adopted. On the other hand, if the difference between the RPO and the backup period is one hour and minimum probability is equal to 0.99, all data centers respect the requirement.

Figure 7.4: Recovery probability along the time



Source: Made by author.

Figure 7.5: Backup probability along the time



Source: Made by author.

Table 7.12: Probability to backup the service for different data centers

Time(h)	Zurich	Vienna	New York	Rio de Janeiro	Sydney
0.2	0.9801	0.7715	0.0449	0.0019	0.0001
0.4	0.9999	0.9998	0.7235	0.3062	0.0019
0.6	∼1.000	∼1.000	0.9771	0.7928	0.4980
0.8	∼1.000	∼1.000	0.9988	0.9545	0.9100
1.0	∼1.000	∼1.000	∼1.000	∼1.000	0.9911

7.5 Case Study V

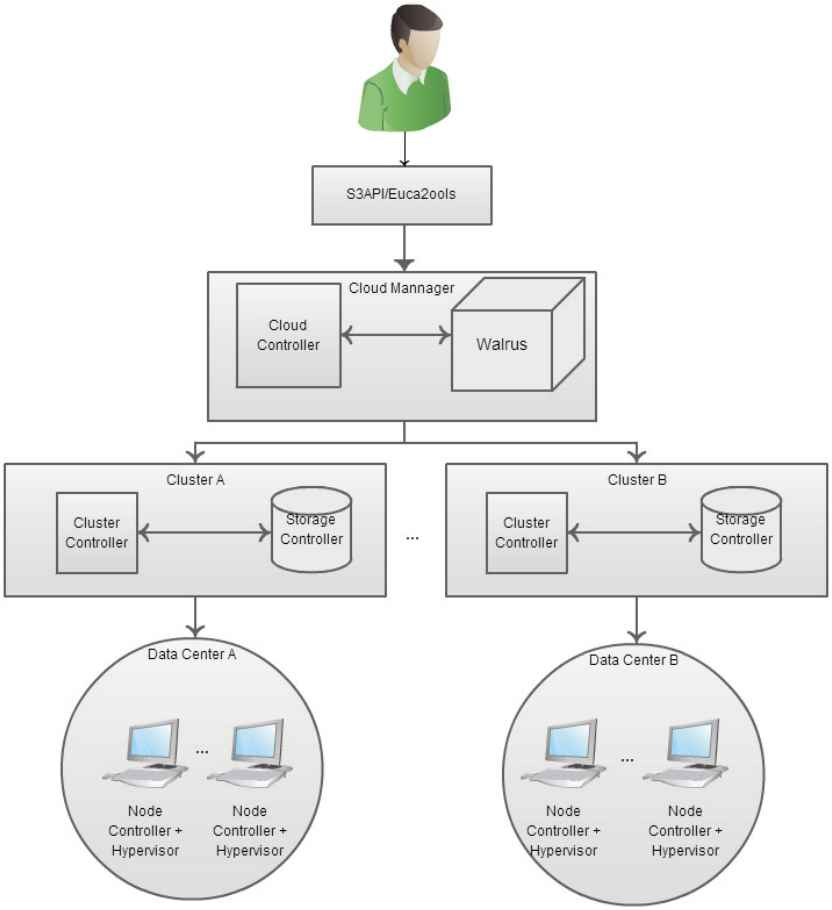
This case study compares the results of the proposed availability models with fault injection experiments performed by Eucabomber 2.0. Eucalyptus cloud platform is adopted for this case study. However, the proposed models can be adopted to evaluate other cloud infrastructures.

This section is presented as follows. First, the Eucalyptus cloud platform is briefly presented. Then, the experiment environment is presented. Next, the availability model to represent the experiment environment is presented. Finally, the results are presented.

7.5.1 Eucalyptus IaaS Platform

Eucalyptus IaaS platform is a popular open source solution for cloud software. The environment adopts an API compatible with AWS/EC2 (90) and is one of the most utilized software solutions for building private or hybrid clouds.

Figure 7.6: Eucalyptus Cloud Computing Architecture.



Source: Made by author.

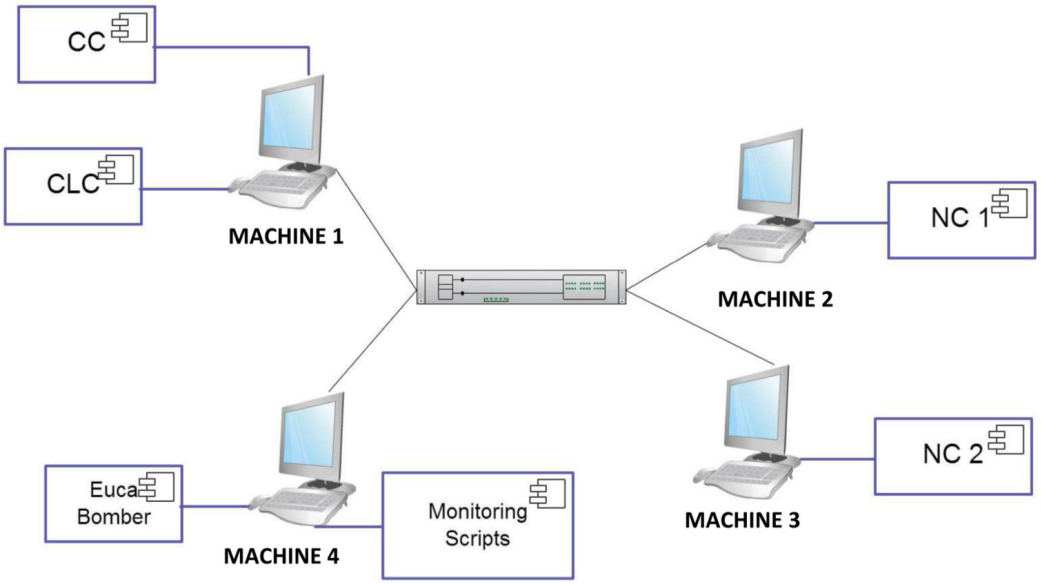
As illustrated in Figure 7.6, Eucalyptus clouds are composed of the following components: one or more Node Controller (NC); one or more Cluster Controllers (CCs); one Cloud Controller (CLC) and a storage (Walrus). NC is the component that controls the VM life-cycle operations (e.g. run, terminate) which is adopted in conjunction with an external hypervisor (e.g., VMware, Xen, Kvm).

CC manages the local network and the elastic memory resources of NC sets, generally in the same data center. It corresponds to the front-end for a cluster within Eucalyptus cloud and controls the node controllers and walrus. CLC and Walrus correspond to the components responsible for providing an API for user management and system property control (e.g. VM image control, data base management) (91). CLC is front-end of the entire cloud and provides web services interface compatible with Amazon EC2/S3 to the client tools. Additionally, interacts with the rest of the components of the Eucalyptus infrastructure on the other side.

7.5.2 Experiment Environment

The test environment consists of four machines (Figure 7.7). A fault injector and monitor (Machine 4), front-end (Machine 1) that contains the Eucalyptus Cloud Controller (CLC) and Cluster Controller (CC). Machines 2 and 3 execute Eucalyptus Node Controller (NC and KVM hypervisor) and host virtual machines.

Figure 7.7: Structural components of testbed environment.



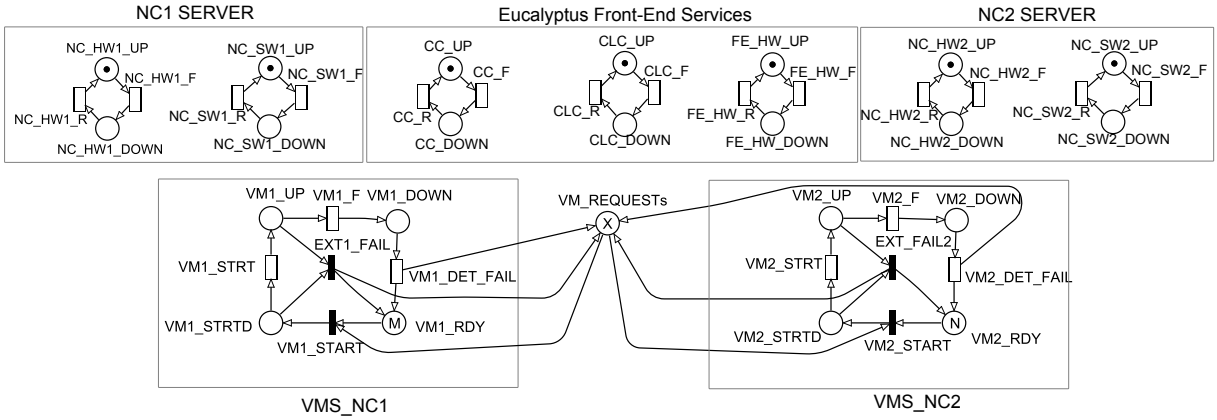
Source: Made by author.

All servers are in the same private network and Eucabomber's server accesses other components using SSH commands. Eucabomber's machine runs the monitoring server (Section 6.2.5) and the virtual machines hosted in the node controllers (NC 1 and NC 2) are pre-loaded with the monitoring client daemon. With this testbed environment, faults can be injected in the principal Eucalyptus resources (e.g., physical and virtual machines), as well as service status can be traced by means of monitoring scripts.

7.5.3 Availability Model

This section presents the proposed availability model to evaluate availability in Eucalyptus infrastructure. In this case, CC, NC, and CLC should be considered in the evaluation model to represent the Eucalyptus components. This section presents the SPN availability model for representing the proposed testbed environment. This model is divided in five main parts (Figure 7.8): **Eucalyptus Front-End Services** for modeling CC, CLC and Front-end hardware (Machine 1); **NC1 SERVER** and **NC2 SERVER** for representing the hardware and software of Machines 2 and 3 (see Figure 7.7); **VMS_NC1** and **VMS_NC2** that correspond to running VMs on Machines 2 and 3. As the components of Machine 4 (Eucabomber and monitoring scripts) are not part of cloud system, these entities are not represented in the model.

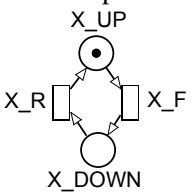
Figure 7.8: SPN model for Eucalyptus testbed environment



Source: Made by author.

Components with no redundancy mechanisms or dependency relations (e.g., NC hardware) are represented as submodels (see Figure 7.9), which are composed of two states and two transitions. Assuming *CC* as the represented component, this submodel might be in two states, operational (*CC_UP*) or failed (*CC_DOWN*). Transitions *CC_F* and *CC_R* denote respectively the component's failure and repair.

Figure 7.9: Basic component SPN submodel



Source: Made by author.

This submodel has two parameters (not shown in Figure 7.9), namely *X_MTTF* and *X_MTTR*, which represent delays associated to transitions *X_F* and *X_R*, respectively. Table 7.13 depicts the attributes related to these transitions. In this context, a component is working if there is no tokens in place *X_DOWN*. Therefore, to represent a failed component, the number of tokens in *X_UP* must be zero. Table 7.14 shows the submodels adopted in Figure 7.8 for representing components with no dependency relations.

Table 7.13: Transition attributes associated with a component.

Transition	Delay	Description
X_F	MTTF	Component failure event
X_R	MTTR	Component repair event

In Figure 7.8, *VMS_NC1* and *VMS_NC2* submodels represent the set of VMs that run on NC1 and NC2 servers, and whenever a dependent device (e.g., underlying hardware) fails, the respective VMs fail too. *VMS_NC1* and *VMS_NC2* are analogous, then just *VMS_NC1*'s structure

Table 7.14: Submodels for representing no redundancy components.

Submodel's name	Description
NC_HW1	NC1's hardware
NC_SW1	NC1's software
NC_HW2	NC2's hardware
NC_SW2	NC2's software
FE_HW	Front-end's hardware
CC	Cloud controller software
CLC	Cluster controller software

will be detailed. *VMS_NC1* is composed of places *VM1_UP*, *VM1_DOWN*, *VM1_STRTD* and *VM1_RDY*. These places denote, respectively, the amount of VMs in states operational, failed, starting, and waiting for request. Place *VM_REQUESTs* represent requested VMs, which can be executed on NC1 or NC2 server.

