Pós-Graduação em Ciência da Computação

Rubens de Souza Matos Júnior

IDENTIFICATION OF AVAILABILITY AND PERFORMANCE BOTTLENECKS IN CLOUD COMPUTING SYSTEMS: AN APPROACH BASED ON HIERARCHICAL MODELS AND SENSITIVITY ANALYSIS

Ph.D. Thesis

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

RECIFE
March 2016

Federal University of Pernambuco
Center for Informatics
Graduate in Computer Science

Rubens de Souza Matos Júnior

IDENTIFICATION OF AVAILABILITY AND PERFORMANCE BOTTLENECKS IN CLOUD COMPUTING SYSTEMS: AN APPROACH BASED ON HIERARCHICAL MODELS AND SENSITIVITY ANALYSIS

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*
Co-Advisor: *Kishor S. Trivedi*

RECIFE
March 2016

Tese de doutorado apresentada por **Rubens de Souza Matos Júnior** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Identification of availability and performance bottlenecks in cloud computing systems: An approach based on hierarchical models and sensitivity analysis**, orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela banca examinadora formada pelos professores:

_____

Prof. Paulo Roberto Freire Cunha
Centro de Informática/UFPE

_____

Prof. Nelson Souto Rosa
Centro de Informática/UFPE

_____

Prof. Eduardo Antônio Guimarães Tavares
Centro de Informática/UFPE

_____

Prof. Edmundo Roberto Mauro Madeira
Instituto de Computação/UNICAMP

_____

Prof. André Brinkmann
Computer Science Department/Johannes Gutenberg - Universität Mainz

RECIFE
March 2016

I dedicate this thesis to my beloved wife, for her love, friendship, patience, and dedication through these long years. It is also dedicated to my mother and sisters, that gave me comfort and full support during this journey. And in memoriam to my father, who could not live up to this moment, but I must remark how much I owe him for reaching this stage.

Acknowledgements

Thanks God for allowing me to reach this moment, giving me health and strength to overcome adversities.

Thanks to all colleagues and friends that I met in UFPE, especially those from MoDCS group: Jean, Jamilson, Danilo, Vandi, Artur, Ermeson, Debora, Julian, Gustavo, Rafael, Joao, Jair, Bruno Silva, Bruno Nogueira, Airton, Charles, Carlos, Almir, Erica, Gracieth, and Tiago. They made this undertaking lighter but also more productive than I ever imagined. The collaboration in such a prolific research group helped for the success in my activities.

The support from the funding agencies, CAPES (grant PDSE 17988/12-8) and FACEPE (grant IBPG-0430-1.03/10) was essential for proper development of the activities reported here.

My gratitude also goes to the people from DHAAL group, and professor Kishor Trivedi, for accepting me for a short but valuable experience at Duke University and for sharing his knowledge.

At last, but not least, I am so grateful for the opportunities, confidence, and support that my advisor, professor Paulo Maciel, has always given me from the very beginning. The technical knowledge, perseverance, and hard work are examples that inspired through the last years.

Obstacles are those frightful things you see when you take your eyes off your goal.

—HENRY FORD

Resumo

O paradigma de computação em nuvem é capaz de reduzir os custos de aquisição e manutenção de sistemas computacionais e permitir uma gestão equilibrada dos recursos de acordo com a demanda. Modelos analíticos hierárquicos e compostos são adequados para descrever de forma concisa o desempenho e a confiabilidade de sistemas de computação em nuvem, lidando com o grande número de componentes que constituem esse tipo de sistema. Esta abordagem usa sub-modelos distintos para cada nível do sistema e as medidas obtidas em cada sub-modelo são usadas para calcular as métricas desejadas para o sistema como um todo. A identificação de gargalos em modelos hierárquicos pode ser difícil, no entanto, devido ao grande número de parâmetros e sua distribuição entre os distintos formalismos e níveis de modelagem. Esta tese propõe métodos para a avaliação e detecção de gargalos de sistemas de computação em nuvem. A abordagem baseia-se na modelagem hierárquica e técnicas de análise de sensibilidade paramétrica adaptadas para tal cenário. Esta pesquisa apresenta métodos para construir rankings unificados de sensibilidade quando formalismos de modelagem distintos são combinados. Estes métodos são incorporados no software Mercury, fornecendo uma estrutura automatizada de apoio ao processo. Uma metodologia de suporte a essa abordagem foi proposta e testada ao longo de estudos de casos distintos, abrangendo aspectos de hardware e software de sistemas IaaS (Infraestrutura como um serviço), desde o nível de infraestrutura básica até os aplicativos hospedados em nuvens privadas. Os estudos de caso mostraram que a abordagem proposta é útil para orientar os projetistas e administradores de infraestruturas de nuvem no processo de tomada de decisões, especialmente para ajustes eventuais e melhorias arquiteturais. A metodologia também pode ser aplicada por meio de um algoritmo de otimização proposto aqui, chamado Sensitive GRASP. Este algoritmo tem o objetivo de otimizar o desempenho e a confiabilidade de sistemas em cenários onde não é possível explorar todas as possibilidades arquiteturais e de configuração para encontrar a melhor qualidade de serviço. Isto é especialmente útil para os serviços hospedados na nuvem e suas complexas infraestruturas subjacentes.

Palavras-chave: Computação em nuvem; avaliação de desempenho; dependabilidade; modelos analíticos; análise de sensibilidade; cadeias de Markov; otimização

Abstract

Cloud computing paradigm is able to reduce costs of acquisition and maintenance of computer systems, and enables the balanced management of resources according to the demand. Hierarchical and composite analytical models are suitable for describing performance and dependability of cloud computing systems in a concise manner, dealing with the huge number of components which constitute such kind of system. That approach uses distinct sub-models for each system level and the measures obtained in each sub-model are integrated to compute the measures for the whole system. Identification of bottlenecks in hierarchical models might be difficult yet, due to the large number of parameters and their distribution among distinct modeling levels and formalisms. This thesis proposes methods for evaluation and detection of bottlenecks of cloud computing systems. The methodology is based on hierarchical modeling and parametric sensitivity analysis techniques tailored for such a scenario. This research introduces methods to build unified sensitivity rankings when distinct modeling formalisms are combined. These methods are embedded in the Mercury software tool, providing an automated sensitivity analysis framework for supporting the process. Distinct case studies helped in testing the methodology, encompassing hardware and software aspects of cloud systems, from basic infrastructure level to applications that are hosted in private clouds. The case studies showed that the proposed approach is helpful for guiding cloud systems designers and administrators in the decision-making process, especially for tune-up and architectural improvements. It is possible to employ the methodology through an optimization algorithm proposed here, called Sensitive GRASP. This algorithm aims at optimizing performance and dependability of computing systems that cannot stand the exploration of all architectural and configuration possibilities to find the best quality of service. This is especially useful for cloud-hosted services and their complex underlying infrastructures.

Keywords: Cloud computing; performance evaluation; dependability; analytical modeling; sensitivity analysis; Markov chains; optimization

List of Figures

List of Tables

List of Algorithms

List of Acronyms

API Application Programming Interface 25

CC Cluster Controller 27

CLC Cloud Controller 27

CRM Customer Relationship Management 21

CTMC Continuous Time Markov Chain 31

DBaaS Database as a Service 22

DHCP Dynamic Host Configuration Protocol 25

DNS Domain Name System 25

DOE Design of experiments 39

DRaaS Disaster Recovery as a Service 22

DSaaS Data Storage as a Service 22

DSPN Deterministic and Stochastic Petri Net 36

DTMC Discrete Time Markov Chain 33

EKI Eucalyptus Kernel Image 97

ELB Elastic Load Balancing 30

EMI Eucalyptus Machine Image 97

ERI Eucalyptus Ramdisk Image 97

ERP Enterprise Resource Planning 21

GPS Global Positioning System 95

GRASP Greedy Randomized Adaptive Search Procedure

GSPN Generalized Stochastic Petri Net 35

IaaS Infrastructure as a Service 17

KVM Kernel-based Virtual Machine

MBaaS Mobile Backend as a Service 22

MRM Markov Reward Model 31

NAT Network Address Translation 25

NC Node Controller 27

NIST National Institute of Standards and Technology 20

PaaS Platform as a Service 20

RBD Reliability Block Diagram 16

S3 Simple Storage Service 28

SaaS Software as a Service 20

SC Storage Controller 27

SLA Service Level Agreement 16

SPN Stochastic Petri Net 31

SRN Stochastic Reward Net 43

UML Unified Modeling Language 95

VLAN Virtual Local Area Network 25

VM virtual machine 22

VMI Virtual Machine Image

VMM Virtual Machine Monitor 24

XaaS Everything as a Service 22

Summary

1

Introduction

Cloud computing makes computer resources (processing power, storage, software) available through the Internet, with service providers potentially located anywhere around the world. These service providers have been using cloud computing paradigm to reduce acquisition costs and manage the highly variable demands requested by their customers. Cloud computing provided high flexibility due to the integration of virtualization technologies and mechanisms for automated hardware and network management (VOORSLUYS et al., 2009; BARHAM et al., 2003). The variety of components in such systems and the interaction among them bring new challenges to assure the desired or contracted levels of performance and dependability (reliability, availability, and security).

Analytical modeling helps to plan and manage hardware, network and software infrastructures (CALLOU et al., 2011; MATOS JUNIOR et al., 2011; SOUSA; MACIEL; ARAUJO, 2009), either by comparing alternative configurations before implementing a system, or by allowing the prediction of effects in system availability and performance after changes in its components. Such planning is essential for mission critical systems, and service providers, who need to meet strict Service Level Agreements (SLAs), whose violation may result in fines, cancellation of contracts and other financial losses (SATO; TRIVEDI, 2007; ARAUJO et al., 2011). Reliability Block Diagrams (RBDs) (MACIEL et al., 2011), Fault Trees, queuing networks, Markov chains and Petri nets (MALHOTRA; TRIVEDI, 1994) are among the formal models commonly found in this context. Cloud computing systems, due to their characteristics (e.g., virtualization, and layered modular architecture), can be described in a more concise manner through hierarchical models. In this approach, different levels of the system are represented separately in sub-models, and the measures of each sub-model are integrated to obtain the measures for the system as a whole.

Systems modeling usually requires to handle many different parameters, for both performance and dependability studies. Each parameter can have a distinct impact on availability, reliability and performance measures, therefore is crucial to know the “order of importance” of the model parameters, so you can decide the appropriate level of attention given to each one (BONDAVALLI; MURA; TRIVEDI, 1999). Parametric sensitivity analysis (FRANK, 1978;

HAMBY, 1994) is a method to determine the order of influence of the parameters on the results of a model. This method has been applied in some types of analytical models, such as Markov chains, Petri nets and queuing networks, but in an isolated way, without considering the integration of different models in a hierarchical approach.

The computation of the order of importance for parameters in a hierarchical model is not trivial, because there are multiple components in models of different levels. An example is found in studies involving availability and reliability because the failure of a component in a given model may impact the measures of a subsystem represented by another model, incurring in relations difficult to be measured. Therefore, the creation of a sensitivity analysis methodology tailored to the characteristics of hierarchical modeling would benefit the evaluation of dependability and performance of cloud systems, as well as other areas that use this kind of approach.

Considering the challenges just presented, this study aims at developing methods for accurate identification of performance, reliability and availability bottlenecks, especially for designers and administrators of complex systems, such as virtualized data centers and Infrastructure as a Service (IaaS) cloud computing environments. a methodology that leverages the evaluation of cloud computing systems. Such an approach should employ hierarchical modeling techniques and combine specific sensitivity analysis indices for distinct models. Decision-making processes may be guided by the hierarchical analytical models and their sensitivity analysis, providing robust and clear information about the points of optimization in the analyzed system.

1.1 Objectives

The main objective of this research is to propose methods for the detection of performance and dependability bottlenecks in cloud computing systems. This approach should enable the identification of points for improvement at different levels of the systems under study, through hardware and software. The proposed methods will ease compliance with some non-functional requirements expected by users of cloud systems, such as availability and response time. Among the specific goals of the research, we can list:

- Build and validate performance and availability models for cloud computing architectures found in corporate environments and described in the literature.
- Develop composition methods for sensitivity metrics of analytical models commonly used in the areas of performance and dependability evaluation.
- Automate the computation of sensitivity metrics for hierarchical models in the Mercury tool (MERCURY, 2016).

- Recommend improvements to some existing cloud computing architectures through the proposed methodology and its sensitivity analysis results.

1.2 Aimed contributions

The main contributions that we plan to provide through this thesis are the following:

- Models for dependability evaluation of cloud computing systems, addressing application and infrastructure issues.
- Models for performance evaluation of applications that employ specific features of cloud computing systems, such as elasticity mechanisms (e.g., autoscaling).
- Methods and tools for automated sensitivity analysis of hierarchical models.
- A supporting methodology that describes the activities required for proper definition and analysis of models.
- Algorithms for optimization of cloud computing infrastructures and services, integrating sensitivity analysis techniques and well-established optimization methods.

Such results shall empower the proper planning of cloud computing infrastructures, especially the modifications made for systems already in production. The products of this research are supposed to aid the fulfillment of users and administrators expectations that are usually defined by means of SLAs.

1.3 Organization of the document

This thesis is structured as follows. Chapter 2 clarifies some relevant background themes the reader should know for properly understanding this document. Chapter 3 discusses noteworthy works found in literature that have some topics in common to those addressed in this thesis. The proposed sensitivity analysis methods for identification of bottlenecks are described in Chapter 4. It also explains the methodology that supports the sensitivity analysis for hierarchical models.

Chapter 5 presents case studies which were used to verify the applicability of the proposed approach as well as to demonstrate its benefits and limitations. Final remarks are discussed in Chapter 6. The Appendix A presents details on the implementation of sensitivity analysis features for the Mercury tool (MERCURY, 2016)(SILVA et al., 2015).