Exponential transitions *VM1_DET_FAIL*, *VM1_F* and *VM1_STRT* model fault detection time, failure and starting activities related to NC1 virtual machines (see Table 7.15). The association with the underlying infrastructure is carried out by immediate transitions *EXT1_FAIL* and *VM1_START*, and the respective guard conditions are shown in Table 7.16. It is important to state that the virtual machine stops working whenever the respective physical machine fails. To start a VM, the respective physical machine, NC, CC, CLC and front-end machine must be operational. Therefore, *EXT1_FAIL* fires if the respective physical machine fails. Transition *VM1_START* denotes the opposite idea in the sense that virtual machines start only if the required infrastructure is operational.

Table 7.15: Transitions of VM life-cycle model.

Transition	Delay	Description
VM1_F	VM_MTTF	VM failure
VM1_DET_FAIL	Detection_Time	VM fault detection
VM1_STRT	VM Start Time	VM Start

Variables X, M and N represent the number of requested VMs, the maximum number of running VMs on NC1 and NC2 servers, respectively. Considering the presented environment, the values of M and N are the same and equal to four. Availability is calculated based on the total amount of virtual machines running in both node controllers. Consequently, the availability is estimated as $P\{(\#VM1\_UP + \#VM2\_UP) \geq X\}$.

7.5.4 Scenarios and Results

Four validation scenarios (A1, A2, B1, and B2) are proposed to evaluate the experimental environment. Scenarios A1 and B1 require one running VM to consider the system operational. Scenarios A2 and B2 need two operational VMs to assume the system working. No experiment included a backup instance to user service and both employed exponential distributions to

Table 7.16: Condition to enable immediate transitions.

Transition	Condition
VM1_START	(#NC_HW1_UP>0) AND (#NC_SW1_UP>0) AND (#CC_UP>0) AND (#CLC_UP>0) AND (#FE_HW_UP>0) AND (#VM1_RDY>0) AND (#VM_REQUESTs>0)
EXT1_FAIL	(#NC_HW1_UP=0) AND (#VM1_UP+#VM1_STRTD) >0

represent mean time to fail (MTTF) and mean time to repair (MTTR). Finally, the results were compared with the respective results obtained from SPN models.

As stated before, faults were injected in all testbed components, including the physical machine hardware. This case study aim to evaluate the service availability (running VMs) considering the cloud environment behavior.

Scenarios A1 and A2 employs literature parameters (32) (accelerated by a constant factor) for Eucalyptus high-level components and hardware (see Table 7.17). In this table, MTTR experimental values are actually adopted in the Eucabomber. These values corresponds the the literature values (Real Values) divided by 60 (acceleration factor). On the other hand, MTTF values corresponds to the literature values divided by a $16,6$. These acceleration factor were introduce to speed up the evaluation process. Otherwise, the evaluation process would take a very long time to be performed.

Table 7.17: Parameters of Scenarios A1 and A2.

Component Type (Scenario A)	Experimental Values		Real Values	
	MTTF	MTTR	MTTF	MTTR
Hardware	31536 s	1.6 s	8760 h	100 min
Cloud Controller	2838 s	1 s	788 h	1 h
Cluster Controller	2838 s	1 s	788 h	1 h
Node Controller 1	2838 s	1 s	788 h	1 h
Node Controller 2	2838 s	1 s	788 h	1 h
Virtual Machines	10414 s	0.25 s	2893 h	15 min

Scenario B1 and B2 adopt hypothetical values where the MTTF is closer to the MTTR value. Therefore, a lower value for availability would be expected in this case.

Table 7.20: Availability evaluated from experiments.

Scenario	Number of Faults	Confidence interval of availability	Estimated value
A1	14	(0.9934 , 0.9986)	0.9935
A2	20	(0.9876 , 0.9935)	0.9879
B1	9	(0.8074 , 0.9658)	0.8279
B2	20	(0.6509 , 0.8676)	0.7921

Table 7.18: Parameters of Scenario B.

Component Type (Scenario B)	Experimental Values	
	MTTF	MTTR
Hardware	5600 s	1200 s
Cloud Controller	7200 s	1200 s
Cluster Controller	5600 s	600 s
Node Controller 1	3600 s	600 s
Node Controller 2	2700 s	300s
Virtual Machines	2700 s	900 s

Table 7.19 shows uptime and downtime results of the experiments executed over a 24 hour period. As expected, the scenarios that require more virtual machines to be operational have higher downtime values. This behavior is easily understood since each VM is another component that the user applications depends on and therefore failure here would further impact the total downtime. In this case, a comparison of B1 (dependent on one VM) with B2 (dependent on two VMs) exhibits an increase in downtime of over 266 percent.

Table 7.19: Up-time and Downtime from experiments.

Scenario	Number of VMs	Uptime	Downtime
A1	1	23.93 h	4.2 min
A2	2	23.89 h	6.6 min
B1	1	21.98 h	121.2 min
B2	2	18.66 h	320.4 min

Table 7.20 presents the comparison between the experiment results and the model evaluation. As predicted, Scenario B produced lower availability values when compared to the Scenario A. As the estimated results (SPN model) are contained in the Eucabomber results confidence interval, there is no evidence to reject the hypothesis that the results are equivalent for this case study.

7.6 Case Study VI

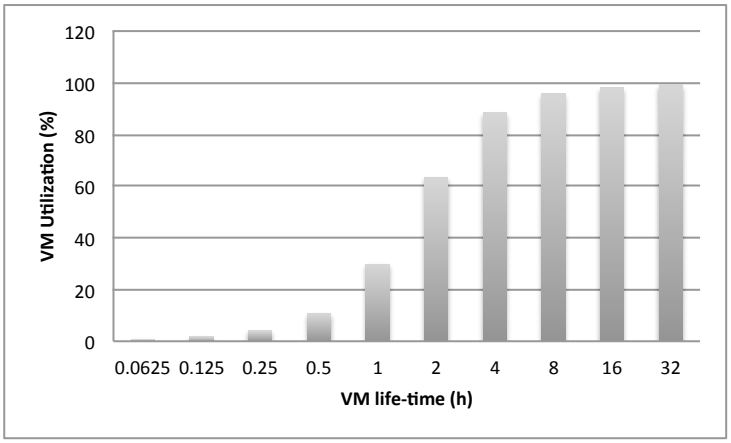
This case study presents an evaluation of a distributed cloud computing system considering performance issues. The objective of this case study is to compare the results of performance models and experiments. For this purpose, we constructed a script that simulates VM requests.

The experiments architecture is similar to the structure presented in Figure 7.7. In this case study, each node controller can run up to four VMs. Therefore, our experiment environment can execute at maximum eight VMs.

To simulate the user behavior, we created scripts to generate user requests and finalize the VM execution (VM release). As only performance issues were considered in this case study, failures and repairs are not generated. Ten clients are considered in this case study, and each client can request a single VM. When the VM execution is finished, a new VM can be requested. We assume the mean time between requests is one hour. The assumed time to start a VM is one minute.

Two metrics are considered to verify system performance, VM utilization and execution throughput (See Section 4.3.3). The performance model is analogous to the model presented in Figure 5.12 but with only two VM performance components and one data center. Figures 7.10 and 7.11 presents the results of performance related to aforementioned model.

Figure 7.10: VM Utilization results of performance model.



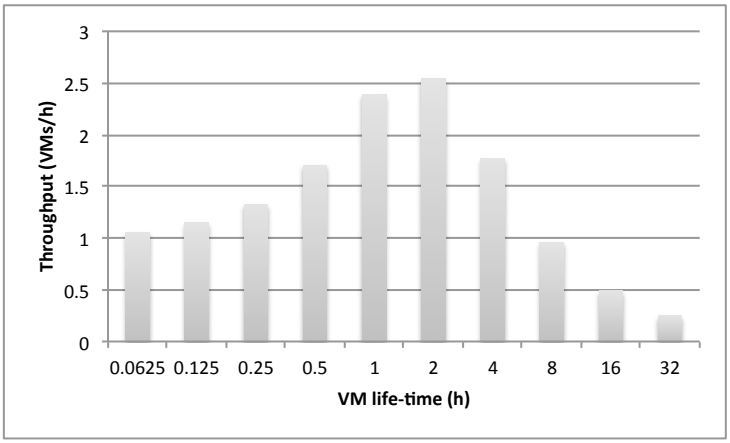
Source: Made by author.

Regarding the chart presented in Figure 7.10, the VM utilization metric presents values close to 100% for VM life-time value is higher than four hours. When the life-time value is getting close to the VM request time (one hour), the utilization values decrease drastically. In this case, if the user-load usually presents VM life-time values higher than four hours, it is recommended to acquire more machines to provide a less stressed service.

Figure 7.11 presents the number of VMs executed per hour. In this chart, when the VM life-time is high, the VM throughput value is low as a running VM takes a long to leave the cloud. However, when the VM life-time is getting low, the VM throughput becomes close to one VM per hour (request mean time). For instance, when the VM life-time is 0.0006 the VM throughput is 1.0004. Based on this chart, we can conclude that the highest throughput value occurs when the VM life-time is close to two hours.

Table 7.21 presents the comparison between the experiment results and the model evaluation. We conducted experiments of 72 hours for 32 and 16 hours of life-time. The rest

Figure 7.11: VM throughput results of performance model.



Source: Made by author.

of the experiments taken 24 four hours. The experiments were evaluated by adopting a script analogous to the previous case study (by using heart beat functions). As the model results (SPN model) are contained in the experiment results confidence interval, there is no evidence to reject the hypothesis that the results are equivalent for this case study.

Table 7.21: Utilization and Execution throughput values for different VM life-times.

VM life-time (h)	Execution Throughtput (VMs/h) Model	Execution TP (VMs/h) Experiments	VM Util (%)	VM Util (%) Experiments
32	0.2478	(0.2335, 0.2581)	99.154	(99.107, 99.306)
16	0.4908	(0.4682, 0.5175)	98.163	(97.770, 97.965)
8	0.9578	(0.9129, 1.0090)	95.785	(95.762, 95.954)
4	1.7697	(1.6968, 1.8755)	88.487	(88.201, 88.377)
2	2.5390	(2.4275, 2.6830)	63.475	(62.941, 63.067)
1	2.3898	(2.2773, 2.5170)	29.873	(28.857, 30.117)
0.5	1.7061	(1.6202, 1.7907)	10.663	(10.553, 10.574)
0.25	1.3241	(1.2657, 1.3989)	4.1379	(4.1614, 4.1698)
0.125	1.1477	(1.0845, 1.1987)	1.7934	(1.8057, 1.8093)
0.0625	1.0523	(1.0050, 1.1108)	0.8221	(0.8294, 0.8311)

7.7 Final Remarks

This chapter presented several case studies to illustrate the feasibility of the proposed framework. Modeling and experiments were provided to provide results related to availability, performance and survivability evaluation of disaster tolerant clouds computing systems.

8 Conclusion

With increased dependence on computing services, cloud computing performance, dependability and survivability has become serious requirements. For companies that heavily depend on the Internet for their operations, service outages can be very expensive, easily running into millions of dollars per hour.

A widely adopted design principle to provide cloud computing survivability is to introduce geographical distribution of data centers to mitigate the impact of disasters. However, geographical distances leads to additional time to transmit VM data and may affect system recovery. At present stage, system designers do not have many mechanisms to support the integrated performance, dependability and survivability evaluation of data center infrastructures.

This work aims at reducing this gap by proposing a framework (GeoClouds Modcs) to evaluate performance, dependability and survivability of cloud computing systems deployed into geographically distributed data centers taking into account disaster occurrence. The framework allows the impact assessment of disaster occurrence, VM transmission time and different client loads on system performability. The proposed approach also allows survivability assessment of cloud systems taking into account the distance between data centers, RPO and RTO requirements. The following sections highlight the thesis' contributions and future works that will be conducted based on the results presented in this document.

8.1 Contributions

This document presented a set of formal models to allow the integrated evaluation of dependability, performance and survivability issues on cloud computing. The main contribution of this thesis corresponds to the set of models and the modeling approach to evaluate performance, availability and survivability in the context of disaster tolerant cloud computing systems. To accomplish that objective, a set of activities intermediate activities were conducted and the main points are detailed as follows.

The proposed modeling strategy also takes the advantage of both RBD and SPN formalism. For instance, to estimate dependability metrics, RBD submodels can be created to mitigate the system complexity. RBD considers closed-form equations, the results are exact and usually obtained faster than using SPN evaluation. However, to represent the final model which is not trivial, SPN represent a more feasible modeling choice. The default approach to evaluate the proposed SPN models is based on simulation. However, the models can be assessed by adopting

numerical techniques such as SPN analysis (26).

A high-level model is proposed to precisely define the system structure and the interactions between the cloud computing components. Important aspects of a disaster tolerant cloud computing are described. For instance, the data transmission is represented by a mathematical function and is adopted to evaluate the time to recover the cloud system after a disaster. Based on this formal specification, cloud computing designers are able to create the evaluation models or adopt tools (e.g., GeoClouds Modcs) to generate them automatically.

Two algorithms are proposed to translate high-level models to performance, and dependability models. This translation process allows non-specialized users to automatically create evaluation models. Therefore, the system designer needs to provide the system structure and parameters and the evaluation can be done transparently through GeoClouds Modcs tools. We proof that the generated models are always structurally conservative and bounded by providing an inductive demonstration. In this way, the models can scale to represent larger cloud computing infrastructures. An evaluation considering large data center systems is presented in Case Study III.

This document has proposed GeoClouds Modcs tool that considers RBD, SPN and high level models, to support distributed cloud evaluation. The tool adopts Java programming language, therefore it can be executed in any operating system that runs Java Virtual Machine. The proposed tool executes in conjunction with Mercury tool that is responsible for evaluating the generated evaluation models. An interesting features of GeoClouds corresponds to the exporting of generated models to TimeNET and Mercury. Therefore, users with modeling skills can edit the models to consider a more advanced evaluation.

Eucabomber 2.0 was proposed to support dependability studies on cloud platforms using fault injection techniques. The tool generates faults events and the system behavior is monitored to check system dependability and the system models can be validated. Additionally, several case studies were proposed to demonstrate the feasibility of the proposed framework.

8.2 Future works

Although this work tackles some issues regarding dependability, performance and survivability for distributed cloud systems, there are many possibilities to improve and extend the current work. The following items summarize some possibilities:

- In this work, we adopt one backup server to provide cloud system backup in case of disaster. However, other backup policies (e.g., multiple backup servers) can be adopt to provide VM data redundancy. Future works can evaluate various backup strategies and assess the impact of these approaches on the system metrics.
- As a future work, we can adopt the evaluation models to estimate the costs related to data center allocation. As the energy cost may vary from place to place, the data center allocation may represent a significant aspect of cloud system.

- An evaluation considering different priorities for data center user utilization can adopted in future works. In this case, users may preferentially utilize an specific data center rather than others, and in case of disasters the users may adopt a different data center.

- Eucabomber tool is designed to evaluate system dependability in cloud infrastructures. However, it can be extended to estimate performance and survivability by generating system load and disaster events.

- This work can be also extended to consider maintenance policies as well as different service level agreements (SLAs). For instance, the operational cost can consider different maintenance policies, as well as fines associated in case the contracted maintenance company does not provide the required availability.

8.3 Final remarks

This work presented GeoClouds Modcs Framework for performance, dependability and survivability evaluation of cloud computing systems deployed into geographically distributed data centers taking into account disaster occurrence. Modeling strategies, specification models, automatic evaluation model generation and evaluation tools are important contributions of this work. The proposed framework allows the impact assessment of disaster occurrence, VM transmission time and different client loads on system performability. The case studies results demonstrate the different cloud configurations on performance, survivability and dependability metrics taking into account disaster occurrence.

Academic works are extending this work by proposing a more detailed evaluation of survivability scenarios. For instance, the possibility of having more than one backup server and the impact of this allocation in the system metrics.

References

[1] MELL, P.; GRANCE, T. Draft nist working definition of cloud computing. *Referenced on June. 3rd*, v. 15, p. 32, 2009.

[2] ARMBRUST, M. et al. A view of cloud computing. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782.

[3] BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009.

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

[5] MENASCé, D. A.; NGO, P. *Understanding Cloud Computing: Experimentation and Capacity Planning*. 2009.

[6] DILLON, T.; WU, C.; CHANG, E. Cloud computing: issues and challenges. In: IEEE. *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. [S.l.], 2010. p. 27–33.

[7] SALESFORCE website. Disponível em: <http://www.salesforce.com>.

[8] GOOGLE App services. Disponível em: <https://www.google.com/about/products>.

[9] MICROSOFT office online. Disponível em: <https://www.office.com/>.

[10] GOOGLE App engine. Disponível em: <https://cloud.google.com/appengine>.

[11] AWS Elastic Beans Talk Cloud. Disponível em: <https://aws.amazon.com/elasticbeanstalk>.

[12] MENDIX scalable platform for the rapid development. Disponível em: <https://www.mendix.com>.

[13] HEROKU | Cloud Application Platform. Disponível em: <https://www.heroku.com>.

[14] GARTNER: Forecast Analysis: Public Cloud Services, Worldwide. 2014. Disponível em: <https://www.gartner.com/doc/3027317/forecast-analysis-public-cloud-services>.

[15] AMAZON EC2. Disponível em: <http://aws.amazon.com/ec2>.

[16] IBM Smart Business Cloud. Disponível em: <http://www-935.ibm.com/services/us/igs/cloud-development/>.

[17] PATEL, P.; RANABAHU, A.; SHETH, A. Service level agreement in cloud computing. 2009.

[18] TAKABI, H.; JOSHI, J. B.; AHN, G.-J. Security and privacy challenges in cloud computing environments. *Security & Privacy, IEEE*, IEEE, v. 8, n. 6, p. 24–31, 2010.

[19] IDG Enterprise Cloud Computing Study. 2012. Disponível em: <http://www.idgenterprise.com/report/idg-enterprises-cloud-computing>.

[20] THOMAS, V.; LÓPEZ, R. Global increase in climate-related disasters. *Asian Development Bank Economics Working Paper Series*, n. 466, 2015.

[21] BOSSE, S.; SPLIETH, M.; TUROWSKI, K. Optimizing IT service costs with respect to the availability service level objective. In: IEEE. *Availability, Reliability and Security (ARES), 2015 10th International Conference on*. [S.l.], 2015. p. 20–29.

[22] LI, Z. et al. The cloud's cloudy moment: A systematic survey of public cloud service outage. *arXiv preprint arXiv:1312.6485*, 2013.

[23] RISTOV, S. et al. Business continuity challenges in cloud computing. *ICT Innovations 2011, Web Proceedings ISSN 1857-7288*, Citeseer, p. 149, 2012.

[24] ALEXANDER, P. *Information Security: A Manager's Guide to Thwarting Data Thieves and Hackers*. [S.l.]: ABC-CLIO, 2008.

[25] SHELKOVNYKOV, P.; BOTULINSKIY, S. Improving microsoft hyper-v live migration efficiency over distance. In: *IEEE International Crimean Conference on Microwave and Telecommunication Technology*. [S.l.: s.n.], 2011. p. 12–16.

[26] MACIEL, P. et al. Performance and dependability in service computing: Concepts, techniques and research directions. In: ____. [S.l.]: Igi Global, 2011. (Premier Reference Source), cap. Dependability Modeling.

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

[28] GHOSH, R. et al. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In: *Proceedings of the 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (PRDC '10), p. 125–132. ISBN 978-0-7695-4289-8.

[29] BRUNEO, D. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems. *Parallel and Distributed Systems, IEEE Transactions on*, IEEE, v. 25, n. 3, p. 560–569, 2014.

[30] BRADFORD, R. et al. Live wide-area migration of virtual machines including local persistent state. In: *Proceedings of the 3rd international conference on Virtual execution environments*. New York, NY, USA: ACM, 2007. (VEE '07), p. 169–179. ISBN 978-1-59593-630-1. Disponível em: <http://doi.acm.org/10.1145/1254810.1254834>.

[31] VOORSLUYS, W. et al. Cost of virtual machine live migration in clouds: A performance evaluation. In: *Proceedings of the 1st International Conference on Cloud Computing*. Berlin, Heidelberg: Springer-Verlag, 2009. (CloudCom '09), p. 254–265. ISBN 978-3-642-10664-4.

[32] 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*. [S.l.: s.n.], 2012. p. 1664 –1669.

[33] MATOS, R. de S.; ANDRADE, E. C.; MACIEL, P. R. M. Evaluation of a disaster recovery solution through fault injection experiments. In: *2014 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2014, San Diego, CA, USA, October 5-8, 2014*. [s.n.], 2014. p. 2675–2680. Disponível em: <http://dx.doi.org/10.1109/SMC.2014.6974331>.

[34] KAUSHIK, R. T.; BHANDARKAR, M.; NAHRSTEDT, K. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. [S.l.], 2010. p. 274–287.

[35] KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability modeling and analysis of a virtualized system. In: *Proceedings of the 2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*. Washington, DC, USA: IEEE Computer Society, 2009. (PRDC '09), p. 365–371. ISBN 978-0-7695-3849-5. Disponível em: <http://dx.doi.org/10.1109/PRDC.2009.64>.

[36] MATOS, R. et al. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, 2014. ISSN 1569-190X.

[37] SUN, D. et al. A dependability model to enhance security of cloud environment using system-level virtualization techniques. In: IEEE. *Pervasive Computing Signal Processing and Applications (PCSPA), 2010 First International Conference on*. [S.l.], 2010. p. 305–310.

[38] BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In: IEEE. *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*. [S.l.], 2009. p. 1–11.

[39] CLOTH, L.; HAVERKORT, B. R. Model checking for survivability! In: IEEE. *Quantitative Evaluation of Systems, 2005. Second International Conference on the*. [S.l.], 2005. p. 145–154.

[40] XU, X. et al. Availability analysis for deployment of in-cloud applications. In: *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems*. New York, NY, USA: ACM, 2013. (ISARCS '13), p. 11–16. ISBN 978-1-4503-2123-5. Disponível em: <http://doi.acm.org/10.1145/2465470.2465472>.

[41] LENK, A.; TAI, S. Cloud standby: disaster recovery of distributed systems in the cloud. In: *Service-Oriented and Cloud Computing*. [S.l.]: Springer, 2014. p. 32–46.

[42] ZHANG, Y.; LIU, B.; ZHOU, Q. A dynamic software binary fault injection system for real-time embedded software. In: IEEE. *Reliability, Maintainability and Safety (ICRMS), 2011 9th International Conference on*. [S.l.], 2011. p. 676–680.

[43] GUAN, Q.; CHIU, C.-C.; FU, S. CDA: a cloud dependability analysis framework for characterizing system dependability in cloud computing infrastructures. In: IEEE. *Dependable Computing (PRDC), 2012 IEEE 18th Pacific Rim International Symposium on*. [S.l.], 2012. p. 11–20.

[44] COOPER, B. F. et al. Benchmarking cloud serving systems with YCSB. In: ACM. *Proceedings of the 1st ACM symposium on Cloud computing*. [S.l.], 2010. p. 143–154.

[45] SOUZA, D. et al. A tool for automatic dependability test in eucalyptus cloud computing infrastructures. *Computer and Information Science*, Canadian Center of Science and Education, v. 6, n. 3, p. 57–67, 2013.

[46] ARAUJO, J. et al. Experimental evaluation of software aging effects on the Eucalyptus cloud computing infrastructure. In: *Proceedings of the Middleware 2011 Industry Track Workshop*. New York, NY, USA: ACM, 2011. (Middleware '11), p. 4:1–4:7. ISBN 978-1-4503-1074-1. Disponível em: <http://doi.acm.org/10.1145/2090181.2090185>.

[47] OPEN Source Private and Hybrid Clouds from Eucalyptus. Disponível em: <http://www.eucalyptus.com>.

[48] BANZAI, T. et al. D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2010. (CCGRID '10), p. 631–636. ISBN 978-0-7695-4039-9.

[49] CLARK, C. et al. Live migration of virtual machines. In: *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*. Berkeley, CA, USA: USENIX Association, 2005. (NSDI'05), p. 273–286. Disponível em: <http://dl.acm.org/citation.cfm?id=1251203.1251223>.

[50] MEYER, J. *On Evaluating the Performability of Degradable Computing Systems*. [S.l.: s.n.], 1978. (R).

[51] JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. [S.l.]: Wiley, 1991. (Wiley Professional Computing). ISBN 9780471503361.

[52] AVIZIENIS, A.; LAPRIE, J.; RANDELL, B. Fundamental Concepts of Dependability. *Technical Report Series-University of Newcastle upon Tyne Computing Science*, 2001.