2

Background

This chapter discusses the basic concepts of three main areas that set up the focus for this thesis: cloud computing, dependability and performance modeling, and sensitivity analysis. The background presented here shall provide the necessary knowledge for a clear comprehension of the chapters ahead, including the aspects surrounding the proposed techniques and subsequent case studies.

2.1 Cloud Computing

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction (MELL; GRANCE, 2011). ARMBRUST et al. (2010) stress that cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. In such a model, users access services based on their requirements without regard to where the services are hosted or how they are delivered (BUYYA et al., 2009).

Numerous advances in software architecture and hardware virtualization have leveraged the adoption of cloud computing, supporting the development of applications which scale gracefully and automatically (SUN, 2009), also known as elastic computing.

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

Different cloud computing offerings will be distinguished based on the level of abstraction presented to the programmer and the level of management of the resources. Therefore, it is possible to identify distinct service models and deployment models for cloud systems, that are presented as follows.

2.1.1 Service Models

The National Institute of Standards and Technology (NIST) of USA, has defined three service models for cloud computing: Software as a Service (SaaS), Platform as a Service (PaaS), and IaaS (MELL; GRANCE, 2011). Figure 2.1 shows the services models arranged as layers of a complete cloud service, and shows examples of services usually provided in each layer.

With IaaS, the consumer can allocate processing, storage, networks, and other fundamental computing resources. The consumer is able to deploy and run arbitrary software on top of such resources, which may include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, data, and deployed applications; and usually limited regulation of some networking components (e.g., host firewalls).

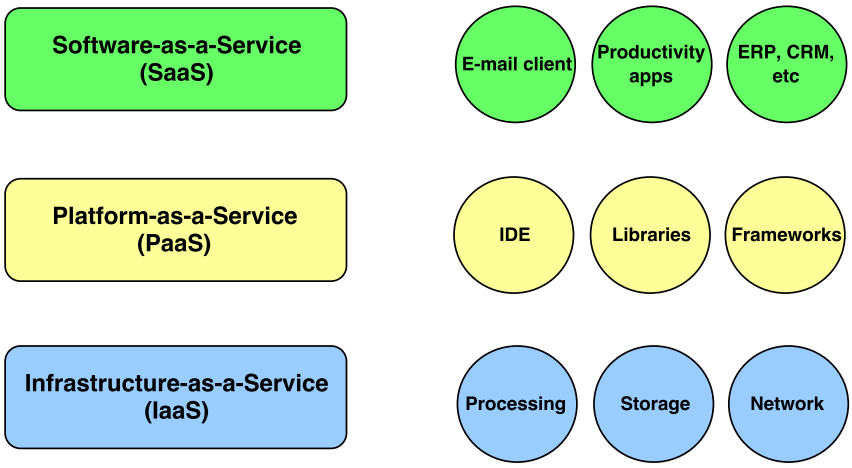


Figure 2.1: Cloud service models

Amazon, Google, Microsoft, Rackspace, and Salesforce are among the major cloud computing providers. Those companies enable fast deployment of various computational resources to their customers, that pay only for what they effectively used. Amazon Elastic Compute Cloud (EC2) is one of the pioneer and most successful IaaS products, which was followed by competitors such as Google Compute Cloud, and Microsoft Azure. There is a variety of options in terms of processing power (e.g, number of CPU cores), main memory capacity, and storage space for the virtual machines that a user can deploy in those services.

PaaS provides a framework that developers can build upon to develop or customize cloud-based applications. It includes middleware, programming libraries, development tools, database management systems, and other services to support the web application lifecycle, i.e.,

building, testing, deploying, managing, and updating (MICROSOFT, 2016a).

Google App Engine is an example of PaaS, that allows developers creating and deploying applications with Python, Java, PHP, and Go programming languages directly on the Google's infrastructure, benefiting from ready-to-use libraries and frameworks, as well as the ability to scale as traffic and data storage needs change (GOOGLE, 2016). Microsoft Azure also has PaaS capabilities, enabling software development with JavaScript, Python, .NET, PHP, Java, and Node.js (MICROSOFT, 2016b).

With SaaS, the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface (MELL; GRANCE, 2011). The consumer only controls limited user-specific application configuration settings, and does not know nor has access to the underlying hardware, operating system, and any other software infrastructure that supports that application.

SaaS has become a major trend in the development of applications. There are important examples of cloud-based software in almost any category, including office suites, Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), photo edition tools, and media players. Google Docs, Microsoft Office 360, Adobe Creative Cloud, Salesforce CRM, GMail for Work, and Youtube are services that replace traditional client-server or stand-alone desktop applications. Such services are usually built on top of PaaS, or directly on IaaS environments without the aid of PaaS features. The customer of SaaS has little or none installation requirements besides a web browser, but fast and reliable network connectivity becomes an important need.

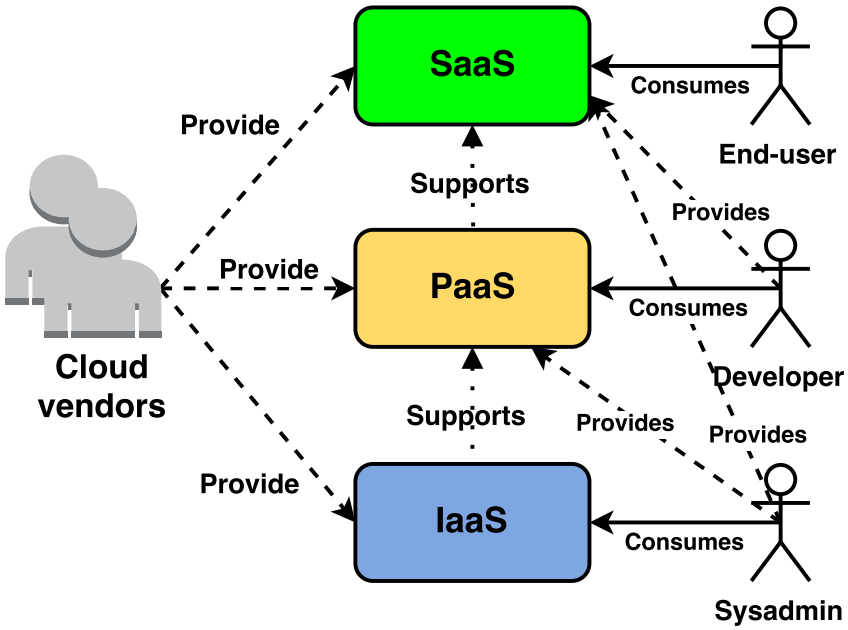


Figure 2.2: Cloud computing actors by service model

Figure 2.2 depicts the target audience of each service model. End-users consume ap-

plications provided by SaaS vendors; software developers consume programming resources at PaaS environments; and systems administrators (sysadmins) consume processing, networking, and storage capacity from IaaS providers. It is worth noticing that cloud computing users at IaaS level might use the resources to become a PaaS or SaaS provider. A PaaS user is also expected to create applications to be accessed in the SaaS model.

The three service models defined by NIST were expanded in the literature and industry terminology by the inclusion of Data Storage as a Service (DSaaS), Database as a Service (DBaaS), Mobile Backend as a Service (MBaaS), Disaster Recovery as a Service (DRaaS), and many similar acronyms. This induced the creation of the term Everything as a Service (XaaS) to emphasize that potentially any computational resource or activity can be offered as a service in the cloud.

This Ph.D. thesis deals mainly with IaaS, and the concept of cloud most used here is: "a group of machines configured in such a way that an end-user can request any number of virtual machines (VMs) of a desired configuration" (SEMPOLINSKI; THAIN, 2010). The cloud will run these VMs somewhere on the pool of physical machines that it owns. The word "cloud" in this context denotes the tenuous or almost intangible nature of these VMs. As mentioned by SEMPOLINSKI; THAIN (2010), the end-user neither knows nor cares where exactly these VMs are physically located or the configuration of the underlying hardware, so long as they can access their bank of properly configured VMs. On the other hand, a cloud system administrator must have deeper knowledge on the infrastructure that is being offered and how those components can be tuned to assure acceptable quality of service. Therefore, the methods proposed in this thesis should be most useful for the system administrators of IaaS platforms. To a lesser extent, this work also takes into account details of software applications running on top of IaaS clouds. The distinction between IaaS components and end-user application components will be clearly specified in the case studies when this becomes necessary.

2.1.2 Deployment Models

Clouds can also be classified in terms of deployment models that define who owns and manages the cloud (FURHT; ESCALANTE, 2010). Figure 2.3 depicts the three main deployment models: public, private, and hybrid clouds.

According to NIST (MELL; GRANCE, 2011), in a public cloud the infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider, and the access to resources is usually charged in a pay-as-you-go manner. ZHANG; CHENG; BOUTABA (2010) emphasize that public clouds offer several key benefits to service providers, including no initial capital investment on infrastructure and shifting of risks to infrastructure providers.

A private cloud, or internal cloud, is provisioned for exclusive use by a single organi-

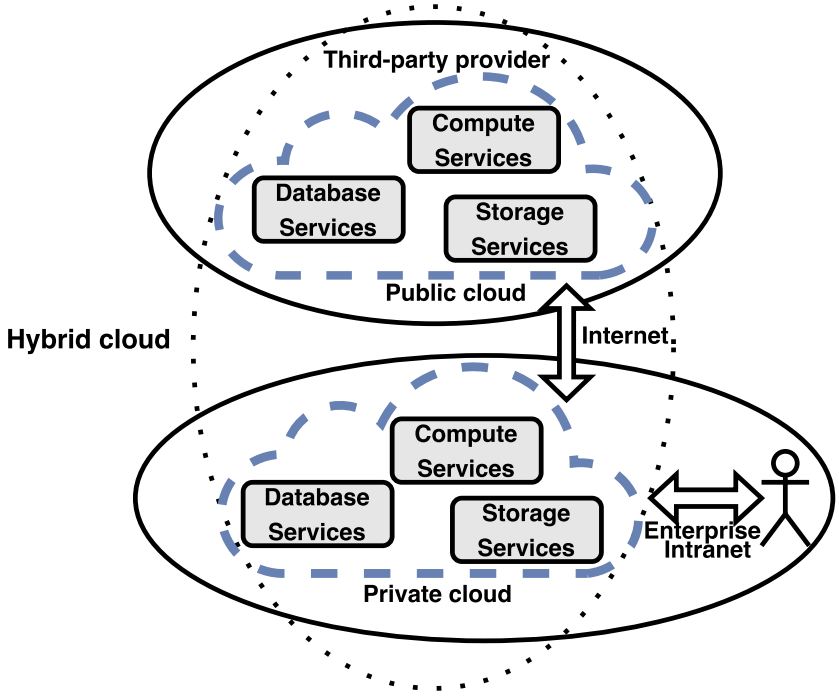


Figure 2.3: Cloud deployment models

zation comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises (MELL; GRANCE, 2011) (FURHT; ESCALANTE, 2010). It is usually adopted when the organization has concerns on storing or processing data outside their own facilities. Beyond privacy and security concerns, a need for fine-grained level of control on software, network, and hardware components might lead to the choice of private clouds instead of public clouds.

A hybrid cloud is a composition of two or more distinct cloud infrastructures that remain unique entities, but are accessed jointly by means of standardized or proprietary technology that enables data and application portability (MELL; GRANCE, 2011). Hybrid clouds might be arranged to keep confidential or other sensible data exclusively on local premises whereas common information is stored and processed on public infrastructure. Load balancing between clouds is also an application for hybrid clouds.

Some works also consider a fourth deployment model, called as community clouds (MELL; GRANCE, 2011). In such a model, infrastructure resources and costs are shared by distinct organizations with common purposes or concerns. The resources are integrated in a manner that every partner has access to a virtually larger computational capacity with reduced costs, but without demanding public cloud services.

2.1.3 Open-source Cloud Computing Platforms

There are some software frameworks for building private clouds. Eucalyptus (EUCALYPTUS, 2016), OpenNebula (OPENNEBULA, 2016), Openstack (OPENSTACK, 2016), and

Cloudstack (CLOUDSTACK, 2016) are well-known open-source cloud platforms and the documentation available about their internal functioning makes them interesting for a deeper analysis of the components in an IaaS cloud. Those platforms share some characteristics that allow us to describe a generic infrastructure needed to make a cloud computing system work (SEMPOLINSKI; THAIN, 2010)(VON LASZEWSKI et al., 2012). Figure 2.4 depicts such parts of an abstract cloud software architecture.

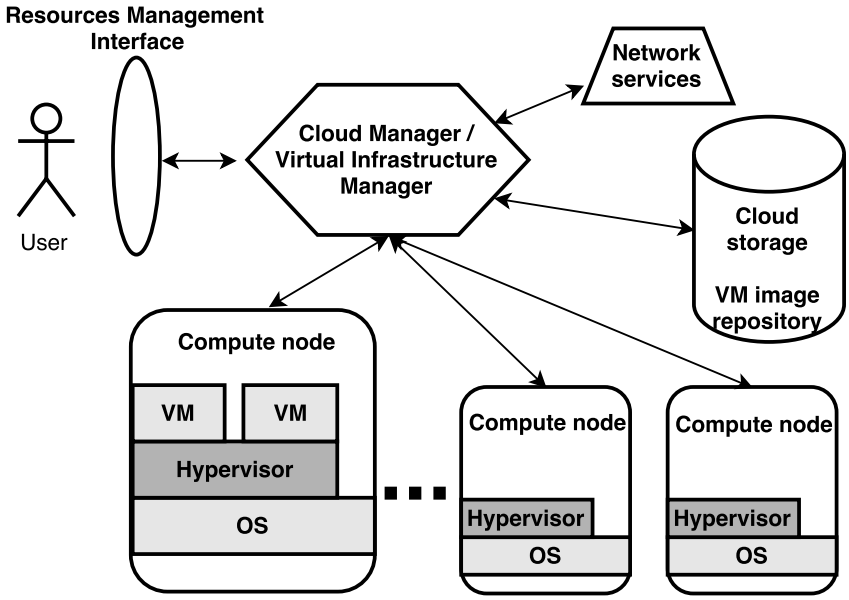


Figure 2.4: Generic cloud software architecture

A cloud must have a pool of physical machines (compute nodes), where VMs will be instantiated. The hardware and the base operating system on the compute nodes have important requirements for proper usage on a cloud environment. First, the processors of the physical nodes should have extensions for virtualization, and these extensions must be enabled on the BIOS setup. According to (VON LASZEWSKI et al., 2012), the absence of such virtualization extensions greatly limit both the speed of the VMs and the choice of software components. Kernel modules for virtualization might also be required on the operating system.

The virtual machine hypervisor, also known as Virtual Machine Monitor (VMM), provides a framework which allows VMs to run. Popular VMMs include Xen, KVM, and VirtualBox, that are open-source, and VMware, which is proprietary. Distinct cloud frameworks support distinct subsets of hypervisors.

Another common component among cloud frameworks is a repository of VM disk images. In order to usefully run VMs, a virtual hard drive must be available. In cases where one is simply creating a single VM on a single physical machine, a blank disk image is created and the VM installs an operating system and other software. However, in a cloud environment, dozens or even thousands of VMs might be created and terminated in a short timespan, so the installation of a full operating system on each VM would take too much time. For this reason, each cloud system has a repository of disk images, including ready-to-use operating systems

snapshots, that can be copied and used as the basis for new virtual disks. The devices that hold the VM image repository might also host a cloud storage for keeping data that are directly handled by user applications. Such a remote storage is often classified as object storage or block storage, depending on the nature of data organization and basic operations available for user access.

SEMPOLINSKI; THAIN (2010) highlight that cloud frameworks in general also manage network services such as DNS, DHCP, NAT, VLANs, and the subnet organization of the physical machines. They perform virtual bridging of the network, that is required to give each VM a unique virtual MAC address and so enable full network communication in the VM. The cloud framework configures DHCP and DNS processes to handle the MAC and IP addresses of virtual nodes, releasing the cloud operator from many network administration tasks. IP reservation and specification of firewall rules for specific VMs are other important services available in IaaS cloud computing platforms.

A front-end interface for resources management enables users to request VMs, specify their parameters, and obtain needed certificates and credentials in order to remotely access the created VMs. Some front-ends perform various types of scheduling that limit the amount of resources that a user can allocate as well as choose the physical location of those resources. Some of those front-ends implement *de facto* industry standard Application Programming Interfaces (APIs) such as EC2 from Amazon, what might constitute an advantage for implementing hybrid clouds.

At last, the cloud framework itself orchestrates the entire system. It processes inputs from the front-end interface, loads the needed disk images from the repository, requests that a hypervisor in one node sets up a VM and then signals DHCP and IP bridging programs to define and configure MAC and IP addresses for the VM.

Most cloud platforms also include additional modules for load balancing, high availability, performance monitoring, and similar mechanisms. The focus of those components is the automation of VM and application management tasks. They should enable a fast response to failures, peeks of workload, and other events that might affect the quality of service before the system administrator is able to detect the problem and counteract directly.

Next section describes Eucalyptus, that is an example of IaaS private cloud platform deployed in many companies and organizations. There is much documentation about its components, installation, and configuration, what makes it suitable for the study of IaaS private clouds.

2.1.4 Challenges for Cloud Computing

There are various challenges that providers and consumers usually face for implementing, expanding, or maintaining their cloud computing services. Research challenges usually mentioned are: automated service provisioning, interoperability, energy management, traffic

analysis, data security, availability, and performance unpredictability (ZHANG; CHENG; BOUTABA, 2010) (ARMBRUST et al., 2010).

Cloud consumers have little or none control over the underlying computing resources, so the providers are the major actors in charge of ensuring the quality, availability, reliability, and performance of the hardware/software/network infrastructure. In other words, it is vital for consumers to obtain guarantees from providers on service delivery (ZHANG; CHENG; BOUTABA, 2010)

Fault tolerance and reliability were listed by a report from European Commission among the implicit challenges and requirements of cloud computing. This is related to the expected characteristics of a cloud system, where availability is a major service principle, to allow access to resources from anywhere at anytime. Large-scale outages in cloud services such as Amazon EC2, GMail, Google Compute Engine, Apple iCloud, and Microsoft Azure, affected thousands or even millions of users. Such unavailability episodes cause big concerns to potential users, even knowing that the downtime of most traditional data centers and small infrastructures can be even bigger than observed in cloud providers. For both, public and private clouds, it is important to plan the system to achieve high levels of availability, what is a hard task due to the large number of components and the dependency relationships between them.

Performance and, more specifically, scalability are other major concerns for the quality of service being provided. When a cloud computing platform enables provisioning more resources to process a surge of incoming workload, it is difficult to predict how fast the additional capacity will be deployed and if those resources will be sufficient, undersized or oversized.

Figure 2.5 shows results from a survey conducted in 2015 by RightScale company RIGHTSCALE (2015). 930 technical professionals across several organizations were questioned about the benefits that they experienced using cloud. The three most cited benefits were greater scalability, faster access to infrastructure, and higher availability. Such aspects constitute a big part of the expectations of cloud users, and proper capacity and dependability planning must be done for achieving those benefits when creating a new cloud infrastructure, or improving an existing environment.

2.1.5 Eucalyptus Platform

EUCALYPTUS (Elastic Utility Computing Architecture Linking Your Programs To Useful Systems) is a software that implements scalable IaaS-style private and hybrid clouds (EUCALYPTUS, 2010). It is interface-compatible with the commercial services Amazon EC2 – Elastic Compute Cloud – and Amazon S3 – Simple Storage Service (EUCALYPTUS, 2016), while it also emulates the EBS – Elastic Block Store – service (AMAZON, 2012) (EUCALYPTUS, 2009). In general, Eucalyptus and other private cloud platforms use the virtualization capabilities (i.e., hypervisor) of the underlying computer system to enable flexible allocation of computing resources uncoupled from specific hardware (EUCALYPTUS, 2010).

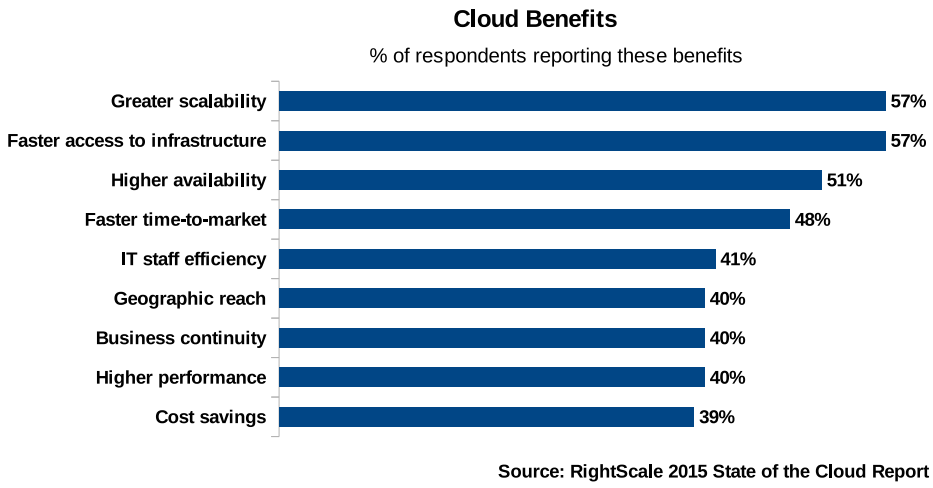


Figure 2.5: Benefits from adopting cloud computing

There are five high-level components in the Eucalyptus architecture, each with its own web service interface: *Cloud Controller, Cluster Controller, Node Controller, Storage Controller*, and *Walrus* (EUCALYPTUS, 2010). Figure 2.6 depicts these main components, which are briefly explained as follows.

The *Cloud Controller (CLC)* is the front-end of the entire cloud infrastructure, exposing and managing the underlying virtualized resources (servers, network, and storage) via Amazon EC2 API (SUN, 2009). This component uses web services interfaces to receive the client requests on one side and to interact with the remaining Eucalyptus components on the other.

The *(Cluster Controller (CC)* usually executes on a cluster front-end machine (EUCALYPTUS, 2010) (EUCALYPTUS, 2009), or on any machine that is able to communicate to both the nodes running Node Controllers and the machine running the CLC. The role of the CC may be summarized in three functions: determining which Node Controller will process the incoming requests for creating a VM instance (i.e., scheduling VM execution), controlling the instance virtual network overlay, and gathering/reporting information about the nodes which compose its cluster (EUCALYPTUS, 2009).

Each physical node which is supposed to run VMs must have a *Node Controller (NC)*. NCs control the execution, inspection, and termination of VM instances on the host where it runs through interaction with the operating system running on the node and with the hypervisor. The NC fetches from Walrus a copy of the VM image which will be instantiated. It queries and controls the system software on its node in response to queries and control requests from the CC (EUCALYPTUS, 2010). An NC must also discover and report the node's physical resources - number of CPU cores, size of memory, available disk space - as well as learn about the state of VM instances on that node (EUCALYPTUS, 2009) (JOHNSON et al., 2010).

The *Storage Controller (SC)* provides persistent block storage for use by the virtual machine instances. It implements block-accessed network storage, similar to that provided by Amazon Elastic Block Store - EBS. An elastic block storage is a Linux block device that can

be attached to a virtual machine but sends disk traffic across the network to a remote storage location. VM instances are not allowed to share the same EBS volume (JOHNSON et al., 2010).

Walrus is a file-based data storage service, that is interface compatible with Amazon's Simple Storage Service (S3) (EUCALYPTUS, 2009). Eucalyptus cloud users can use Walrus to stream data into and out of the cloud as well as from VMs that they have instantiated. In addition, Walrus acts as a storage service for VM images.

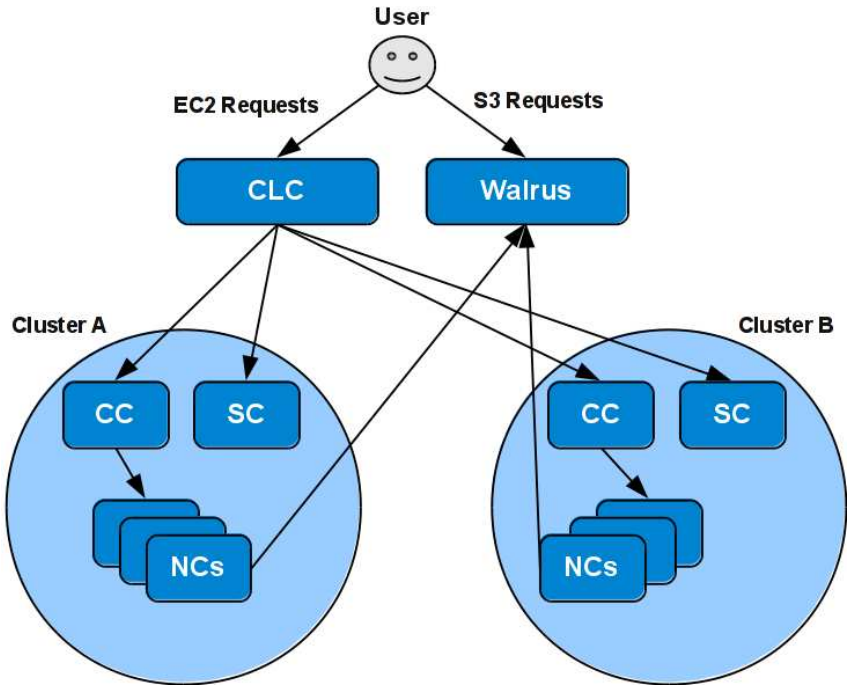
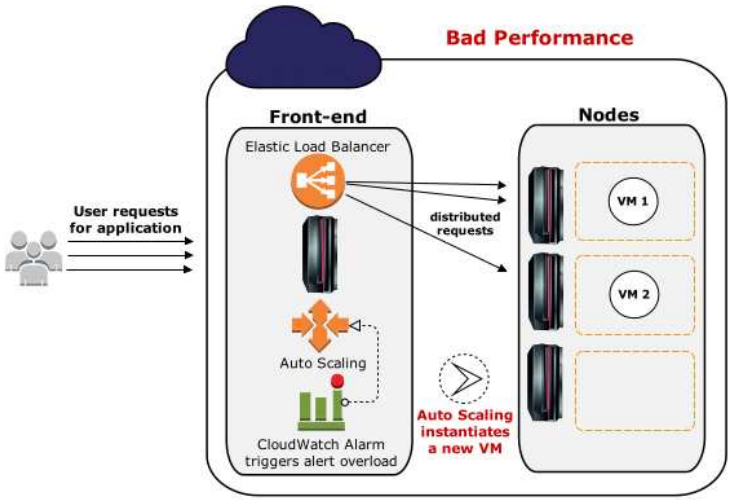


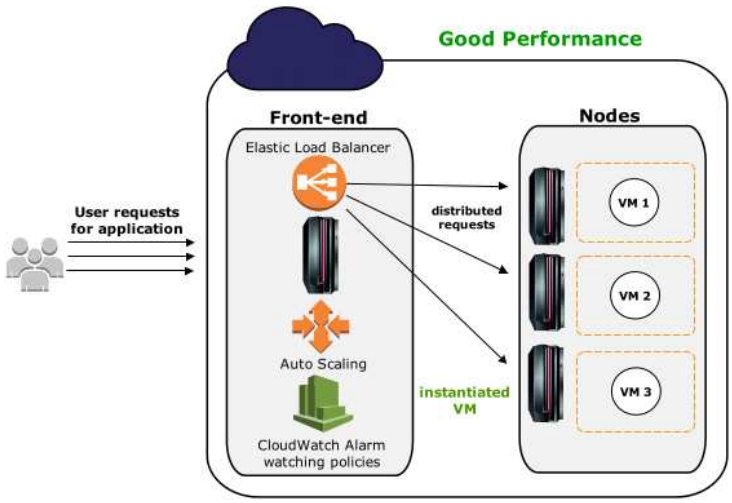
Figure 2.6: Eucalyptus high-level components

The Eucalyptus-based cloud computing environment depicted in Figure 2.6 considers two clusters (A and B). Each cluster has one Cluster Controller, one Storage Controller, and various Node Controllers. The components in each cluster communicate to the Cloud Controller and Walrus in order to service the user requests. A user is able to perform requests for VM instantiation –and other related features– to the Cloud Controller, or file storage requests to Walrus, using the proper tools for each case. Cluster and Storage Controller may be installed in the same machine where Cloud Controller and Walrus are running. In this case, the front-end host will also be responsible for one of the clusters and its corresponding nodes.