[53] EBELING, C. *An Introduction to Reliability and Maintainability Engineering*. [S.l.]: Waveland Press, 1997.

[54] BRITISH STANSDARD. *Business Continuity Management: BS25999-1*. 2006. 2–3 p. Disponível em: <https://www.bsigroup.com/>.

[55] WOOD, T. et al. Disaster recovery as a cloud service: Economic benefits & deployment challenges. In: *2nd USENIX Workshop on Hot Topics in Cloud Computing*. [S.l.: s.n.], 2010.

[56] KEETON, K. et al. Designing for disasters. In: *FAST*. [S.l.: s.n.], 2004. v. 4, p. 59–62.

[57] KUO, W.; ZUO, M. J. *Optimal Reliability Modeling - Principles and Applications*. [S.l.]: Wiley, 2003. ISBN 047139761X.

[58] MURATA, T. Petri nets: Properties, analysis and applications. *Proc. IEEE*, v. 77, n. 4, p. 541–580, April 1989.

[59] MARSAN, A. *Modelling with generalized stochastic Petri nets*. [S.l.]: Wiley, 1995. (Wiley series in parallel computing). ISBN 9780471930594.

[60] TRIVEDI, K. *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. 2. ed. [S.l.]: Wiley Interscience Publication, 2002.

[61] GERMAN, R. *Performance Analysis of Communication Systems with Non-Markovian Stochastic Petri Nets*. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[62] SILVA, B.; MACIEL, P. R. M.; ZIMMERMANN, A. Geoclouds modcs: A performability evaluation tool for disaster tolerant IaaS clouds. In: *8th annual ieee international systems conference*. [S.l.: s.n.], 2013.

[63] BARBALHO, D. S. *Conception et mise en oeuvre de la fonction coordination pour une commande distribuee d'atelier 2F*. Tese (Doutorado) — Toulouse 3, 1987. Th. : informatique indus. Disponível em: <http://opac.inria.fr/record=b1056579>.

[64] TAVARES, E. G. *Software Synthesis for Energy-Constrained Hard Real-Time Embedded Systems*. Tese (Doutorado) — UFPE, 2009.

[65] DESROCHERS, A. A.; AL-JAAR, R. Y. *Applications of Petri nets in manufacturing systems: modeling, control, and performance analysis*. [S.l.]: IEEE press Piscataway^ eNJ NJ, 1995.

[66] ARLAT, J. et al. Fault injection and dependability evaluation of fault-tolerant systems. *Computers, IEEE Transactions on*, IEEE, v. 42, n. 8, p. 913–923, 1993.

[67] YU, Y.; BASTIEN, B.; JOHNSON, B. W. A state of research review on fault injection techniques and a case study. In: IEEE. *Reliability and Maintainability Symposium, 2005. Proceedings. Annual*. [S.l.], 2005. p. 386–392.

[68] HSUEH, M.-C.; TSAI, T. K.; IYER, R. K. Fault injection techniques and tools. *Computer*, IEEE, v. 30, n. 4, p. 75–82, 1997.

[69] WANG, W.; KECECIOGLU, D. B. Confidence limits on the inherent availability of equipment. In: IEEE. *Reliability and Maintainability Symposium, 2000. Proceedings. Annual*. [S.l.], 2000. p. 162–168.

[70] SILVA, B. et al. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Industry Track at 45th Dependable Systems and Networks Conference (DSN-2015)*. [S.l.: s.n.], 2015.

[71] SILVA, B. et al. Astro: A tool for dependability evaluation of data center infrastructures. In: *SMC'10*. [S.l.: s.n.], 2010. p. 783–790.

[72] GERMAN, R. et al. TimeNET - a toolkit for evaluating non-markovian stochastic petri nets. In: . [S.l.: s.n.], 1995. v. 24, p. 69–87.

[73] MATOS, R. et al. Sensitivity analysis of a hierarchical model of mobile cloud computing. *Simulation Modelling Practice and Theory*, Elsevier, v. 50, p. 151–164, 2015.

[74] MARSAN, M. et al. *Modelling with Generalized Stochastic Petri Nets*. [S.l.]: ACM Press New York, NY, USA, 1998.

[75] MATHIS, M. et al. The macroscopic behavior of the TCP congestion avoidance algorithm. In: . New York, NY, USA: ACM, 1997. v. 27, n. 3, p. 67–82. ISSN 0146-4833.

[76] YANG, Q.; XIAO, W.; REN, J. Trap-array: A disk array architecture providing timely recovery to any point-in-time. In: IEEE COMPUTER SOCIETY. *ACM SIGARCH Computer Architecture News*. [S.l.], 2006. v. 34, n. 2, p. 289–301.

[77] SILVA, B. et al. Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable Computing: Informatics and Systems*, 2012. ISSN 2210-5379.

[78] JAVA website. Disponível em: <https://www.java.com/en/>.

[79] JAVA Virtual Machine Specification. Disponível em: <http://docs.oracle.com/javase/specs/jvms/se7/html/>.

[80] GOOGLE Maps API. Disponível em: <https://developers.google.com/maps/documentation/javascript/tutorial>.

[81] COTTRELL, W. M. L.; LOGG, C. *Tutorial on Internet Monitoring and PingER at SLAC*. [S.l.], 1996. Disponível em: <http://www.slac.stanford.edu/comp/net/wan-mon/tutorial.html>.

[82] JAVA XStream. 2015. Disponível em: <http://xstream.codehaus.org/>.

[83] GALINDO, H. E. S. et al. WGCap: A synthetic trace generation tool for capacity planning of virtual server environments. In: IEEE. *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. [S.l.], 2010. p. 2094–2101.

[84] SSH2 Ganymed for java. Disponível em: <http://www.ganymed.ethz.ch/ssh2/>.

[85] YLONEN, T.; LONVICK, C. The secure shell (SSH) protocol architecture. 2006.

[86] LIEBERMAN, P. *Wake-on-LAN technology*. 2010.

[87] CHISNALL, D. *The definitive guide to the Xen hypervisor*. [S.l.]: Pearson Education, 2007.

[88] ONLINE Distance Calculator. 2016. Disponível em: <http://www.distance-cities.com>.

[89] SILVA, B.; MACIEL, P. R. M.; ZIMMERMANN, A. Dependability models for designing disaster tolerant cloud computing systems. In: *The Third International Workshop on Dependability of Clouds, Data Centers and Virtual Machine Technology (DCDV)*. [S.l.: s.n.], 2013.

[90] AMAZON Web Service SDK for Java. Disponível em: <http://aws.amazon.com/pt/sdkforjava/>.

[91] EUCALYPTUS Cloud Computing System. Disponível em: <http://www.eucalyptus.com/>.

[92] FALIVA, M.; ZOIA, M. *Dynamic Model Analysis: Advanced Matrix Methods and Unit-Root Econometrics Representation Theorems*. Springer Berlin Heidelberg, 2008. (Business and Economics). ISBN 9783540859963. Disponível em: <https://books.google.com.br/books?id=8J-usmKLcnsC>.

Appendix

A

Juxtaposition of P-Invariants

The adopted modeling approach utilize models that are structurally bounded and conservative. To prove that properties, it is necessary to demonstrate that the combination of basic SPN components generate structurally conservative as well as structurally bounded models. The preservation of these properties are demonstrated by using juxtaposition of P-invariants. From now on, assume that all variables in each vector corresponds to positive integers.

A.1 Base Case: Juxtaposition of P-Invariants

The following section present the resulting P-invariant related to the net union of two SPN components. It is demonstrated that the resulting SPN is structurally conservative and bounded.

A.1.1 Two Dependability Components

In our approach, there are two possibilities of merging two dependability components. The dependability components can be related to servers of the same data center or different data centers. If the components are related servers of different data center there will be no common place or transition between them. In this case, the union of these SPNs will be structurally bounded and conservative as net union of disjoint nets preserves the P-invariants of each subnet in the generated model (64). Therefore, this section presents only the P-Invariants of merged dependability submodels ($\mathscr{N}_{(1)} and \mathscr{N}_{(2)}$) related to servers from the same data center.

$$\mathscr{I}_{(dp)(1)} = \begin{bmatrix} \overset{VM\_UP_{(1)}}{dp_{2(1)}+dp_{1(1)}} & \overset{VM\_DOWN_{(1)}}{dp_{1(1)}} & \overset{VM\_WAIT_{(1)}}{dp_{1(1)}} & \overset{VM\_STRTD_{(1)}}{dp_{2(1)}+dp_{1(1)}} & \overset{CHC\_RES}{dp_{2(1)}} \end{bmatrix}^T.$$

$$\mathscr{I}_{(dp)(2)} = \begin{bmatrix} \overset{VM\_UP_{(2)}}{dp_{2(2)}+dp_{1(2)}} & \overset{VM\_DOWN_{(2)}}{dp_{1(2)}} & \overset{VM\_WAIT_{(2)}}{dp_{1(2)}} & \overset{VM\_STRTD_{(2)}}{dp_{2(2)}+dp_{1(2)}} & \overset{CHC\_RES}{dp_{2(2)}} \end{bmatrix}^T.$$

$\mathscr{N}_{(1)\sqcup(2)} = \mathscr{N}_{(1)} \sqcup \mathscr{N}_{(2)}$. By juxtaposition $\mathscr{I}_{(1)\sqcup(2)} = \mathscr{J}\left(\mathscr{I}_{(dp)(1)}, \mathscr{I}_{(dp)(1)}\right)$ and $dp_{2(2)} = dp_{2(1)} = dp$:

$$\mathscr{I}_{(1)\sqcup(2)} = \begin{bmatrix} \overset{VM\_UP_{(1)}}{dp+dp_{1(1)}} & \overset{VM\_DOWN_{(1)}}{dp_{1(1)}} & \overset{VM\_WAIT_{(1)}}{dp_{1(1)}} & \overset{VM\_STRTD_{(1)}}{dp+dp_{1(1)}} \end{bmatrix}$$

$$\begin{matrix} \overset{VM\_UP_{(2)}}{} & \overset{VM\_DOWN_{(2)}}{} & \overset{VM\_WAIT_{(2)}}{} & \overset{VM\_STRTD_{(2)}}{} & \overset{CHC\_RES}{} \\ dp+dp_{1(1)} & dp_{1(1)} & dp_{1(1)} & dp+dp_{1(1)} & dp \end{matrix} \Big]^T$$

Since $\mathscr{I}_{(1)\sqcup(2)} > 0$ and $\mathscr{I}^T_{(1)\sqcup(2)} \times A_{(1)\sqcup(2)} = 0$, in which $A_{(1)\sqcup(2)}$ is the incidence matrix, $\mathscr{N}_{(1)\sqcup(2)}$ is structurally conservative as well as structurally bounded.

A.1.2 SPN Dependability Component and Transmission Component

In this example, a SPN Dependability Component ($\mathscr{N}_{(1)}$) and a transmission component ($\mathscr{N}_{(2)}$) are merged.

$$\mathscr{I}_{(dp)} = \begin{bmatrix} \overset{VM\_UP}{} & \overset{VM\_DOWN}{} & \overset{VM\_WAIT}{} & \overset{VM\_STRTD}{} & \overset{CHC\_RES}{} \\ dp_2+dp_1 & dp_1 & dp_1 & dp_2+dp_1 & dp_2 \end{bmatrix}^T.$$

$$\mathscr{I}_{(tr)} = \begin{bmatrix} \overset{CHC\_RES}{} & \overset{TRF\_P1}{} & \overset{TRF\_P2}{} & \overset{TBK\_P1}{} & \overset{TBK\_P2}{} & \overset{CHC\_RES2}{} \\ tr & tr & tr & tr & tr & tr \end{bmatrix}^T.$$

$\mathscr{N}_{(1)\sqcup(2)} = \mathscr{N}_{(2)} \sqcup \mathscr{N}_{(1)}$. By juxtaposition $\mathscr{I}_{(1)\sqcup(2)} = \mathscr{J}(\mathscr{I}_{(dp)}, \mathscr{I}_{(tr)})$ and $dp_2 = tr = dp$:

$$\begin{matrix} \overset{VM\_UP}{} & \overset{VM\_DOWN}{} & \overset{VM\_WAIT}{} & \overset{VM\_STRTD}{} & \overset{CHC\_RES}{} \\ dp+dp_1 & dp_1 & dp_1 & dp+dp_1 & dp \end{matrix}$$

$$\mathscr{I}_{(1)\sqcup(2)} = \Big[$$

$$\begin{matrix} \overset{TRF\_P1}{} & \overset{TRF\_P2}{} & \overset{TBK\_P1}{} & \overset{TBK\_P2}{} & \overset{CHC\_RES2}{} \\ dp & dp & dp & dp & dp \end{matrix} \Big]^T$$

Since $\mathscr{I}_{(1)\sqcup(2)} > 0$ and $\mathscr{I}^T_{(1)\sqcup(2)} \times A_{(1)\sqcup(2)} = 0$, in which $A_{(1)\sqcup(2)}$ is the incidence matrix, $\mathscr{N}_{(1)\sqcup(2)}$ is structurally conservative as well as structurally bounded.