Eucalyptus Auto Scaling is a mechanism designed to handle applications that require adding and removing VM instances based on predefined thresholds of selected metrics (e.g.: CPU usage, number of user requests). Auto Scaling is particularly useful for applications that exhibit variability in use by hour, day or week. During demand peaks, the auto scaling mechanism increases the number of VM instances automatically to maintain the performance of the application hosted in the cloud. In a similar manner, when the demand decreases, the number of VM instances might be reduced to minimize costs and save physical resources (EUCALYPTUS, 2014a; AMAZON, 2014a). Eucalyptus Auto Scaling works in conjunction with CloudWatch



(a) Bad performance.



(b) Good Performance.

Figure 2.7: General operation of Eucalyptus Auto scaling mechanism.

and Elastic Load Balancing (ELB) mechanisms. Such an interaction is depicted in Figure 2.7. The ELB distributes client requests to the existing VMs of the target web application. The CloudWatch monitors established metrics (e.g.: average number of web requests per second) periodically (EUCALYPTUS, 2014a). The CloudWatch service inserts data from monitored metrics at arbitrary intervals and extract statistics of the collected data for a particular time interval (time window), with a user-defined granularity (EUCALYPTUS, 2014b). The statistical information allows you to make business and operational decisions. When a certain condition is met (e.g.: a poor performance metric), as it is illustrated in Figure 2.7 (a), the CloudWatch triggers an alarm for Auto Scaling, which instantiates one or more VMs. Shortly thereafter, ELB automatically distributes the requests considering the new VM. The collaboration of these services results in a performance gain and allows an efficient usage of cloud resources, while it might also be used for fault tolerance purposes (EUCALYPTUS, 2014a).

2.2 Dependability and Performance Modeling

The activity of evaluating a system for achieving desired quality of service measures requires specific techniques, which might include prototyping, measurement, and modeling. JAIN (1991) established significant guidelines on the selection of evaluation techniques.

Measurements are not possible if a new system is being planned, instead of just an improvement for an existing one. Building a prototype of the upcoming system just for measurement purposes is not always feasible, due to timing and budget constraints. Therefore, analytical modeling and simulation are the techniques usually chosen when a new infrastructure is being designed. Even if the system already exists, analytical and simulation models can be the best options when the installation or execution of measurement tools would be too intrusive, causing bad effects on the system.

Performance and dependability modeling of computer systems enable one to represent the behavior of a system and compute measures which describe, in a quantitative way, how the service is provided and how much confidence can be put on the system operation. The measures of interest and the purposes of the performance evaluation may influence the choice of modeling technique to be employed.

In performance evaluation studies, some metrics which usually deserve interest are: response time, job completion rate (throughput), and level of resource utilization. These metrics are directly related to the user perception of system performance and they may also highlight the need for improvements.

Besides performance, dependability aspects deserve great attention for the assurance of the quality of the service provided by a system. System dependability can be understood as the ability to deliver a specified functionality that can be justifiably trusted (AVIZIENIS et al., 2004). An alternate definition of dependability is “the ability of a system to avoid failures that are more frequent or more severe, and outage durations that are longer than is acceptable

to the user"(AVIZIENIS et al., 2004). Dependability studies look for determining reliability, availability, security, and safety metrics for the infrastructure under analysis (MALHOTRA; TRIVEDI, 1994).

There are formal techniques which may be used for modeling computer systems and estimating measures related to system availability, reliability, and performance. RBDs (O'CONNOR; KLEYNER, 2012), Fault Trees (O'CONNOR; KLEYNER, 2012), Stochastic Petri Nets (SPNs) (MOLLOY, 1982), Markov chains (BOLCH et al., 2001) and Markov Reward Models (MRMs) (CLOTH et al., 2005) have been used to model many kinds of systems and to evaluate various availability and reliability measures. When dealing with strict performance issues, queueing models, SPNs, and Markov chains are modeling formalisms widely adopted in the literature. The mentioned model types may be broadly classified into non–state-space and state-space models (MACIEL et al., 2011). Non–state-space models (e.g., RBDs, fault trees) enable, in general, a more concise representation of the system than state-space models. State-space models (e.g., Markov chains, SPNs, Stochastic Automata Networks) allow the representation of more complex relationships between system components, such as dependencies involving subsystems and resource constraints (MACIEL et al., 2011). Generally, state-space models require the numerical solution of an underlying system of equations, that will provide useful information such as state probabilities, mean passage time, etc. Discrete event simulation is another way of obtaining some desired metrics from those models. In some special cases, closed-form answers can be derived from state-space models.

Distinct model types may be hierarchically combined, enabling the representation of many kinds of dependency between components, and avoiding the known issue of state-space explosion when dealing with large systems. An example of this hierarchical approach is the usage of combinatorial models to represent the availability relationship between independent subsystems, while detailed or more complex failure and repair mechanisms are modeled with state-based models. Such an approach in seen in (DANTAS et al., 2012a), that combines RBDs and Continuous Time Markov Chains (CTMCs), in (KIM; MACHIDA; TRIVEDI, 2009a), that combines Fault Trees and CTMCs, among other works. Such a composition is usually called as heterogeneous hierarchical modeling. Other kinds of composition are possible and found in the literature. For instance, one CTMC model might yield results that are used as input for another CTMC model, as seen in (MA; HAN; TRIVEDI, 2001) and (GHOSH et al., 2013).

2.2.1 Reliability Block Diagrams

Reliability Block Diagrams are networks of functional blocks connected according to the effect of each block failure on the system reliability (MACIEL et al., 2011). RBDs indicate how the operational state (broken or functioning) of the system´s components affect the functioning of the system. RBD was initially proposed as a model for calculating reliability, but it can be used for computing other dependability metrics, such as availability and maintainability.

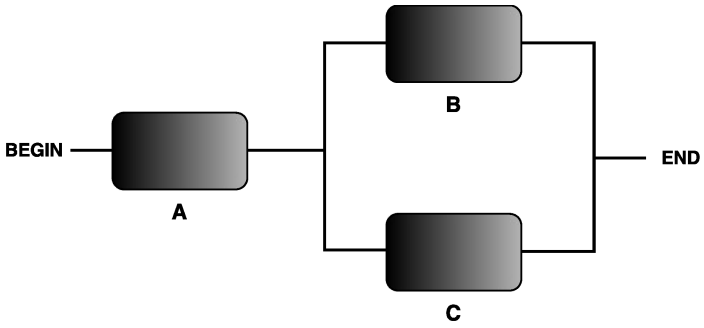


Figure 2.8: Example of RBD

Figure 2.8 depicts an example of RBD for a system with thee components: A, B, and C. RBDs have a source and a target vertex, a set of blocks (usually rectangles), where each block represents a component; and arcs connecting the components and the vertices. The source node is usually placed at the left hand side of the diagram whereas the target vertex is positioned at the right. Graphically, the system is properly working when there is at least one path from the source node to the target node. Therefore, in Figure 2.8, the system is operational if the component A is functioning and either B or C is working too.

In RBDs, the system state is described as a Boolean function of states of its components or sub-systems, where the Boolean function is evaluated as true whenever at least the minimal number of components is operationally enabled to perform the intended functionality (MACIEL et al., 2011)(KUO; ZUO, 2003). The system state may also be described by the respective structure functions of its components or sub-systems, so that the system structure function is evaluated to 1 whenever at least the minimal number of components are operational.

RBDs have been adopted to evaluate series-parallel and more generic structures, such as bridges, stars and delta arrangements. The most common RBDs support series-parallel structures only.

Consider a pure series structure composed of n independent components, where $p_i = P\{x_i = 1\}$ are the functioning probabilities of blocks x_i. These probabilities could be reliabilities or availabilities, for instance. Equation (2.1) is used for computing the system steady-state availability of the series composition in an RBD (KUO; ZUO, 2003). It denotes that the availability of a series system is the product of each component's availability.

$$A_s = \prod_{i=1}^{n} P\{x_i = 1\} = \prod_{i=1}^{n} A_i, \tag{2.1}$$

where A_i is the steady-state availability of block x_i. The reliability of a series system is computed in a similar manner.

Equation (2.2) is used to compute the availability of a pure parallel structure, composed of n independent components. It is based on the fact that the failure probability of a parallel system is the product of the failure probabilities ($P\{x_i = 0\}$) of its components.

$$A_p = 1 - \prod_{i=1}^{n} P\{x_i = 0\} = 1 - \prod_{i=1}^{n} UA_i = 1 - \prod_{i=1}^{n} 1 - A_i, \tag{2.2}$$

where $UA_i = 1 - A_i$ is the unavailability of each block x_i.

Further knowledge on series-parallel, parallel-series, and other RBD structures is found in (MACIEL et al., 2011) and (KUO; ZUO, 2003).

2.2.2 Markov Chains

Markov models are the fundamental building blocks upon which many quantitative analytical performance techniques are built (KOLMOGOROV, 1931; TRIVEDI, 2001). Such models may be used to represent the interactions between various system components, for both descriptive and predictive purposes (MENASCÉ; ALMEIDA; DOWDY, 2004). Markov models have been in use intensively in performance and dependability modeling since around the fifties (MACIEL et al., 2011). Besides computer science, the range of applications for Markov models is very extensive. Economics, meteorology, physics, chemistry and telecommunications are some examples of fields which found in this kind of stochastic [1] modeling a good approach to address various problems.

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

Definition 2.1. A stochastic process is a family of random variables $\{X_t : t \in T\}$ where each random variable X_t is indexed by parameter $t \in T$, which is usually called the time parameter if $T \subset R_+ = [0, \infty)$, i.e., T is in the set of non-negative real numbers. The set of all possible values of X_t (for each $t \in T$) is known as the state space S of the stochastic process (BOLCH et al., 2001).

Let $Pr\{k\}$ be the probability of a given event k occurs. A Markov process is a stochastic process in which $Pr\{X_{t_{n+1}} \leq s_{i+1}\}$ depends only on the last previous value X_{t_n}, for all $t_{n+1} > t_n > t_{n-1} > ... > t_0 = 0$, and all $s_i \in S$. This is the so-called Markov property (HAVERKORT, 2002), which, in plain words, means that the future evolution of the Markov process is totally described by the current state, and is independent of past states (HAVERKORT, 2002).

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

[1] Stochastic refers to something which involves or contains a random variable or variables

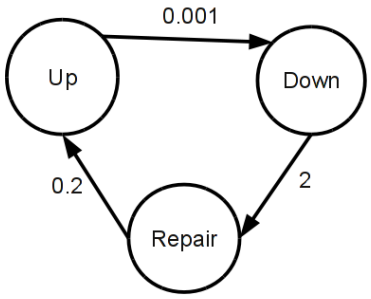


Figure 2.9: Simple CTMC

transitions is driven by a memoryless distribution (BOLCH et al., 2001). In the case of DTMC, the geometric distribution is the only discrete time distribution that presents the memoryless property. In the case of CTMC, the exponential distribution is used.

Markov chains can be represented as a directed graph with labeled transitions, indicating the probability or rate at which such transitions occur. When dealing with CTMCs, such as the availability model of Figure 2.9, transitions occur with a rate, instead of a probability, due to the continuous nature of this kind of model. The CTMC is represented through its transition matrix, often referenced as infinitesimal generator matrix. Considering the CTMC availability model of Figure 2.9, the rates are measured in failures per second, repairs per second, and detections per second. The generator matrix Q is composed by components q_{ii} and q_{ij}, where $i \neq j$ and $\sum q_{ij} = -q_{ii}$. Using the availability model that was just mentioned, considering a state-space $S = \{Up, Down, Repair\} = \{0,1,2\}$ the Q matrix is:

$$Q = \begin{pmatrix} q_{00} & q_{01} & q_{02} \\ q_{10} & q_{11} & q_{12} \\ q_{20} & q_{21} & q_{22} \end{pmatrix} = \begin{pmatrix} -0.001 & 0.001 & 0 \\ 0 & -2 & 2 \\ 0.2 & 0 & -0.2 \end{pmatrix}$$

Equation 2.3 and the system of Equations 2.4 describe the computation of the state probability vector, respectively for transient (i.e., time-dependent) analysis, and steady-state (i.e., stationary) analysis. From the state probability vector, nearly all other metrics can be derived, depending on the system that is represented.

$$\underline{\pi}'(t) = \underline{\pi}(t)Q, \qquad \text{given } \underline{\pi}(0). \tag{2.3}$$

$$\underline{\pi}Q = 0, \sum_{i \in S} \pi_i = 1 \tag{2.4}$$

Detailed explanations about how to obtain these equations may be found in (HAVERKORT; MEEUWISSEN, 1995; BOLCH et al., 2001).

For all kinds of analysis using Markov chains, an important aspect must be kept in mind: the exponential distribution of transition rates. The behavior of events in many computer systems may be fit better by other probability distributions, but in some of these situations

the exponential distribution is considered an acceptable approximation, enabling the use of Markov models. It is also possible to adapt transition in Markov chains to represent other distributions by means of phase approximation, as shown in (TRIVEDI, 2001). The use of such technique allows the modeling of events described by distributions such as Weibull, Erlang, Cox, hypoexponential, and hyperexponential.

2.2.3 Stochastic Petri Nets

Petri Nets (MURATA, 1989) are a family of formalisms very well suited for modeling several system types due to their capability for representing concurrency, synchronization, communication mechanisms, as well as deterministic and probabilistic delays. The original Petri Net does not have the notion of time for analysis of performance and dependability. The introduction of duration of events results in a timed Petri Net. A special case of timed Petri Nets is the Stochastic Petri Net (SPN) (MOLLOY, 1982), where the delays of activities (represented as transitions) are considered random variables with exponential distribution. An SPN can be translated to a CTMC, which may then be solved to get the desired performance or dependability results. This is especially useful because building a Markov model manually may be tedious and error prone, especially when the number of states becomes very large. SPN family of formalisms is a possible solution to deal with such an issue. MARSAN; CONTE; BALBO (1984) proposed Generalized Stochastic Petri Net (GSPN), which is an extension of SPN that considers two types of transitions: timed and immediate. An exponentially distributed firing time is associated only with timed transitions, since immediate transitions, by definition, fire in zero time. For the sake of conciseness, the acronym SPN here is often used for expressing the whole family of models derived from the original SPN model defined by (MOLLOY, 1982).

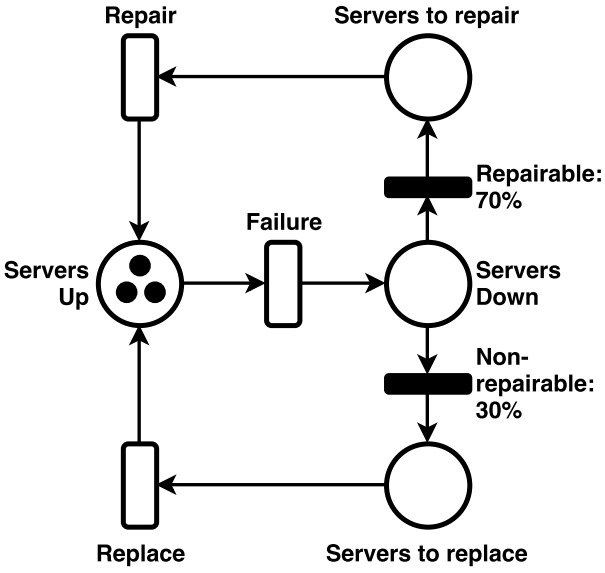


Figure 2.10: Example of GSPN

Figure 2.10 depicts an example of a GSPN model. 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. A vector containing the current number of tokens in each place denote the global state (i.e., marking) of a Petri Net. An inhibitor arc is a special arc type that depicts a small white circle at one edge, instead of an arrow, and they are used to disable transitions if there are tokens present in a given place. The behaviour of Petri Nets in general is defined in terms of a token flow, in the sense that tokens are created and destroyed according to the transition firings (GERMAN, 2000). 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 specify a different firing priority among other immediate transitions.

Figure 2.10 represents the availability of a system comprising three computer servers. Each token in the place **Servers Up** denote one server that is properly running. All three servers might fail independently, by the firing of (exponential) timed transition **Failure**. A token in **Servers Down** might be consumed either by immediate transition **Repairable** or by immediate transition **Non-repairable**. Weights are assigned to each of those transitions to represent the probability of firing one or another. When the failed server can be repaired, the transition **Repairable** puts a token in **Servers to repair**. When the repair is not possible, the transition **Non-repairable** puts the token in **Servers to replace**. The transitions **Repair** and **Replace** fire after exponential delays corresponding to those activities. The probability of having at least one server available, the average number of servers waiting for repair or replacement and other similar metrics can be computed from the underlying CTMC generated from that GSPN.

SPNs also allow the adoption of simulation techniques for obtaining dependability and performance metrics as an alternative to the generation of a CTMC, which is sometimes prohibitive due to the state-space explosion. Regarding SPN and GSPN formal definition and semantics, the reader is referred to (MOLLOY, 1982) (MARSAN; CONTE; BALBO, 1984). Those formalisms were further expanded to allow deterministic delays for timed transitions, generating Deterministic and Stochastic Petri Nets (DSPNs) (GERMAN; MITZLAFF, 1995). Other SPN extensions were proposed in literature for enabling other probability distributions, but the solution of those models require simulation techniques and non-Markovian processes that might not be so computationally efficient as the solution methods for traditional SPNs.

2.3 Sensitivity Analysis

Parametric sensitivity analysis aims at identifying the factors for which the smallest variation implies the highest impact in model's output measure (FRANK, 1978; HAMBY, 1994). The main aim of parametric sensitivity analysis is to predict the effect on outputs (measures) with respect to variations in inputs (parameters), helping to find performance or reliability bottlenecks, and guiding an optimization process (BLAKE; REIBMAN; TRIVEDI, 1988). Another

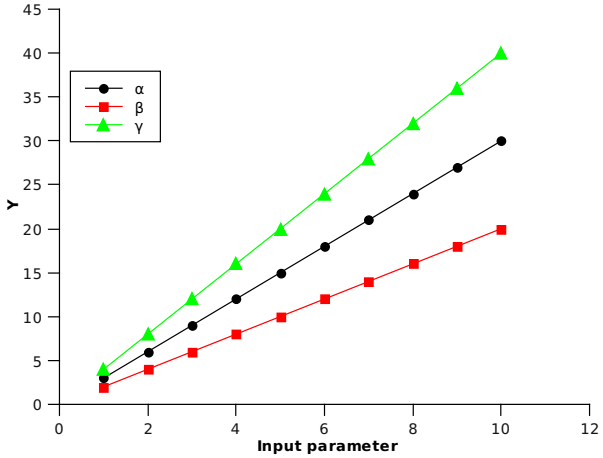


Figure 2.11: Example of plot for one parameter at a time analysis

benefit of sensitivity analysis is the identification of parameters which can be removed without significant effect to the results. Large models, with dozens of rates, may be drastically reduced by using this approach. The results from a sensitivity analysis may be summarized in a list of the input parameters sorted by the amount of contribution each one has on the model output. Such a list is a sensitivity ranking.

There are many ways of performing parametric sensitivity analysis. Factorial experimental design (JAIN, 1991), correlation analysis and regression analysis (ROSS, 2010) are some well known techniques. The simplest method is to repeatedly vary one parameter at a time while keeping the others constant. When applying this method, a sensitivity ranking is obtained by noting the changes to the model output. This method is commonly used in conjunction with plots of input versus output. Such plots enable graphic detection of non-linearities, non-monotonicities, and correlations between model inputs and outputs (MARINO et al., 2008). Unexpected relationships between input and output variables may also be revealed with this approach, triggering the need for further investigations, based on different approaches (HAMBY, 1994). A given percentage of the parameter's mean value may be used as the increment for the cited approach. Each parameter may also be increased by a factor of its standard deviation, in case this information is known (DOWNING; GARDNER; HOFFMAN, 1985).

Figure 2.11 shows a simple example of plot, in which a hypothetical measure Y is plotted against its input parameters: α, β and γ. In this case, the impact caused by each parameter variation is clearly distinguished among them. The result from sensitivity analysis using this approach should be a sensitivity ranking with the following order: $\{1^{st}$: γ, 2^{nd}: α, 3^{rd}: $\beta\}$. Therefore, γ is considered to be the input parameter that cause the major influence on the measure Y.

Although, varying one parameter at a time is less useful in some opportunities. When the amount of parameters is large, the analysis of scatter plots becomes harder, mainly due to the proximity of curves. The difference in magnitude orders is another possible complicating factor, since all parameters cannot be visualized in the same plot, forbidding accurate interpretations

about the differences among parameters influence. Due to such cases, methods that are based on numerical sensitivity indexes should have preference in spite of a visual inspection based on the "one parameter at a time" approach.

Differential analysis is the backbone of many parametric sensitivity analysis techniques (HAMBY, 1994). Differential sensitivity analysis is performed by computing the partial derivatives of the measure of interest with respect to each input parameter. Thus, the sensitivity of a given measure Y, which depends on a specific parameter θ, is computed as shown in Equation (2.5), or (2.6) for a scaled sensitivity.

$$S_\theta(Y) = \frac{\partial Y}{\partial \theta}, \tag{2.5}$$

$$SS_\theta(Y) = \frac{\lambda}{Y}\frac{\partial Y}{\partial \theta}. \tag{2.6}$$

$S_\theta(Y)$ is the sensitivity index (or coefficient) of Y with respect to θ, and $SS_\theta(Y)$ is the scaled sensitivity index, commonly used to counterbalance the effects of largely different units between distinct parameters values.

Specific methods for performing the differential sensitivity analysis in analytic models are needed when there is no direct closed-form equations for computing the measure of interest and finding its derivative expression. Many papers have already described how to apply differential sensitivity analysis in a variety of analytic models, including CTMC (BLAKE; REIBMAN; TRIVEDI, 1988) (OU; DUGAN, 2003), MRM (ABDALLAH; HAMZA, 2002), GSPN (MUPPALA; TRIVEDI, 1990), and Queuing Networks (YIN et al., 2007). When dealing with hierarchical or composite models, the analysis needs to consider all parameters from each model, determining their impact to the global measure of interest. A closed-form equation based on measures of each sub-model may be used for those cases, so that the partial derivatives computed for the sub-models enable us to obtain a complete sensitivity ranking. This is one of the approaches used in this paper.

Differential sensitivity analysis is closely related to the approach in which one parameter at a time is changed and plotted against the result in the measure Y. The sensitivity coefficient may be understood as the slope of the corresponding line for a specific point in the plot. From this view, it is possible to notice that interpretation of analysis results must be more careful if the parameters can be far removed from the base values, mainly if the function is not linear or not monotonic. An example of a non-linear and non-monotonic function is $Z = (\alpha-3)^2 + (\beta-4)^2 + (\gamma-2)^2$, depicted in Figure 2.12, in which the slopes of curves vary in each point of the analysis, so the sensitivity of Z with respect of each parameter quantify the impact of changes just in regions close to that analyzed point.

When carrying computer performance and dependability analyses, it is common looking for incremental improvements in system configuration, so the localized range of sensitivity results is well fitted in this context.

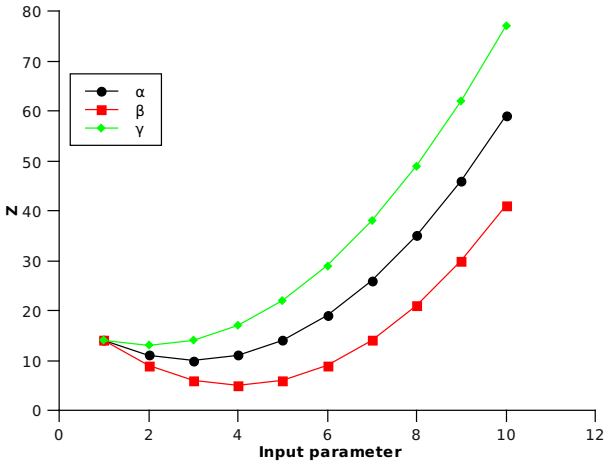


Figure 2.12: Plot for non-linear and non-monotonic function

Partial derivatives are an important means of performing sensitivity analysis, but they may not properly evaluate the sensitivity with respect to integer-valued parameters, because the approach is designed for parameter input values in a continuous domain. An approach to address such an issue is based on calculating the percentage difference when varying one input parameter from its minimum value to its maximum value. Hoffman and Gardner (HOFFMAN; GARDNER, 1983) advocate utilizing each parameter's entire range of possible values to compute parameter sensitivities. Equation (2.7) shows the expression for this approach, where $max\{Y(\theta)\}$ and $min\{Y(\theta)\}$ are the maximum and minimum output values, respectively, computed when varying the parameter θ over the range of its n possible values of interest. If $Y(\theta)$ is known to vary monotonically, so only the extreme values of θ (i.e., θ_1 and θ_n) may be used to compute $max\{Y(\theta)\}$, $min\{Y(\theta)\}$, and subsequently $S_\theta(Y)$.

$$S_\theta(Y) = \frac{max\{Y(\theta)\} - min\{Y(\theta)\}}{max\{Y(\theta)\}}, \tag{2.7}$$

where

$$max(Y(\theta)) = max\{Y(\theta_1), Y(\theta_2), ..., Y(\theta_n)\}, \tag{2.8}$$

and

$$min(Y(\theta)) = min\{Y(\theta_1), Y(\theta_2), ..., Y(\theta_n)\}. \tag{2.9}$$

Another important method to assess the importance of each parameter is the analysis of a factorial experimental design. Design of experimentss (DOEs) techniques can be used to determine simultaneously the individual and interactive effects of many factors that may affect the output measures (JAIN, 1991). In DOE terminology, each parameter is called a factor and each value possibly assigned to each factor is a level. DOE involves choosing a given number of levels for each factor and running the model for all combinations of the levels. The analysis may be prohibitive due to a large number of factors or levels, which would require several model runs and a huge computation time for some cases (HAMBY, 1994). A fractional factorial design may

be chosen for such cases, or the number of parameters may first be reduced to an acceptable value, through the ranking obtained by differential sensitivity analysis, for example, and then the factorial analysis may be applied.

2.4 Concluding Remarks

This chapter provided theoretical foundations that are not exhaustive but essential for the reader awareness regarding the building blocks that compose this thesis. The background on dependability and performance modeling as well as on sensitivity analysis of those models enable understanding the application of such concepts to the field of cloud computing planning and evaluation.

Specific software tools can play an important role for fulfilling the gap of knowledge that a cloud system administrator may have on some of the concepts presented here. Such tools can ease the application of the proposed methods, described further, for their target audience.