A.1.3 Two Performability Components

Just like the union of dependability components, this section presents the P-Invariants of merged performability submodels ($\mathscr{N}_{(1)} and \mathscr{N}_{(2)}$) related to servers from the same data center.

$\mathscr{I}_{(pb)(1)} =$

$$\begin{bmatrix} \overset{CLTS}{} & \overset{DC\_CHC}{} & \overset{CHC\_RES}{} & \overset{VM\_DOWN_{(1)}}{} & \overset{WM\_WAIT_{(1)}}{} & \overset{VM\_STRTD_{(1)}}{} & \overset{VM\_UP_{(1)}}{} \\ pb_{2(1)} & pb_{2(1)} & pb_{2(1)} & pb_{1(1)} & pb_{1(1)} & pb_{1(1)}+pb_{2(1)} & pb_{1(1)}+pb_{2(1)} \end{bmatrix}^T.$$

$\mathscr{I}_{(pb)(2)} =$

$$\begin{bmatrix} \overset{CLTS}{} & \overset{DC\_CHC}{} & \overset{CHC\_RES}{} & \overset{VM\_DOWN_{(2)}}{} & \overset{WM\_WAIT_{(2)}}{} & \overset{VM\_STRTD_{(2)}}{} & \overset{VM\_UP_{(2)}}{} \\ pb_{2(2)} & pb_{2(2)} & pb_{2(2)} & pb_{1(2)} & pb_{1(2)} & pb_{1(2)}+pb_{2(2)} & pb_{1(2)}+pb_{2(2)} \end{bmatrix}^T.$$

$\mathscr{N}_{(1)\sqcup(2)} = \mathscr{N}_{(2)} \sqcup \mathscr{N}_{(1)}$. By juxtaposition $\mathscr{I}_{(1)\sqcup(2)} = \mathscr{J}(\mathscr{I}_{(pb)(1)}, \mathscr{I}_{(pb)(2)})$ and $pb_{2(1)} = pb_{2(2)} = pb$:

$$\mathscr{I}_{(1)\sqcup(2)} = \begin{bmatrix} \overset{VM\_DOWN_{(1)}}{pb_{1(2)}} & \overset{WM\_WAIT_{(1)}}{pb_{1(1)}} & \overset{VM\_STRTD_{(1)}}{pb_{1(1)}+pb} & \overset{VM\_UP_{(1)}}{pb_{1(1)}+pb} & \overset{VM\_DOWN_{(2)}}{pb_{1(2)}} \\ \overset{WM\_WAIT_{(2)}}{pb_{1(2)}} & \overset{VM\_STRTD_{(2)}}{pb_{1(2)}+pb} & \overset{VM\_UP_{(2)}}{pb_{1(2)}+pb} & \overset{DC\_CHC}{pb} & \overset{CHC\_RES}{pb} & \overset{CLTS}{pb} \end{bmatrix}$$

Since $\mathscr{I}_{(1)\sqcup(2)} > 0$ and $\mathscr{I}^T_{(1)\sqcup(2)} \times A_{(1)\sqcup(2)} = 0$, in which $A_{(1)\sqcup(2)}$ is the incidence matrix, $\mathscr{N}_{(1)\sqcup(2)}$ is structurally conservative as well as structurally bounded.

A.1.4 SPN Performability Component and Transmission Component

In this example, a performability component ($\mathscr{N}_{(1)}$) and a transmission component ($\mathscr{N}_{(2)}$) are merged.

$\mathscr{I}_{(pb)}$ =

$$\begin{bmatrix} \overset{CLTS}{pb_2} & \overset{DC\_CHC}{pb_2} & \overset{CHC\_RES}{pb_2} & \overset{VM\_DOWN}{pb_1} & \overset{WM\_WAIT}{pb_1} & \overset{VM\_STRTD}{pb_1+pb_2} & \overset{VM\_UP}{pb_1+pb_2} \end{bmatrix}^T.$$

$$\mathscr{I}_{(tr)} = \begin{bmatrix} \overset{CHC\_RES}{tr} & \overset{TRF\_P1}{tr} & \overset{TRF\_P2}{tr} & \overset{TBK\_P1}{tr} & \overset{TBK\_P2}{tr} & \overset{CHC\_RES2}{tr} \end{bmatrix}^T.$$

$\mathscr{N}_{(1)\sqcup(2)}$ = $\mathscr{N}_{(2)} \sqcup \mathscr{N}_{(1)}$. By juxtaposition $\mathscr{I}_{(1)\sqcup(2)} = \mathscr{J}(\mathscr{I}_{(pb)}, \mathscr{I}_{(tr)})$ and $pb_2 = tr = pb$:

$$\mathscr{I}_{(1)\sqcup(2)} = \begin{bmatrix} \overset{CLTS}{pb} & \overset{DC\_CHC}{pb} & \overset{VM\_DOWN}{pb_1} & \overset{WM\_WAIT}{pb_1} & \overset{VM\_STRTD}{pb_1+pb} & \overset{VM\_UP}{pb_1+pb} \\ \overset{CHC\_RES}{pb} & \overset{TRF\_P1}{pb} & \overset{TRF\_P2}{pb} & \overset{TBK\_P1}{pb} & \overset{TBK\_P2}{pb} & \overset{CHC\_RES2}{pb} \end{bmatrix}^T$$

Since $\mathscr{I}_{(1)\sqcup(2)} > 0$ and $\mathscr{I}^T_{(1)\sqcup(2)} \times A_{(1)\sqcup(2)} = 0$, in which $A_{(1)\sqcup(2)}$ is the incidence matrix, $\mathscr{N}_{(1)\sqcup(2)}$ is structurally conservative as well as structurally bounded.

A.1.5 Two Performance Components

Similar to the union of dependability and performability components, this section presents the P-Invariants of merged performance submodels ($\mathscr{N}_{(1)} and \mathscr{N}_{(2)}$) related to servers from the same data center.

$\mathscr{I}_{(pf)(1)}$ =

$$\begin{bmatrix} \overset{CLTS}{pf_{1(1)}} & \overset{DC\_CHC}{pf_{1(1)}} & \overset{CHC\_RES}{pf_{1(1)}} & \overset{WM\_WAIT_{(1)}}{pf_{2(1)}} & \overset{VM\_STRTD_{(1)}}{pf_{1(1)}+pf_{2(1)}} & \overset{VM\_UP_{(1)}}{pf_{1(1)}+pf_{2(1)}} \end{bmatrix}^T.$$

$$\mathscr{I}_{(pf)(2)} =$$

$$\begin{matrix} CLTS & DC\_CHC & CHC\_RES & WM\_WAIT_{(2)} & VM\_STRTD_{(2)} & VM\_UP_{(2)} \\ \left[\begin{matrix} pf_{1(2)} & pf_{1(2)} & pf_{1(2)} & pf_{2(2)} & \begin{matrix} pf_{1(2)}+ \\ pf_{2(2)} \end{matrix} & \begin{matrix} pf_{1(2)}+ \\ pf_{2(2)} \end{matrix} \end{matrix} \right]^T. \end{matrix}$$

$\mathscr{N}_{(1)\sqcup(2)} = \mathscr{N}_{(2)} \sqcup \mathscr{N}_{(1)}$. By juxtaposition $\mathscr{I}_{(1)\sqcup(2)} = \mathscr{J}(\mathscr{I}_{(pf)(1)}, \mathscr{I}_{(pf)(2)})$ and $pf_{2(1)} = pf_{2(2)} = pf$:

$$\mathscr{I}_{(1)\sqcup(2)} = \begin{matrix} & WM\_WAIT_{(1)} & VM\_STRTD_{(1)} & VM\_UP_{(1)} & WM\_WAIT_{(2)} \\ pf_{1(1)} & \left[\begin{matrix} pf_{1(1)} & pf_{1(1)}+pf & pf_{1(1)}+pf & pf_{1(2)} \end{matrix} \right. & & & \end{matrix}$$

$$\begin{matrix} VM\_STRTD_{(2)} & VM\_UP_{(2)} & DC\_CHC_{(d)} & CHC\_RES_{(d)} & CLTS \\ \left. \begin{matrix} pf_{1(2)}+pf & pf_{1(2)}+pf & pf & pf & pf \end{matrix} \right]^T & & & & \end{matrix}$$

Since $\mathscr{I}_{(1)\sqcup(2)} > 0$ and $\mathscr{I}^T_{(1)\sqcup(2)} \times A_{(1)\sqcup(2)} = 0$, in which $A_{(1)\sqcup(2)}$ is the incidence matrix, $\mathscr{N}_{(1)\sqcup(2)}$ is structurally conservative as well as structurally bounded.

A.2 Inductive Step: Juxtaposition of P-Invariants

The following section present the resulting P-invariant related to the net union of a SPN component with a previously merged SPN. It is demonstrated that the resulting SPN is structurally conservative and bounded.

A.2.1 A VM dependability component and *m* VM dependability submodels.

In this section, a block representing *m* merged VM dependability components $(\mathscr{N}_{(m)})$ is combined with a new dependability model $(\mathscr{N}_{(n)})$. It is important to emphasize that in this example all dependability components are related to servers of the same data center. The P-invariants of the result model are presented as follows:

$$\mathscr{I}_{(m)} = \begin{matrix} & VM\_UP_{(1)} & VM\_DOWN_{(1)} & VM\_WAIT_{(1)} & VM\_STRTD_{(1)} \\ & \left[\begin{matrix} dp_{2(m)}+dp_{1(1)} & dp_{1(1)} & dp_{1(1)} & dp_{2(m)}+dp_{1(1)} \end{matrix} \right. & & & \end{matrix}$$

$$\begin{matrix} VM\_UP_{(2)} & VM\_DOWN_{(2)} & VM\_WAIT_{(2)} & VM\_STRTD_{(2)} & \cdots \\ \left. \begin{matrix} dp_{2(m)}+dp_{1(2)} & dp_{1(2)} & dp_{1(2)} & dp_{2(m)}+dp_{1(2)} & \cdots \end{matrix} \right. & & & & \end{matrix}$$

$$\begin{matrix} VM\_UP_{(m)} & VM\_DOWN_{(m)} & VM\_WAIT_{(m)} & VM\_STRTD_{(m)} & CHC\_RES \\ \left. \begin{matrix} dp_{2(m)}+dp_{1(m)} & dp_{1(m)} & dp_{1(m)} & dp_{2(m)}+dp_{1(m)} & dp_{2(m)} \end{matrix} \right] & & & & \end{matrix}$$

$$\mathscr{I}_{(n)} =$$

$$\begin{matrix} VM\_UP_{(n)} & VM\_DOWN_{(n)} & VM\_WAIT_{(n)} & VM\_STRTD_{(n)} & CHC\_RES \\ \left[\begin{matrix} dp_{2(n)}+dp_{1(n)} & dp_{1(n)} & dp_{1(n)} & dp_{2(n)}+dp_{1(n)} & dp_{2(n)} \end{matrix} \right]^T. \end{matrix}$$

$\mathscr{N}_{(m)\sqcup(n)} = \mathscr{N}_{(m)} \sqcup \mathscr{N}_{(n)}$. By juxtaposition $\mathscr{I}_{(m)\sqcup(n)} = \mathscr{J}(\mathscr{I}_{(m)}, \mathscr{I}_{(n)})$ and $dp_{2(m)} = dp_{2(n)} = dp$:

$$\mathscr{I}_{(m)} = \begin{bmatrix} \overset{VM\_UP_{(1)}}{dp + dp_{1(1)}} & \overset{VM\_DOWN_{(1)}}{dp_{1(1)}} & \overset{VM\_WAIT_{(1)}}{dp_{1(1)}} & \overset{VM\_STRTD_{(1)}}{dp + dp_{1(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{(m)}}{dp + dp_{1(m)}} & \overset{VM\_DOWN_{(m)}}{dp_{1(m)}} & \overset{VM\_WAIT_{(m)}}{dp_{1(m)}} & \overset{VM\_STRTD_{(m)}}{dp + dp_{1(m)}} & \\ \overset{VM\_UP_{(n)}}{dp + dp_{1(n)}} & \overset{VM\_DOWN_{(n)}}{dp_{1(n)}} & \overset{VM\_WAIT_{(n)}}{dp_{1(n)}} & \overset{VM\_STRTD_{(n)}}{dp + dp_{1(n)}} & \overset{CHC\_RES}{dp} \end{bmatrix}$$

Since $\mathscr{I}_{(m)\sqcup(n)} > 0$ and $\mathscr{I}^T_{(m)\sqcup(n)} \times A_{(m)\sqcup(n)} = 0$, in which $A_{(m)\sqcup(n)}$ is the result block incidence matrix, $\mathscr{N}_{(m)\sqcup(n)}$ is structurally conservative as well as structurally bounded.