3

Related works

The works found during the literature review are described in this chapter. The papers are divided into two categories: **Dependability and performance evaluation of cloud computing**, and **Sensitivity analysis of analytical models**, which are the main topics in this thesis. The following sections are not intended to provide an exhaustive view of the works published on those topics, but rather to point out significant advances which go towards a similar direction as this research do, or give basis for future extensions.

3.1 Dependability and Performance Evaluation of Cloud Computing

The work presented in (IOSUP et al., 2011) analyzes the performance of cloud computing services for scientific computing workloads. The authors do not use analytical or simulation models. Instead, they carry out an empirical evaluation of the performance of four commercial cloud computing services. The testbed data are then used in a trace-based simulation to compare the performance and cost of clouds and other computing platforms, such as grids, for general and scientific computing workloads. Their results indicate that the current clouds need an order of magnitude in performance improvement to be useful to the scientific community, and show which improvements should be considered first to address this discrepancy between offer and demand.

CHAISIRI; LEE; NIYATO (2012) propose an algorithm for optimization of costs to provision VMs in public clouds. The authors formulate a stochastic programming model which takes into account the demand for VMs and respective costs, but they do not evaluate any specific performance or dependability metrics regarding the service provided in the cloud.

GHOSH et al. (2010) propose a composite modeling approach for addressing performability issues of IaaS clouds. Their work uses outputs from interacting performance models to feed up an availability model, so metrics such as job rejection probability and mean response delay are obtained as final result. A sensitivity analysis, through variation of one parameter at a time, enables them quantifying the effects of variations in workload, failure rates, and sys-

tem capacity (number of physical machines) on IaaS cloud service quality. The authors also quantify the reduction of state space achieved by the composite modeling when compared to a monolithic model, what is reflected in smaller solution time and required main memory storage and space.

The work of (GHOSH et al., 2013) is similar to the one proposed in (GHOSH et al., 2010), but it focuses on pure performance models. They use an approach of multi-level interacting stochastic sub-models, where the overall model solution is obtained iteratively over individual sub-model solutions. The authors mention the need for a formal sensitivity analysis in this composite model. The large number of parameters brings out the need for determining the most important ones, and revealing bottlenecks in the system.

Outstanding surveys on cloud computing, such as (ARMBRUST et al., 2010), (RIMAL et al., 2011), and (SUN, 2009) have been mentioning availability and reliability as major concerns for cloud infrastructures. Therefore, those systems must rely on various fault tolerance mechanisms, such as redundancy, for coping with failures, so that resources are accessible anywhere and anytime as expected (ARMBRUST et al., 2009).

MENDEZ MUNOZ et al. (2013) propose an architecture for resilient services on hybrid clouds, which should monitor availability of distinct cloud providers and provide graceful degradation for adapting to outages or unresponsiveness of those providers. CUOMO et al. (2013) provide mechanisms for monitoring and predicting quality of service and SLA metrics in federated private clouds. They forecast resources availability by measuring mean time to failure (MTTF) and mean time to repair (MTTR). Hearbeat remote monitoring and VM boot time logging are the main mechanisms for collecting MTTF and MTTR values and therefore computing system availability. It is worth noting that VMs are the single resource considered in that work, and analytical or simulation models are not used.

SUN et al. (2010) propose system-level virtualization through identical VM replicas as fault-tolerance mechanism for achieving dependability improvement. In (SUN et al., 2010), the authors also propose a combinatorial model for evaluating dependability and security of heterogeneous cloud environments. The authors consider a series combination of components, so the system dependability is computed through product of hardware and software dependability metrics.

Some related works employ hierarchical and composite modeling approaches to tackle the complexity of evaluating cloud systems. Although, many of those works do not address software dependability nor consider the influence of adding new equipments to provide redundancy for existing architectures. WEI; LIN; KONG (2011) use hierarchical method and proposes hybrid models combining RBD and GSPN (Generalized Stochastic Petri Net) models to analyze the relation between reliability and servers consolidation ratio, as well as the relation between availability and the workload experienced by the cloud-based datacenter. In (CHUOB; POKHAREL; PARK, 2011), the authors propose a private cloud environment suitable for e-government purposes and provide a hierarchical model to predict the availability of the proposed

Eucalyptus-based architecture. The hierarchical model proposed in (CHUOB; POKHAREL; PARK, 2011) uses distinct Markov chains for cluster level and node level, while the operation level is described in a non-formal manner. LONGO et al. (2011) propose an availability model for IaaS clouds that deal with distinct pools of physical machines where VMs are instantiated. They compare accuracy and time of solution for a monolithic Stochastic Reward Net (SRN) model and a composite model to demonstrate the benefits of the latter approach. It is also important to highlight that (LONGO et al., 2011) does not consider cloud management components in their work.

In (DANTAS et al., 2012a) and (DANTAS et al., 2012b), the authors propose hierarchical availability models for evaluating private cloud systems. Such a modeling approach considers replication of specific private cloud components, such as Eucalyptus Cloud, Cluster, and Node controllers, dealing with both hardware and software faults. This approach is further adapted to assess capacity-oriented availability (COA) in (DANTAS et al., 2015). That work builds models for predicting the average computational power available in case of partial failures of a multi-cluster cloud. Such models enable comparing the costs and COA of private and public clouds.

This thesis presents differential sensitivity analysis on models that are similar to those found in (DANTAS et al., 2012b), in order to identify bottlenecks and determining which components deserve priority for system availability improvements. Another original contribution of the current work is proposing a general approach for performing parametric sensitivity analysis on hierarchical models, using it for improving various types of cloud systems.

3.2 Sensitivity Analysis of Analytical Models

In the fields of performance and dependability evaluation, it is possible to find a number of researchers that have already demonstrated how to perform parametric sensitivity analysis in some analytic models. One of the seminal works in this topic is found in (BLAKE; REIBMAN; TRIVEDI, 1988), which presents the foundations for transient sensitivity analysis in continuous time Markov chains and Markov reward models, and shows how the sensitivity functions can guide system optimization, model refinement and the detection of reliability and performability bottlenecks.

The development of differential sensitivity analysis methods followed similar ways for all modeling formalisms, mainly when only state-space models are considered.

The following sections summarize some related works on the field of sensitivity analysis of analytic models, focusing specifically on Markov chains, queueing systems, and SPNs.

3.2.1 Differential Sensitivity Analysis on Queueing Systems

Queueing system is one example of analytic model whose sensitivity analysis has been described in literature. YIN et al. (2007) give sensitivity formulas for the performance of $M/G/1$[1] queueing systems, which are described by semi-Markov processes. They show that the embedded Markov chain of a $M/G/1$ model may be used to provide the desired steady-state sensitivity measures. This is possible because the semi-Markov process has always the same steady-state probabilities as the embedded Markov chain. In (YIN et al., 2007), the sensitivity analysis of $M/M/1$ and $M/C_2/1$[2] queueing systems is also discussed, since they can be considered as specialized versions of the $M/G/1$ case.

OPDAHL (1995) presents sensitivity functions for the performance of open queue networks[3], while CAO (1996) proposes an approach for sensitivity estimation in closed queueing networks. Instead of actual differentiation of the performance measure of interest, the algorithm proposed in (CAO, 1996) uses a sample path of the model to estimate the derivative of the steady-state probability vector.

LIU; NAIN (1991) propose general formulas to quantify the effects of changing the model parameters in open, closed, and mixed product-form queueing networks. These formulas include the derivative of the expectation of known functions of the state of the network with respect to any model parameter (i.e., arrival rate, mean service demand, service rate, visit ratio, traffic intensity). The sensitivity functions for the throughput and queue length are presented in that paper, which also demonstrates an example of cost-based optimization.

3.2.2 Differential Sensitivity Analysis on Markov Chains

Stages of the differential sensitivity analysis of Markov chains include computing the derivative of the rate generator matrix and the differentiation of equations used in Markov chain solution methods, or even the development of new sensitivity computation techniques. MARIE; REIBMAN; TRIVEDI (1987) present a sensitivity analysis method regarding transient and cumulative measures in acyclic Markov chains. The ACE (**A**cyclic Markov **C**hain **E**valuator) algorithm is used to find the state probabilities of an acyclic CTMC as a symbolic function of t, and it is adapted to the compute the respective sensitivity functions.

BLAKE; REIBMAN; TRIVEDI (1988) show how to compute the same measures of the (MARIE; REIBMAN; TRIVEDI, 1987) work, but using the uniformization technique (HEIDELBERGER; GOYAL, 1987), which allows the analysis of more general models, with cycles. The sensitivity functions are applied in a reliability/performability study, which also introduces the sensitivity of expected reward rate and a specific sensitivity function for the mean time to

[1] A queue $M/G/1$ has a service time that follows an arbitrary (general) distribution, in contrast to the exponential nature of service time in a queue $M/M/1$ (KLEINROCK, 1975).

[2] A queue $M/C_2/1$ has a service time that follows a two-stage Coxian distribution (YIN et al., 2007)

[3] Queueing networks whose operations have transaction workload intensities are open, while other queueing networks are closed or mixed (OPDAHL, 1995).

failure (MTTF) of a system.

Another related study is shown in (OU; DUGAN, 2003), that developed an approximate approach for the computation of sensitivity analysis in acyclic Markov reliability models, reducing the computation time for large models. That approach is used for solving a dynamic fault tree and hence assessing the importance of each component according to its failure probability. That work also presents the computation of sensitivities for modules of some components, that can be combined to produce the system level sensitivities. A chain-rule approach is used to calculate sensitivity measures for the separate modules, and to combine them hierarchically for higher-level results.

In (SATO; TRIVEDI, 2007), two distinct Markov chains were created for representing the response time and the reliability of a travel agent system. Those models were analyzed individually. They performed a sensitivity analysis of response time and reliability metrics with respect to each parameter. Despite using Markov chains for computing the metrics of interest, closed-form equations are found for both measures, and the differential sensitivity analysis is carried out using these equations. The authors highlight that closed-form equations can not be found for all systems, so a model-based sensitivity analysis would be helpful for a broader range of situations.

MATOS JÚNIOR (2011) highlights some factors that must influence the decision regarding the use of scaled and unscaled sensitivity indices for performance and availability CTMC models.

3.2.3 Differential Sensitivity Analysis on Petri Nets

MUPPALA; TRIVEDI (1990) introduce a process to compute sensitivity functions of GSPNs. Since the reduced reachability graph of a GSPN is a continuous-time Markov chain, it is possible to translate the process of sensitivity analysis in CTMCs to a GSPN-based sensitivity analysis. MUPPALA; TRIVEDI (1990) demonstrate the derivative of equations for steady-state, transient and cumulative measures in their work, which also includes the implementation of sensitivity analysis features in a modeling software package (HIREL; TU; TRIVEDI, 2010).

In (CIARDO et al., 1993), the definition of Stochastic Reward Nets is complemented by the demonstration of sensitivity formulas for that model. It follows a process that is similar to that for GSPN models, in which tangible and vanishing markings shall be identified first, as well as the transitions that may occur in these sets of markings.

(CHOI; MAINKAR; TRIVEDI, 1993) constitutes the first work to elaborate a method for parametric sensitivity analysis of deterministic and stochastic Petri nets (DSPNs) (GERMAN; MITZLAFF, 1995), which are an extension to GSPN models. Some characteristics of the solution for GSPNs, found in (MUPPALA; TRIVEDI, 1990), are used in that work, but the analysis of a DSPN requires additional steps, since other stochastic processes (semi-Markov process and non-Markov DSPN process) are involved in this type of model.

3.3 Comparison of Main Related Works

Table 3.1 summarizes the main related works mentioned in this chapter, establishing a comparison between them and this Ph.D. thesis with respect to four subjects: performance and dependability models; sensitivity indices, clod computing, and optimization.

Table 3.1: Comparison table of related works

	Analytical, Simulation Models	**Sensitivity indices**	**Cloud computing**	**Optimization**
(Sato; Trivedi, 2007)	Single model	Yes	No	No
(Yin et al., 2007)	Single model	Yes	No	No
(Chaisiri; Lee; Niyato, 2013)	No	No	Yes	Yes
(Ou; Dugan, 2003)	Hierarchical non-heterog.	Yes	No	No
(Chuob; Pokharel; Park, 2011)	Hierarchical non-heterog.	No	Yes	No
(Longo et al., 2011)	Hierarchical non-heterog.	No	Yes	No
(Ghosh et al, 2010)	Hierarchical non-heterog.	No	Yes	No
(Dantas et al., 2012a,b)	Hierachical heterog.	No	Yes	No
(Wei; Lin; Kong, 2011)	Hierarchical heterog.	No	Yes	No
This Ph.D. thesis	Hierarchical heterog.	Yes	Yes	Yes

The papers (SATO; TRIVEDI, 2007) and (YIN et al., 2007) deal with sensitivity indices, but applied to single (i.e., non-hierarchical) models. Their application domain is different than cloud computing and no specific optimization technique is presented. (CHAISIRI; LEE; NIYATO, 2012) deals with optimization related to cloud computing, but no performance or dependability models are proposed, and it does not perform sensitivity analysis.

The sensitivity analysis presented in (OU; DUGAN, 2003) is only applied to compositions of Markov models (i.e, hierarchical homogeneous). Those models are not from the cloud computing domain. (CHUOB; POKHAREL; PARK, 2011), (LONGO et al., 2011), and (GHOSH et al., 2010) address hierarchical non-heterogeneous models for cloud computing systems, although they do not propose nor compute numerical sensitivity indices for those models.

(DANTAS et al., 2012b) and (WEI; LIN; KONG, 2011) propose hierarchical heterogeneous models for evaluating dependability metrics of cloud computing systems. None of both works deal with sensitivity indices or optimization techniques. The PhD thesis presented here covers those four subjects that had not been previously combined in the literature reviewed so far.

3.4 Concluding Remarks

This chapter highlighted the main works that were found during the literature review on the mentioned topics. Although, it is important to emphasize that this is not an exhaustive view of the published papers and related research works. There might be other articles and theses that made significant advances in this field, but to the best of our knowledge the combination of characteristics described in Table 3.1 is one of the major factors that distinguishes this work from the current state of the art.

4

Approach for identification of availability and performance bottlenecks in cloud systems

This PhD thesis proposes an approach for identifying availability and performance bottlenecks in cloud computing infrastructures. This approach focuses on sensitivity analysis of hierarchical models and enables the identification of points for improvement at different levels of hardware and software that constitute IaaS cloud systems.

4.1 Supporting methodology

The techniques proposed here are supported by a methodology that is illustrated by the flowchart in Figure 4.1. It contains the main activities that are required for the proper definition and analysis of models.

Each activity of the supporting methodology is described in details as follows.

Create a top-level model: Given a cloud computing infrastructure, we need to obtain a general view of system performance or dependability that enables creating a top-level model. This is the main model which may describe the interconnection of subsystems in the IaaS environment, the global activities for processing user requests, or the overall dependability relationship between the cloud components (e.g., VMs, processing nodes, cluster managers, remote storage devices, etc.). RBDs and SPNs are among the most proper formalisms for such a main model, due to their conciseness for representing large systems. CTMCs, Queueing Networks, and other models can still be used here, but they may make harder to handle the complexity of a broader system view and connecting with sub-models created in the next step. The top-down modeling induces the creation of condensed models, that only include detailed behavior for the modules and sub-systems which are really well-known or are expected to be relevant for the overall performance and dependability.

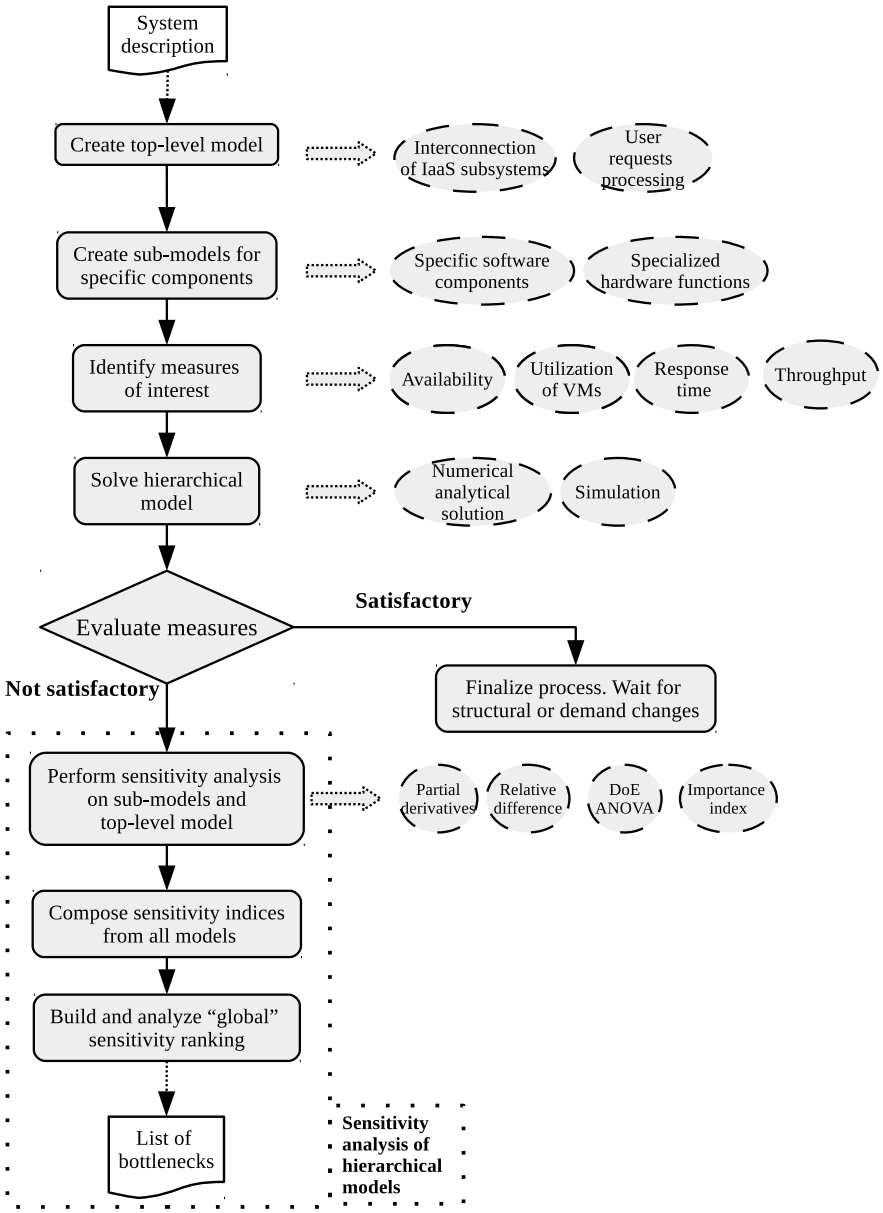


Figure 4.1: Supporting methodology for bottleneck identification on cloud systems

- Precondition: prior knowledge about the cloud system to be modeled and possible modeling formalisms.
- Inputs: intended type of analysis (e.g., availability, reliability, performance); list of major components or subsystems; parameter values; description of dependency or interconnection between them.
- Actions: choice of modeling formalism; creation of the top-level model.
- Products: top-level model.
- Postconditions: top-level model is parameterized and ready to be refined on sub-models.

Create sub-models of specific components: We create sub-models for representing the internal behavior of specific sub-systems or components. The sub-models might deal with specialized hardware or software mechanisms that are not present in the main model, or provide fine-grained evaluation of VM and application behavior. Redundancy schemes, a single sub-system operation, and performance degradation on particular software components are examples that are usually represented through sub-models. It is worth stressing that the proposed methodology does not constrain the amount of levels in the hierarchical models. A sub-model may comprise other lower level models in order to reduce modeling complexity for example. The system analyst should be cautious regarding possible accuracy loss due to excessive levels in the model. Distinct formalisms may be chosen for each sub-model, depending only on the suitability for describing that specific sub-system and the knowledge of the modeler.

- Precondition: there is a main (top-level) model with some sub-systems to be refined.
- Inputs: list of subsystems that can be refined; description of components, parameter values, and internal functioning of each sub-system.
- Actions: choice of modeling formalisms; creation of sub-models and definition of how they are connected to the main model.
- Products: sub-models and connections between main model and sub-models.
- Postconditions: sub-models are parameterized and connected to the main model

Identify measures of interest: Measures of interest must be identified, taking into account the essential information that the model can provide for diagnosing the system performance or dependability status. User-centered metrics (e.g., response time, and downtime) are preferred in many cases, once users of cloud-based systems might be anywhere and the system-centered metrics (e.g., throughput, CPU utilization) might not correspond exactly to the quality of service

perceived by end users. On the other hand, when dealing directly with IaaS, some system-centered metrics might be very important because end users are in fact systems administrators that may want to know VMs utilization or network throughput, for instance.

- Precondition: top-level model and sub-models were created and properly parameterized.
- Inputs: list of measures that can be computed with models; description of performance or availability indicators from the point-of-view of system administrators and end-users.
- Actions: choice of measures of interest for model evaluation.
- Products: chosen metrics.
- Postconditions: all information for solving the main model and sub-models is defined.

Solve hierarchical model: The solution of hierarchical model is the next step in this methodology. Sub-models which do not depend on results of other models are solved first, and the output metrics are assigned to the corresponding input parameters in the main model or other dependent sub-models. The solution method (i.e., numerical analysis or simulation) may vary for each model, depending on constraints of the modeling formalism.

- Precondition: all information for solving the main model and sub-models is defined; solution tools are available.
- Inputs: list of measures of interest; main model and sub-models properly assembled; solution methods and tools;
- Actions: solution of hierarchical model, with computation of chosen measures.
- Products: values of measures of interest.
- Postconditions: the model was successfully solved, and the values of measures of interest were computed

Evaluate measures: The evaluation of measures is the activity of comparing the output values from the hierarchical model to reference values which fulfill SLAs or expectations of the end-users and systems administrators. When the values of computed measures are satisfactory, the improvement process stops, and it is restarted when the system is modified due to events such as replacement of broken or outdated components, or even due to significant changes in user demands. If there is at least one metric of interest that has not achieved a satisfactory level. The identification of potential improvements takes place by means of sensitivity analysis.

- Precondition: the values for the measures of interest were properly computed.
- Inputs: values of measures of interest; description of end-users or systems administrators expectations, or SLAs.
- Actions: define whether estimated measures are satisfactory or not.
- Products: definition (satisfactory or not satisfactory)
- Postconditions: sensitivity analysis will be conducted if measures are not satisfactory, or the workflow will stop until the occurrence of structural or demand changes in the system

Perform sensitivity analysis on sub-models and top-level model: In order to find the sensitivity indices for a hierarchical model, the main model and the sub-models must initially be evaluated in separate. The computation of sensitivity indices $S_{p_{ij}}(f(M_i))$ for each sub-model M_i will provide the impact of parameters p_{ij} to a metric $f(M_i)$. For the main model M^*, we must compute the sensitivity of a metric $g(M^*)$ with respect to each input parameter p_i^*. Partial derivatives, percentage difference, DoE ANOVA, and reliability importance are possible methods for this task. A single sensitivity analysis method may be used for all models comprising the hierarchical model. When there are closed-form equations to solve all models, the partial derivatives method can provide all required indices, and ease the task of building a unified sensitivity ranking with higher accuracy. Distinct methods might also be used for each model to deal with specific solution constraints or analysis preferences. In such a case, the composition of indices, performed in next step, might require greater attention.

- Precondition: the values that were estimated for measures of interest are not in satisfactory levels.
- Inputs: top-level model and sub-models, sensitivity analysis methods and tools
- Actions: compute sensitivity indices for top-level model and for the sub-models.
- Products: sensitivity indices of each model.
- Postconditions: the sensitivity indices of all parameters from top-level model and sub-models are available for further composition.

Compose sensitivity indices from all models: The composition of sensitivity indices from all models is the next step in our methodology. The method of composition depends on the types of indices obtained in the previous activity. Section 4.2 presents more details on this specific activity.

- Precondition: the sensitivity indices from all models were computed.

- Inputs: all sensitivity indices from top-level model and sub-models, considering every parameter; composition techniques; top-level model and sub-models.
- Actions: compose the indices according to the types of indices obtained and types of models employed.
- Products: composite sensitivity indices.
- Postconditions: all sensitivity indices could be composed to generate a unified ranking considering parameters from all models.

Build and analyze "global" sensitivity ranking: After obtaining the composite indices, we must build a unified sensitivity ranking which shows the impact of each parameter from all models on the metric of interest. The analysis of such a ranking enables identifying the top-ranked parameters, that deserve priority on actions for overall system improvement. As soon as changes are performed, the hierarchical model must be evaluated again, computing metrics of interest with the new parameters setup.

- Precondition: all sensitivity indices could be composed to generate a unified ranking considering parameters from all models.
- Inputs: composite sensitivity indices for all parameters from all models.
- Actions: build unified sensitivity ranking, by sorting the sensitivity indices, and identify the parameters which are bottlenecks for the entire system.
- Products: list of bottlenecks (i.e., most impacting parameters).
- Postconditions: the bottlenecks were identified and can be used as priority targets for system improvements.

The last three activities —enclosed in the dotted rectangle—constitute the core of bottleneck identification. They compose the sensitivity analysis of hierarchical models proposed in this Ph.D. thesis, that is explained in details on Section 4.2.

4.2 Sensitivity Analysis of Hierarchical Models

In order to explain the development of sensitivity analysis tailored for hierarchical models, let us first use the introductory example of sensitivity analysis for single models MATOS JÚNIOR (2011). The process for a single Markov chain model is illustrated in Figure 4.2. Initially, a symbolic generator matrix (Q) for the Markov chain is created. That matrix contains all transition rates as symbolic expressions using the input parameters from the model. The partial derivative of the Q matrix with respect to one of the parameters produces another matrix (V). The V matrix

is used in the computation of the respective sensitivity index that might indicate, for instance, the impact of parameter q on the probability of system being in state 3. If the same method is applied to all parameters, we obtain a list of sensitivity indices that is called a sensitivity ranking, when sorted according to the absolute values.

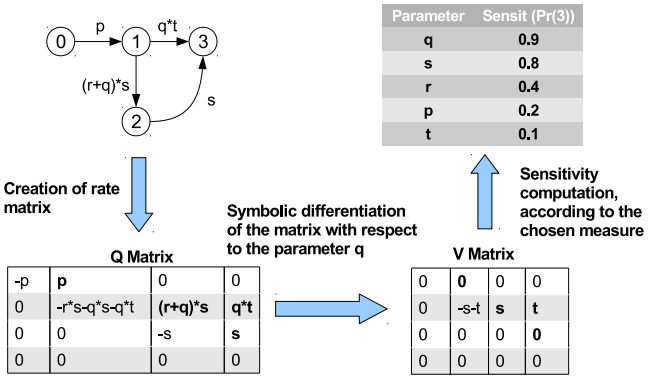


Figure 4.2: Process of sensitivity index computation with symbolic differentiation

Frameworks for symbolic computation such as GiNaC (GINAC, 2015) and Symja (SYMJA, 2015) allow solving partial derivatives, that is an essential step for computing the aimed sensitivity indices. Other mathematical software, such as Mathematica (WOLFRAM, 2016) and R (R-PROJECT, 2016), may also be considered in the tool set used to automate that process.