A.2.2 A VM transmission component and m VM dependability submodels

In this section, a block representing $k = m + o$ merged VM dependability components $(\mathscr{N}_{(m)})$ is combined with a VM transmission component $(\mathscr{N}_{(n)})$. As previously stated, VM transmission components represent the transmission of VM data between two data center. In this example, the first data center has m dependability components and the second data center o dependability components. The P-invariants of the result model are presented as follows:

$$\mathscr{I}_{(m)} = \begin{bmatrix} \overset{VM\_UP_{d_1(1)}}{dp_{d_1} + dp_{d_1(1)}} & \overset{VM\_DOWN_{d_1(1)}}{dp_{d_1(1)}} & \overset{VM\_WAIT_{d_1(1)}}{dp_{d_1(1)}} & \overset{VM\_STRTD_{d_1(1)}}{dp_{d_1} + dp_{d_1(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_1(m)}}{dp_{d_1} + dp_{d_1(m)}} & \overset{VM\_DOWN_{d_1(m)}}{dp_{d_1(m)}} & \overset{VM\_WAIT_{d_1(m)}}{dp_{d_1(m)}} & \overset{VM\_STRTD_{d_1(m)}}{dp_{d_1} + dp_{d_1(m)}} & \overset{CHC\_RES_{d_1}}{dp_{d_1}} \\ \overset{VM\_UP_{d_2(1)}}{dp_{d_2} + dp_{d_2(1)}} & \overset{VM\_DOWN_{d_2(1)}}{dp_{d_2(1)}} & \overset{VM\_WAIT_{d_2(1)}}{dp_{d_2(1)}} & \overset{VM\_STRTD_{d_2(1)}}{dp_{d_2} + dp_{d_2(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_2(o)}}{dp_{d_2} + dp_{d_2(o)}} & \overset{VM\_DOWN_{d_2(o)}}{dp_{d_2(o)}} & \overset{VM\_WAIT_{d_2(o)}}{dp_{d_2(o)}} & \overset{VM\_STRTD_{d_2(o)}}{dp_{d_2} + dp_{d_2(o)}} & \overset{CHC\_RES_{d_2}}{dp_{d_2}} \end{bmatrix}$$

$$\mathscr{I}_{(n)} = \begin{bmatrix} \overset{CHC\_RES_{d_1}}{tr} & \overset{TRF\_P1}{tr} & \overset{TRF\_P2}{tr} & \overset{TBK\_P1}{tr} & \overset{TBK\_P2}{tr} & \overset{CHC\_RES_{d_2}}{tr} \end{bmatrix}^T.$$

$\mathscr{N}_{(m)\sqcup(n)} = \mathscr{N}_{(m)} \sqcup \mathscr{N}_{(n)}$. By juxtaposition $\mathscr{I}_{(m)\sqcup(n)} = \mathscr{J}(\mathscr{I}_{(m)}, \mathscr{I}_{(n)})$ and $dp_{d_1} = dp_{d_2} = tr = dp$:

$$\mathscr{I}_{(m)\sqcup(n)} = \begin{bmatrix} \overset{VM\_UP_{d_1(1)}}{dp+dp_{d_1(1)}} & \overset{VM\_DOWN_{d_1(1)}}{dp_{d_1(1)}} & \overset{VM\_WAIT_{d_1(1)}}{dp_{d_1(1)}} & \overset{VM\_STRTD_{d_1(1)}}{dp+dp_{d_1(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_1(m)}}{dp+dp_{d_1(m)}} & \overset{VM\_DOWN_{d_1(m)}}{dp_{d_1(m)}} & \overset{VM\_WAIT_{d_1(m)}}{dp_{d_1(m)}} & \overset{VM\_STRTD_{d_1(m)}}{dp+dp_{d_1(m)}} & \overset{CHC\_RES_{d_1}}{dp} \\ \overset{VM\_UP_{d_2(1)}}{dp+dp_{d_2(1)}} & \overset{VM\_DOWN_{d_2(1)}}{dp_{d_2(1)}} & \overset{VM\_WAIT_{d_2(1)}}{dp_{d_2(1)}} & \overset{VM\_STRTD_{d_2(1)}}{dp+dp_{d_2(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_2(o)}}{dp+dp_{d_2(o)}} & \overset{VM\_DOWN_{d_2(o)}}{dp_{d_2(o)}} & \overset{VM\_WAIT_{d_2(o)}}{dp_{d_2(o)}} & \overset{VM\_STRTD_{d_2(o)}}{dp+dp_{d_2(o)}} & \overset{CHC\_RES_{d_2}}{dp} \\ \overset{TRF\_P1}{dp} & \overset{TRF\_P2}{dp} & \overset{TBK\_P1}{dp} & \overset{TBK\_P2}{dp} & \end{bmatrix}^T$$

Since $\mathscr{I}_{(m)\sqcup(n)} > 0$ and $\mathscr{I}^T_{(m)\sqcup(n)} \times A_{(m)\sqcup(n)} = 0$, in which $A_{(m)\sqcup(n)}$ is the result block incidence matrix, $\mathscr{N}_{(m)\sqcup(n)}$ is structurally conservative as well as structurally bounded.

A.2.3 A VM performability component and m VM performability submodels.

In this section, a block representing m merged VM performability components $(\mathscr{N}_{(m)})$ is combined with a new performability component $(\mathscr{N}_{(n)})$. In this example, all performability components are related to servers of the same data center. The P-invariants of the result model are presented as follows:

$$\mathscr{I}_{(m)} = \begin{bmatrix} \overset{VM\_DOWN_{(1)}}{pb_{1(1)}} & \overset{WM\_WAIT_{(1)}}{pb_{1(1)}} & \overset{VM\_STRTD_{(1)}}{pb_{1(1)}+pb_{2(m)}} & \overset{VM\_UP_{(1)}}{pb_{1(1)}+pb_{2(m)}} & \\ \overset{VM\_DOWN_{(2)}}{pb_{1(2)}} & \overset{WM\_WAIT_{(2)}}{pb_{1(2)}} & \overset{VM\_STRTD_{(2)}}{pb_{1(2)}+pb_{2(m)}} & \overset{VM\_UP_{(2)}}{pb_{1(2)}+pb_{2(m)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_DOWN_{(m)}}{pb_{1(m)}} & \overset{WM\_WAIT_{(m)}}{pb_{1(m)}} & \overset{VM\_STRTD_{(m)}}{pb_{1(m)}+pb_{2(m)}} & \overset{VM\_UP_{(m)}}{pb_{1(m)}+pb_{2(m)}} & \\ \overset{DC\_CHC_{(d)}}{pb_{2(m)}} & \overset{CHC\_RES_{(d)}}{pb_{2(m)}} & \overset{CLTS}{pb_{2(m)}} & & \end{bmatrix}$$

$$\mathscr{I}_{(n)} = \begin{bmatrix} \overset{VM\_DOWN}{pb_{1(n)}} & \overset{WM\_WAIT}{pb_{1(n)}} & \overset{VM\_STRTD}{pb_{1(n)}+pb_{2(n)}} & \overset{VM\_UP}{pb_{1(n)}+pb_{2(n)}} & \overset{CLTS}{pb_{2(n)}} & \overset{DC\_CHC}{pb_{2(n)}} & \overset{CHC\_RES}{pb_{2(n)}} \end{bmatrix}^T.$$

$\mathscr{N}_{(m)\sqcup(n)} = \mathscr{N}_{(m)} \sqcup \mathscr{N}_{(n)}$. By juxtaposition $\mathscr{I}_{(m)\sqcup(n)} = \mathscr{J}(\mathscr{I}_{(m)}, \mathscr{I}_{(n)})$ and $pb_{2(m)} =$

$pb_{2(n)} = pb$:

$$\mathscr{I}_{(m)\sqcup(n)} = \begin{bmatrix} \overset{VM\_DOWN_{(1)}}{pb_{1(1)}} & \overset{WM\_WAIT_{(1)}}{pb_{1(1)}} & \overset{VM\_STRTD_{(1)}}{pb_{1(1)}+pb} & \overset{VM\_UP_{(1)}}{pb_{1(1)}+pb} & \overset{\cdots}{\cdots} \\ \overset{VM\_DOWN_{(m)}}{pb_{1(m)}} & \overset{WM\_WAIT_{(m)}}{pb_{1(m)}} & \overset{VM\_STRTD_{(m)}}{pb_{1(m)}+pb} & \overset{VM\_UP_{(m)}}{pb_{1(m)}+pb} & \\ \overset{VM\_DOWN_{(n)}}{pb_{1(n)}} & \overset{WM\_WAIT_{(n)}}{pb_{1(n)}} & \overset{VM\_STRTD_{(n)}}{pb_{1(n)}+pb} & \overset{VM\_UP_{(n)}}{pb_{1(n)}+pb} & \\ \overset{DC\_CHC_{(d)}}{pb} & \overset{CHC\_RES_{(d)}}{pb} & \overset{CLTS}{pb} & & \end{bmatrix}$$

Since $\mathscr{I}_{(m)\sqcup(n)} > 0$ and $\mathscr{I}^T_{(m)\sqcup(n)} \times A_{(m)\sqcup(n)} = 0$, in which $A_{(m)\sqcup(n)}$ is the result block incidence matrix, $\mathscr{N}_{(m)\sqcup(n)}$ is structurally conservative as well as structurally bounded.

A.2.4 A VM transmission component and m VM performability submodels.

In this section, a block representing $k = m + o$ merged VM performability components ($\mathscr{N}_{(m)}$) is combined with a VM transmission component ($\mathscr{N}_{(n)}$). As previously stated, VM transmission components represent the transmission of VM data between two data center. In this example, the first data center has m performability components and the second data center o performability components. The P-invariants of the result model are presented as follows:

$$\mathscr{I}_{(m)} = \begin{bmatrix} \overset{VM\_UP_{d_1(1)}}{pb_{d_1}+pb_{d_1(1)}} & \overset{VM\_DOWN_{d_1(1)}}{pb_{d_1(1)}} & \overset{VM\_WAIT_{d_1(1)}}{pb_{d_1(1)}} & \overset{VM\_STRTD_{d_1(1)}}{pb_{d_1}+pb_{d_1(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_1(m)}}{pb_{d_1}+pb_{d_1(m)}} & \overset{VM\_DOWN_{d_1(m)}}{pb_{d_1(m)}} & \overset{VM\_WAIT_{d_1(m)}}{pb_{d_1(m)}} & \overset{VM\_STRTD_{d_1(m)}}{pb_{d_1}+pb_{d_1(m)}} & \overset{CHC\_RES_{d_1}}{pb_{d_1}} \\ \overset{DC\_CHC_{d_1}}{pb_{d_1}} & \overset{CHC\_RES_{d_1}}{pb_{d_1}} & \overset{CLTS_{d_1}}{pb_{d_1}} & & \\ \overset{VM\_UP_{d_2(1)}}{pb_{d_2}+pb_{d_2(1)}} & \overset{VM\_DOWN_{d_2(1)}}{pb_{d_2(1)}} & \overset{VM\_WAIT_{d_2(1)}}{pb_{d_2(1)}} & \overset{VM\_STRTD_{d_2(1)}}{pb_{d_2}+pb_{d_2(1)}} & \overset{\cdots}{\cdots} \\ \overset{VM\_UP_{d_2(o)}}{pb_{d_2}+pb_{d_2(o)}} & \overset{VM\_DOWN_{d_2(o)}}{pb_{d_2(o)}} & \overset{VM\_WAIT_{d_2(o)}}{pb_{d_2(o)}} & \overset{VM\_STRTD_{d_2(o)}}{pb_{d_2}+pb_{d_2(o)}} & \end{bmatrix}$$

$$\begin{matrix} DC\_CHC_{d_2} & CHC\_RES_{d_2} & CLTS_{d_2} \\ pb_{d_2} & pb_{d_2} & pb_{d_2} \end{matrix} \Big]$$

$$\mathscr{I}_{(n)} = \Big[\begin{matrix} CHC\_RES_{d_1} & TRF\_P1 & TRF\_P2 & TBK\_P1 & TBK\_P2 & CHC\_RES_{d_2} \\ tr & tr & tr & tr & tr & tr \end{matrix} \Big]^T.$$

$\mathscr{N}_{(m)\sqcup(n)} = \mathscr{N}_{(m)} \sqcup \mathscr{N}_{(n)}$. By juxtaposition $\mathscr{I}_{(m)\sqcup(n)} = \mathscr{J}(\mathscr{I}_{(m)}, \mathscr{I}_{(n)})$ and $pb_{d_1} = pb_{d_2} = tr = pb$:

$$\mathscr{I}_{(m)\sqcup(n)} = \Big[\begin{matrix} VM\_UP_{d_1(1)} & VM\_DOWN_{d_1(1)} & VM\_WAIT_{d_1(1)} & VM\_STRTD_{d_1(1)} & \cdots \\ pb + pb_{d_1(1)} & pb_{d_1(1)} & pb_{d_1(1)} & pb + pb_{d_1(1)} & \cdots \end{matrix}$$

$$\begin{matrix} VM\_UP_{d_1(m)} & VM\_DOWN_{d_1(m)} & VM\_WAIT_{d_1(m)} & VM\_STRTD_{d_1(m)} \\ pb + pb_{d_1(m)} & pb_{d_1(m)} & pb_{d_1(m)} & pb + pb_{d_1(m)} \end{matrix}$$

$$\begin{matrix} DC\_CHC_{d_1} & CHC\_RES_{d_1} & CLTS_{d_1} \\ pb & pb & pb \end{matrix}$$

$$\begin{matrix} VM\_UP_{d_2(1)} & VM\_DOWN_{d_2(1)} & VM\_WAIT_{d_2(1)} & VM\_STRTD_{d_2(1)} & \cdots \\ pb + pb_{d_2(1)} & pb_{d_2(1)} & pb_{d_2(1)} & pb + pb_{d_2(1)} & \cdots \end{matrix}$$

$$\begin{matrix} VM\_UP_{d_2(o)} & VM\_DOWN_{d_2(o)} & VM\_WAIT_{d_2(o)} & VM\_STRTD_{d_2(o)} \\ pb + pb_{d_2(o)} & pb_{d_2(o)} & pb_{d_2(o)} & pb + pb_{d_2(o)} \end{matrix}$$

$$\begin{matrix} DC\_CHC_{d_2} & CHC\_RES_{d_2} & CLTS_{d_2} \\ pb & pb & pb \end{matrix}$$

$$\begin{matrix} TRF\_P1 & TRF\_P2 & TBK\_P1 & TBK\_P2 \\ pb & pb & pb & pb \end{matrix} \Big]^T$$

Since $\mathscr{I}_{(m)\sqcup(n)} > 0$ and $\mathscr{I}^T_{(m)\sqcup(n)} \times A_{(m)\sqcup(n)} = 0$, in which $A_{(m)\sqcup(n)}$ is the result block incidence matrix, $\mathscr{N}_{(m)\sqcup(n)}$ is structurally conservative as well as structurally bounded.

A.2.5 A VM performance component and m VM performance submodels.

In this section, a block representing m merged VM performance components $(\mathscr{N}_{(m)})$ is combined with a new performance component $(\mathscr{N}_{(n)})$. In this example, all performance components are related to servers of the same data center. The P-invariants of the result model are presented as follows:

$$\mathscr{I}_{(m)} = \Big[\begin{matrix} WM\_WAIT_{(1)} & VM\_STRTD_{(1)} & VM\_UP_{(1)} \\ pb_{1(1)} & pb_{1(1)} + pb_{2(m)} & pb_{1(1)} + pb_{2(m)} \end{matrix}$$

$$
\begin{array}{cccc}
\mathit{WM\_WAIT}_{(2)} & \mathit{VM\_STRTD}_{(2)} & \mathit{VM\_UP}_{(2)} & \cdots \\
pb_{1(2)} & pb_{1(2)} + pb_{2(m)} & pb_{1(2)} + pb_{2(m)} & \cdots
\end{array}
$$

$$
\begin{array}{ccc}
\mathit{WM\_WAIT}_{(m)} & \mathit{VM\_STRTD}_{(m)} & \mathit{VM\_UP}_{(m)} \\
pb_{1(m)} & pb_{1(m)} + pb_{2(m)} & pb_{1(m)} + pb_{2(m)}
\end{array}
$$

$$
\begin{array}{ccc}
\mathit{DC\_CHC}_{(d)} & \mathit{CHC\_RES}_{(d)} & \mathit{CLTS} \\
pb_{2(m)} & pb_{2(m)} & pb_{2(m)}
\end{array}
\Bigg]
$$

$$
\mathscr{I}_{(n)} =
\begin{array}{cccccc}
\mathit{WM\_WAIT} & \mathit{VM\_STRTD} & \mathit{VM\_UP} & \mathit{CLTS} & \mathit{DC\_CHC} & \mathit{CHC\_RES} \\
\Big[\; pb_{1(n)} & \begin{array}{c} pb_{1(n)}+ \\ pb_{2(n)} \end{array} & \begin{array}{c} pb_{1(n)}+ \\ pb_{2(n)} \end{array} & pb_{2(n)} & pb_{2(n)} & pb_{2(n)} \;\Big]^T.
\end{array}
$$

$\mathscr{N}_{(m)\sqcup(n)} = \mathscr{N}_{(m)} \sqcup \mathscr{N}_{(n)}$. By juxtaposition $\mathscr{I}_{(m)\sqcup(n)} = \mathscr{J}\left(\mathscr{I}_{(m)}, \mathscr{I}_{(n)}\right)$ and $pb_{2(m)} = pb_{2(n)} = pb$:

$$
\mathscr{I}_{(m)\sqcup(n)} = \Bigg[
\begin{array}{cccc}
\mathit{WM\_WAIT}_{(1)} & \mathit{VM\_STRTD}_{(1)} & \mathit{VM\_UP}_{(1)} & \cdots \\
pb_{1(1)} & pb_{1(1)} + pb & pb_{1(1)} + pb & \cdots
\end{array}
$$

$$
\begin{array}{ccc}
\mathit{WM\_WAIT}_{(m)} & \mathit{VM\_STRTD}_{(m)} & \mathit{VM\_UP}_{(m)} \\
pb_{1(m)} & pb_{1(m)} + pb & pb_{1(m)} + pb
\end{array}
$$

$$
\begin{array}{ccc}
\mathit{WM\_WAIT}_{(n)} & \mathit{VM\_STRTD}_{(n)} & \mathit{VM\_UP}_{(n)} \\
pb_{1(n)} & pb_{1(n)} + pb & pb_{1(n)} + pb
\end{array}
$$

$$
\begin{array}{ccc}
\mathit{DC\_CHC}_{(d)} & \mathit{CHC\_RES}_{(d)} & \mathit{CLTS} \\
pb & pb & pb
\end{array}
\Bigg]
$$

Since $\mathscr{I}_{(m)\sqcup(n)} > 0$ and $\mathscr{I}^T_{(m)\sqcup(n)} \times A_{(m)\sqcup(n)} = 0$, in which $A_{(m)\sqcup(n)}$ is the result block incidence matrix, $\mathscr{N}_{(m)\sqcup(n)}$ is structurally conservative as well as structurally bounded.

B

Properties of Net Union Operator

The following properties are adopted to demonstrate that all generated models, using net union operator and the proposed building block models, are structurally bounded and conservative. As the net union operator is associative, commutative and contains an identity element, it represents a *commutative monoid*. The proofs are presented as follows.

Definition 9. ***Commutative Monoid*** *Let X be a binary operation defined for a domain D ($X : D \times D \to D$). $< D, X >$ is a commutative monoid if X is associative ($(aXb)Xc = aX(bXc), \forall a,b,c \in D$), commutative ($aXb = bXa, \forall a,b \in D$) and contains an identity element $\emptyset \in D$ ($\emptyset Xa = aX\emptyset = a, \forall a \in D$).*

Theorem 2. *(Net Union: Associative Property). Net union operation is associative, since its internal operations are associative.*

Proof. Let $\mathscr{N}_1 = (P_1, T_1, I_1, O_1, H_1, \Pi_1, M_{01}, Atts_1)$, $\mathscr{N}_2 = (P_2, T_2, I_2, O_2, H_2, \Pi_2, M_{02}, Atts_2)$ and $\mathscr{N}_3 = (P_3, T_3, I_3, O_3, H_3, \Pi_3, M_{03}, Atts_3) \in$ RSSPN.

If $\mathscr{N}_a = (P_a, T_a, I_a, O_a, H_a, \Pi_a, M_{0a}, Atts_a)$ is a RSSPN obtained by $\mathscr{N}_a = \mathscr{N}_1 \sqcup \mathscr{N}_2$, then:

- $P_a = P_2 \cup P_1$
- $T_a = T_2 \cup T_1$
- $\forall p_i, t_i \in P_a \cup T_a$:
- $$\mathscr{I}_{\mathscr{N}_a}(p_i,t_t) = \begin{cases} \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{I}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$$
- $$\mathscr{O}_{\mathscr{N}_a}(p_i,t_t) = \begin{cases} \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{O}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$$
- $$\mathscr{H}_{\mathscr{N}_a}(p_i,t_t) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$$

- $\Pi_{\mathcal{N}_a}(t_i) = \begin{cases} \Pi_{\mathcal{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathcal{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$

- $\mathcal{M}_{0_{\mathcal{N}_a}}(p_i) = \begin{cases} \mathcal{M}_{0_{\mathcal{N}_1}}(p_i), & if\ p_i \in P_1 \\ \mathcal{M}_{0_{\mathcal{N}_2}}(p_i), & if\ p_i \in P_2 \end{cases}$

- $Atts_{\mathcal{N}_a}(t_i) = \begin{cases} Atts_{\mathcal{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathcal{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$

Let $\mathcal{N}_b = \mathcal{N}_a \sqcup \mathcal{N}_3$ = $(\mathcal{N}_1 \sqcup \mathcal{N}_2) \sqcup \mathcal{N}_3$ is given by:

- $P_b = P_a \cup P_3 = (P_1 \cup P_2) \cup P_3$

- $T_b = T_a \cup T_3 = (T_1 \cup T_2) \cup T_3$

- $\forall p_i, t_i \in P_b \cup T_b$:

- $\mathcal{I}_{\mathcal{N}_b}(p_i,t_t) = \begin{cases} \mathcal{I}_{\mathcal{N}_a}(p_i,t_i), & if\ p_i \in P_a \wedge t_i \in T_a \\ \mathcal{I}_{\mathcal{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathcal{I}_{\mathcal{N}_b}(p_i,t_t) = \begin{cases} \mathcal{I}_{\mathcal{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathcal{I}_{\mathcal{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathcal{I}_{\mathcal{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathcal{O}_{\mathcal{N}_b}(p_i,t_t) = \begin{cases} \mathcal{O}_{\mathcal{N}_a}(p_i,t_i), & if\ p_i \in P_a \wedge t_i \in T_a \\ \mathcal{O}_{\mathcal{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathcal{O}_{\mathcal{N}_b}(p_i,t_t) = \begin{cases} \mathcal{O}_{\mathcal{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathcal{O}_{\mathcal{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathcal{O}_{\mathcal{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathcal{H}_{\mathcal{N}_b}(p_i,t_t) = \begin{cases} \mathcal{H}_{\mathcal{N}_a}(p_i,t_i), & if\ p_i \in P_a \wedge t_i \in T_a \\ \mathcal{H}_{\mathcal{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathscr{H}_{\mathscr{N}_b}(p_i,t_t) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{H}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\Pi_{\mathscr{N}_b}(t_i) = \begin{cases} \Pi_{\mathscr{N}_a}(t_i), & if\ t_i \in T_a \\ \Pi_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

- $\Pi_{\mathscr{N}_b}(t_i) = \begin{cases} \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ \Pi_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

- $\mathscr{M}_{0\mathscr{N}_b}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_a}(p_i), & if\ p_i \in P_a \\ \mathscr{M}_{0\mathscr{N}_3}(p_i), & if\ p_i \in P_3 \end{cases}$

- $\mathscr{M}_{0\mathscr{N}_b}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_1}(p_i), & if\ p_i \in P_1 \\ \mathscr{M}_{0\mathscr{N}_2}(p_i), & if\ p_i \in P_2 \\ \mathscr{M}_{0\mathscr{N}_3}(p_i), & if\ p_i \in P_3 \end{cases}$

- $Atts_{\mathscr{N}_b}(t_i) = \begin{cases} Atts_{\mathscr{N}_a}(t_i), & if\ t_i \in T_a \\ Atts_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

- $Atts_{\mathscr{N}_b}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathscr{N}_2}(t_i), & if\ t_2 \in T_2 \\ Atts_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

If $\mathscr{N}_c = (P_c, T_c, I_c, O_c, H_c, \Pi_c, M_{0c}, Atts_c)$ is a RSSPN obtained by $\mathscr{N}_c = \mathscr{N}_2 \sqcup \mathscr{N}_3$, then:

- $P_c = P_2 \cup P_3$

- $T_c = T_2 \cup T_3$

- $\forall p_i, t_i \in P_c \cup T_c$:

- $\mathscr{I}_{\mathscr{N}_c}(p_i,t_t) = \begin{cases} \mathscr{I}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{I}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathscr{O}_{\mathscr{N}_c}(p_i,t_t) = \begin{cases} \mathscr{O}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{O}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathscr{H}_{\mathscr{N}_c}(p_i,t_t) = \begin{cases} \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{H}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\Pi_{\mathscr{N}_c}(t_i) = \begin{cases} \Pi_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ \Pi_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

- $\mathscr{M}_{0_{\mathscr{N}_c}}(p_i) = \begin{cases} \mathscr{M}_{0_{\mathscr{N}_2}}(p_i), & if\ p_i \in P_2 \\ \mathscr{M}_{0_{\mathscr{N}_3}}(p_i), & if\ p_i \in P_3 \end{cases}$

- $Atts_{\mathscr{N}_c}(t_i) = \begin{cases} Atts_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ Atts_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

Let $\mathscr{N}_d = \mathscr{N}_1 \sqcup \mathscr{N}_c = \mathscr{N}_1 \sqcup (\mathscr{N}_2 \sqcup \mathscr{N}_3)$ is given by:

- $P_d = P_c \cup P_1 = P_1 \cup (P_2 \cup P_3)$

- $T_d = T_c \cup T_1 = T_1 \cup (T_2 \cup T_3)$

- $\forall p_i \in P_c \wedge t_i \in T_c$:

- $\mathscr{I}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{I}_{\mathscr{N}_c}(p_i,t_i), & if\ p_i \in P_c \wedge t_i \in T_c \\ 0, & otherwise \end{cases}$

- $\mathscr{I}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{I}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{I}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathscr{O}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{O}_{\mathscr{N}_c}(p_i,t_i), & if\ p_i \in P_c \wedge t_i \in T_c \\ 0, & otherwise \end{cases}$

- $\mathscr{O}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{O}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{O}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\mathscr{H}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathscr{N}_c}(p_i,t_i), & if\ p_i \in P_c \wedge t_i \in T_c \\ 0, & otherwise \end{cases}$

- $\mathscr{H}_{\mathscr{N}_d}(p_i,t_t) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{H}_{\mathscr{N}_3}(p_i,t_i), & if\ p_i \in P_3 \wedge t_i \in T_3 \\ 0, & otherwise \end{cases}$

- $\Pi_{\mathscr{N}_d}(t_i) = \begin{cases} \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathscr{N}_c}(t_i), & if\ t_i \in T_c \end{cases}$

- $\Pi_{\mathscr{N}_d}(t_i) = \begin{cases} \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ \Pi_{\mathscr{N}_3}(t_i), & if\ t_i \in T_3 \end{cases}$

- $\mathscr{M}_{0\mathscr{N}_d}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_1}(p_i), & if\ p_i \in P_1 \\ \mathscr{M}_{0\mathscr{N}_c}(p_i), & if\ p_i \in P_c \end{cases}$

- $\mathscr{M}_{0\mathscr{N}_d}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_1}(p_i), & if\ p_i \in P_1 \\ \mathscr{M}_{0\mathscr{N}_2}(p_i), & if\ p_i \in P_2 \\ \mathscr{M}_{0\mathscr{N}_3}(p_i), & if\ p_i \in P_3 \end{cases}$

- $Atts_{\mathscr{N}_d}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathscr{N}_c}(t_i), & if\ t_i \in T_c \end{cases}$

- $Atts_{\mathscr{N}_d}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathscr{N}_2}(t_i), & if\ t_2 \in T_2 \\ Atts_{\mathscr{N}_3}(t_i), & if\ t_3 \in T_3 \end{cases}$

Thus, net union operation is associative, since $\mathscr{N}_b = (\mathscr{N}_1 \sqcup \mathscr{N}_2) \sqcup \mathscr{N}_3$ and $\mathscr{N}_d = \mathscr{N}_1 \sqcup (\mathscr{N}_2 \sqcup \mathscr{N}_3)$, and $\mathscr{N}_b = \mathscr{N}_d$.

Theorem 3. ***Net Union: Commutative Property***. *Net union operation is commutative, since its internal operations are commutative.*

Proof. Let $\mathscr{N}_1 = (P_1, T_1, I_1, O_1, H_1, \Pi_1, M_{01}, Atts_1)$ and $\mathscr{N}_2 = (P_2, T_2, I_2, O_2, H_2, \Pi_2, M_{02}, Atts_2) \in$ RSSPN.

If $\mathscr{N}_a = (P_a, T_a, I_a, O_a, H_a, \Pi_a, M_{0a}, Atts_a)$ is a RSSPN obtained by $\mathscr{N}_a = \mathscr{N}_1 \sqcup \mathscr{N}_2$, then:

- $P_a = P_1 \cup P_2$
- $T_a = T_1 \cup T_2$
- $\forall p_i, t_i \in P_a \cup T_a$:
- $\mathscr{I}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{I}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\mathscr{O}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{O}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\mathscr{H}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ 0, & otherwise \end{cases}$
- $\Pi_{\mathscr{N}_a}(t_i) = \begin{cases} \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ \Pi_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$
- $\mathscr{M}_{0\mathscr{N}_a}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_1}(p_i), & if\ p_i \in P_1 \\ \mathscr{M}_{0\mathscr{N}_2}(p_i), & if\ p_i \in P_2 \end{cases}$
- $Atts_{\mathscr{N}_a}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \\ Atts_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \end{cases}$

If $\mathscr{N}_b = (P_b, T_b, I_b, O_b, H_b, \Pi_b, M_{0b}, Atts_b)$ is a RSSPN obtained by $\mathscr{N}_b = \mathscr{N}_2 \sqcup \mathscr{N}_1$, then:

- $P_b = P_2 \cup P_1$
- $T_b = T_2 \cup T_1$
- $\forall p_i, t_i \in P_b \cup T_b$:

- $\mathscr{I}_{\mathscr{N}_b}(p_i,t_i) = \begin{cases} \mathscr{I}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$
- $\mathscr{O}_{\mathscr{N}_b}(p_i,t_i) = \begin{cases} \mathscr{O}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$
- $\mathscr{H}_{\mathscr{N}_b}(p_i,t_i) = \begin{cases} \mathscr{H}_{\mathscr{N}_2}(p_i,t_i), & if\ p_i \in P_2 \wedge t_i \in T_2 \\ \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$
- $\Pi_{\mathscr{N}_b}(t_i) = \begin{cases} \Pi_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \end{cases}$
- $\mathscr{M}_{0_{\mathscr{N}_b}}(p_i) = \begin{cases} \mathscr{M}_{0_{\mathscr{N}_2}}(p_i), & if\ p_i \in P_2 \\ \mathscr{M}_{0_{\mathscr{N}_1}}(p_i), & if\ p_i \in P_1 \end{cases}$
- $Atts_{\mathscr{N}_b}(t_i) = \begin{cases} Atts_{\mathscr{N}_2}(t_i), & if\ t_i \in T_2 \\ Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \end{cases}$

  Thus, net union operation is commutative, since $\mathscr{N}_a = \mathscr{N}_1 \sqcup \mathscr{N}_2 = \mathscr{N}_b = \mathscr{N}_2 \sqcup \mathscr{N}_1$.

Let $\mathscr{N}_\emptyset = (P_\emptyset, T_\emptyset, I_\emptyset, O_\emptyset, H_\emptyset, \Pi_\emptyset, M_{0\emptyset}, Atts_\emptyset) \in$ RSSPN, where $P_\emptyset = T_\emptyset = \emptyset$ and the other components are empty matrices. An empty matrix corresponds to a matrix whose number of rows or columns (or both) is equal to zero (92).

Theorem 4. *$\mathscr{N}_\emptyset$ is the identity element of net union operation, since $\forall \mathscr{N} \in RSSPN, \mathscr{N} = \mathscr{N} \sqcup \mathscr{N}_\emptyset$*

Proof. If $\mathscr{N}_a = (P_a, T_a, I_a, O_a, H_a, \Pi_a, M_{0a}, Atts_a)$ and $\mathscr{N}_1 = (P_1, T_1, I_1, O_1, H_1, \Pi_1, M_{01}, Atts_1)$ are RSSPNs, and obtained by $\mathscr{N}_a = \mathscr{N}_1 \sqcup \mathscr{N}_\emptyset$, then:

- $P_a = P_1 \cup \emptyset$
- $T_a = T_1 \cup \emptyset$
- $\forall p_i, t_i \in P_1 \cup T_1$:
- $\mathscr{I}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{I}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$

  as $P_1 = P_a$ and $T_1 = T_a$, $\mathscr{I}_{\mathscr{N}_a}(p_i,t_i) = \mathscr{O}_{\mathscr{N}_1}(p_i,t_i)$

- $\mathscr{O}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{O}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$

  as $P_1 = P_a$ and $T_1 = T_a$, $\mathscr{O}_{\mathscr{N}_a}(p_i,t_i) = \mathscr{O}_{\mathscr{N}_1}(p_i,t_i)$

- $\mathscr{H}_{\mathscr{N}_a}(p_i,t_i) = \begin{cases} \mathscr{H}_{\mathscr{N}_1}(p_i,t_i), & if\ p_i \in P_1 \wedge t_i \in T_1 \\ 0, & otherwise \end{cases}$

  as $P_1 = P_a$ and $T_1 = T_a$, $\mathscr{H}_{\mathscr{N}_a}(p_i,t_i) = \mathscr{H}_{\mathscr{N}_1}(p_i,t_i)$

- $\Pi_{\mathscr{N}_a}(t_i) = \begin{cases} \Pi_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \end{cases}$

  as $T_1 = T_a$, $\Pi_{\mathscr{N}_a}(t_i) = \Pi_{\mathscr{N}_1}(t_i)$

- $\mathscr{M}_{0\mathscr{N}_a}(p_i) = \begin{cases} \mathscr{M}_{0\mathscr{N}_1}(p_i), & if\ p_i \in P_1 \end{cases}$

  as $P_1 = P_a$, $\mathscr{M}_{0\mathscr{N}_a}(p_i) = \mathscr{M}_{0\mathscr{N}_1}(p_i)$

- $Atts_{\mathscr{N}_a}(t_i) = \begin{cases} Atts_{\mathscr{N}_1}(t_i), & if\ t_i \in T_1 \end{cases}$

  as $T_1 = T_a$, $Atts_{\mathscr{N}_a}(t_i) = Atts_{\mathscr{N}_1}(t_i)$

  Therefore, net union operation has an identity element, since $\mathscr{N}_a = \mathscr{N}_1$.