From the sensitivity analysis of single CTMCs, we developed methods to combine the sensitivity indices obtained in distinct models created using CTMC and RBD formalisms. This kind of hierarchical model is useful for modeling the availability of IaaS private clouds, as demonstrated in (DANTAS et al., 2012a). Further, other methods of composition of sensitivity indices were developed or adapted to address scenarios containing stochastic Petri nets, based on the existing theoretical framework for sensitivity analysis of GSPNs and SRNs (MUPPALA; TRIVEDI, 1990).

Figure 4.3 illustrates an overview of the sensitivity analysis process aimed for a given system which is represented by a combination of three different models: a CTMC, an RBD, and an SPN. Consider that the RBD is the top-level model, and the other two are sub-models that represent the detailed behavior of blocks B and C, for example. The sensitivity analysis of the hierarchical model must generate a unified ranking that comprises parameters from the three models. This unified ranking must accurately express the order of importance of those parameters for the measure of interest in the top-level model (i.e., the RBD). The individual rankings of the sub-models and the top-level model are computed first, and their indices are combined to produce unified ranking. The position of a parameter in the sensitivity rankings of the isolated models might change on the unified ranking, reflecting that the importance of a component for the metrics of given subsystem is not necessarily equivalent to the importance that this component will have for the system as a whole.

The composition of sensitivity indices to build a unified ranking may require distinct

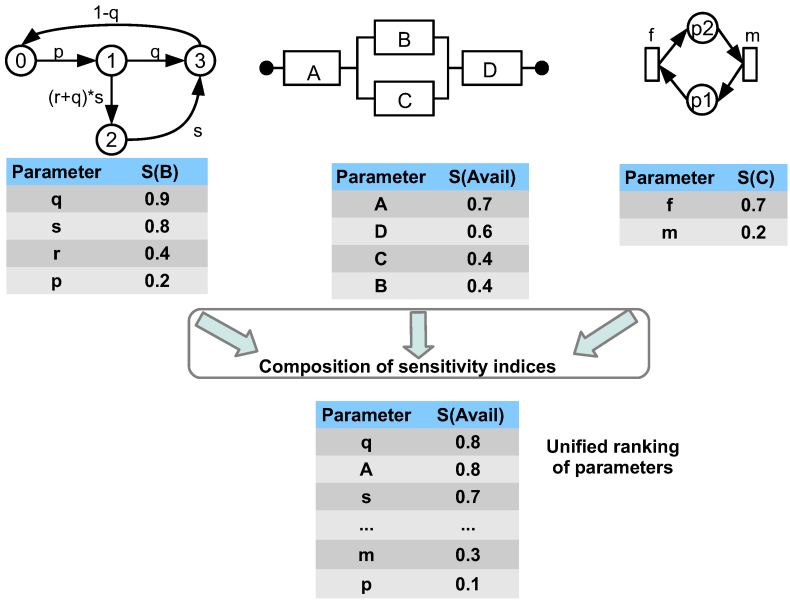


Figure 4.3: Overview of composition of sensitivity indices for distinct models

techniques, according to the types of models involved, and the specific measures of interest. We present here techniques for the following cases: (i) availability and reliability modeling using RBD as top-level model; and (ii) performance, availability, and reliability modeling using SPN as top-level model. The choice of those two formalisms is based on their expressiveness power to represent large systems (as IaaS clouds can be) while keeping the models legibility. Availability and reliability evaluation could also employ Fault Tree as top-level model, with the same benefits as RBD, but this case is not covered here for the sake of conciseness.

The proposed techniques use examples of CTMC as the preferred formalisms for sub-models, due to its flexibility for both, availability and performance studies, and possibility of obtaining closed-form equations. Despite the examples with CTMC, the techniques do not limit the usage of the sub-models for only that formalism.

4.2.1 Composition of Sensitivity Indices with RBD as Top-level Model

The case that comprises RBD as top-level model is evaluated here under two circumstances: using partial derivative for all models —what is particularly suitable when the sub-models are CTMCs that might be described through closed-form equations; or using distinct types of sensitivity indices—e.g., partial derivatives for the RBDs and percentage difference for sub-models, that might be SPNs, CTMCs, or models of other formalisms.

RBDs can be expressed by means of structural equations, that might undergo partial differentiation for obtaining the sensitivity indices. When the RBD is the top-level model which has some blocks refined by means of sub-models (e.g, CTMCs), this differentiation must take into account all parameters of the sub-models. The derivative structural equation is directly

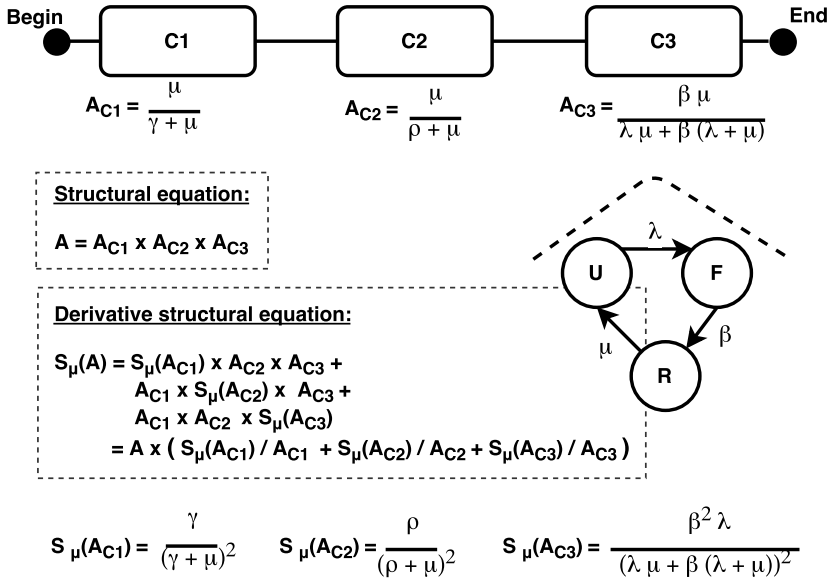


Figure 4.4: Sensitivity computation with RBD as main model and CTMC sub-models

used to build the unified sensitivity ranking, and therefore identify the availability or reliability bottlenecks of the system under analysis.

Equation (2.1) presented the structural function for a series RBD. The corresponding derivative function with respect to a general parameter θ is expressed in Equation (4.1), which yields $S_\theta(A_s)$, the sensitivity of the availability with respect to θ:

$$S_\theta(A_s) = \frac{\partial A_s}{\partial \theta} = \frac{\partial}{\partial \theta}\left[\prod_{i=1}^{n} A_i\right] = \sum_{i=1}^{n}\left(\frac{\partial}{\partial \theta} A_i \prod_{j \neq i} A_j\right) = \left(\prod_{i=1}^{n} A_i\right)\left(\sum_{i=1}^{n} \frac{S_\theta(A_i)}{A_i}\right) = A_s \sum_{i=1}^{n} \frac{S_\theta(A_i)}{A_i}, \tag{4.1}$$

where A_s is the series RBD system availability, A_i is the availability of each block i from the n blocks that compose the series RBD, and $S_\theta(A_i)$ is the sensitivity index of each block's availability with respect to θ.

Similarly, the sensitivity of parallel structure in an RBD is expressed in Equation (4.2), that corresponds to the partial derivative of Equation (2.2).

$$S_\theta(A_p) = \frac{\partial A_p}{\partial \theta} = \frac{\partial}{\partial \theta}\left[1 - \prod_{i=1}^{n}(1 - A_i)\right] = \sum_{i=1}^{n}\left(\frac{\partial}{\partial \theta} A_i \prod_{j \neq i}(A_j - 1)\right), \tag{4.2}$$

Other RBD setups such as series-parallel, and parallel-series require using the chain rule and similar formulas to reach the corresponding derivative functions.

Figure 4.4 depicts the process of sensitivity ranking computation for a composition of RBDs and CTMCs, using partial derivatives. For each single RBD block (or RBD sub-model), a derivative structural equation is obtained. There is a closed-form equation that describes the availability from CTMC sub-model, so the corresponding derivative equation is computed and

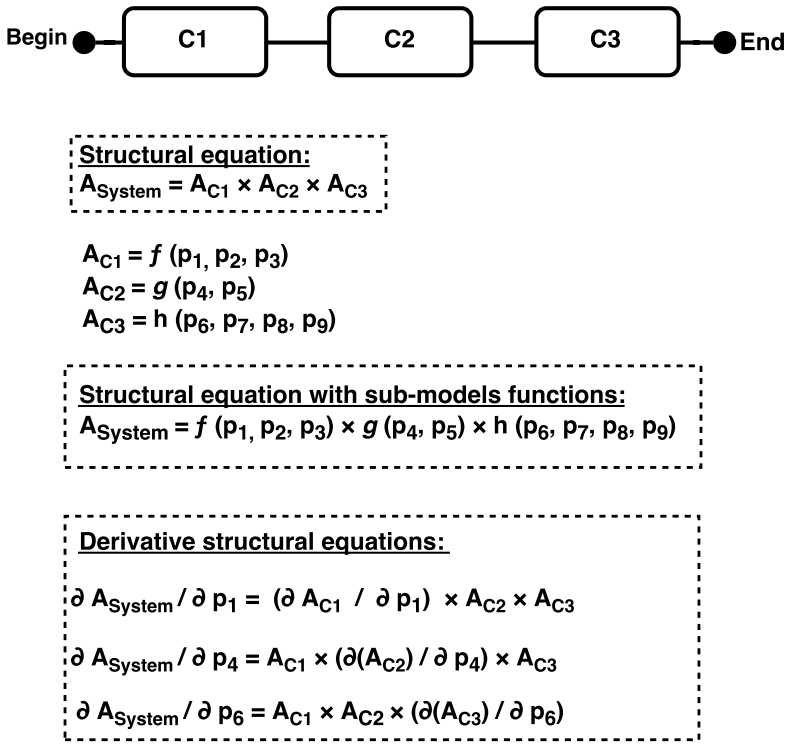


Figure 4.5: Sensitivity computation with RBD as main model and non-specified sub-models

used for obtaining the sensitivity indices (as depicted in Figure 4.4). In Figure 4.4, RBD blocks C1, C2, and C3 are arranged in series and share the parameter μ. Therefore, the computation of system availability sensitivity to μ employs the product rule of partial derivatives, already considered in Equation (4.1). The derivative equations for each block (A_{C1}, A_{C2} and A_{C3}) can be combined in the final derivative equation for the complete model. Whenever the values of parameters have significant changes, the same sensitivity function can be applied for fast computation of the ranking and subsequent identification of current bottlenecks.

If a closed-form equation cannot be reached for the sub-model, the methods for computing sensitivity indices of CTMC discussed in BLAKE; REIBMAN; TRIVEDI (1988) 2.3 must be applied. Such a situation would fit the second scenario for sensitivity analysis with RBD as top-level model, explained as follows.

Figure 4.5 shows a generalized view of sensitivity analysis comprising a series RBD as main model and sub-models from non-specified formalisms. Each block availability (A_{C1}, A_{C2} and A_{C3}) is computed through a sub-model represented as a function of a given set of parameters. If the sub-models do not have any parameter in common, the partial derivative with respect to one specific sub-model parameter does not require derivatives of other sub-models results, as depicted in Figure 4.5. If two or more sub-models depend on the same parameter, such a relation must be expressed in the functions for the respective blocks of the RBD. The derivative of the

top-level RBD structural equation will address the need for specific differentiation rules.

It is important to highlight that each partial derivative ($\frac{\partial A_{C1}}{\partial p_1}$, $\frac{\partial A_{C2}}{\partial p_4}$ and $\frac{\partial A_{C3}}{\partial p_6}$) in Figure 4.5 will be replaced by the specific kind of sensitivity index employed for the corresponding sub-model. This makes the sensitivity analysis more flexibe for situations when closed-form equations or partial derivative methods are not possible for the sub-models.

The method proposed in this thesis for the case of RBD as top-level model, illustrated in Figure 4.5, is described as pseudo-code in Algorithm 1. It produces the unified sensitivity ranking considering an RBD as top-level model and generic sub-models.

```
   Data: MainModel,SubModels
   Result: Unified sensitivity ranking R
1  R ← ∅;
2  MainStructFunction ← Get_Structural_Function(MainModel);
3  foreach Model ∈ SubModels do
4  |  Parameters ← Get_List_of_Parameters(Model);
5  |  SubModelFunction ← Build_Function(Model,Parameters);
6  |  MainStructFunction ← Transform_Structural_Function(MainStructFunction,
   |  Model, SubModelFunction);
7  |  ParametersSet ← ParametersSet ∪ Parameters;
8  |  SensitivitySubModels[Model] ← Compute_Sensitivity_Ranking(Model);
9  end
10 foreach Parameter ∈ ParameterSet do
11 |  DerivativeFunction ← Compute_Derivative(MainStructFunction, Parameter);
12 |  Sensitivity[Parameter] ←
   |  Convert_Symbolic_to_Numeric(DerivativeFunction,SensitivitySubModels);
13 |  R ← R ∪ Sensitivity[Parameter];
14 end
15 R ← Sort(R);
16 Return R;
```

Algorithm 1: Algorithm for sensitivity analysis of hierarchical model with RBD as top-level model

The main steps of the algorithm are explained in natural language as follows:

- Obtain the structural function of the main model (line 2);
- Identify the parameters of main model and each sub-model (line 4);
- Replace structural function components that represent sub-models by functions of the sub-models parameters (lines 5 and 6);
- Replace structural function components that represent common RBD nodes by their availability functions (lines 5 and 6);
- For each sub-model, compute sensitivity indices with respect to its own parameters (line 8);

- For each parameter, compute symbolic partial derivatives of the system structural function with respect to the parameter (line 11);
- For each symbolic derivative expression, convert it into numerical sensitivity index using parameters values and sub-models sensitivity indices (line 12);
- Sort sensitivity indices to obtain the unified sensitivity ranking (line 15).

The unified sensitivity ranking enables assessing the impact of every parameter from the main model and its sub-models on the metric of interest (e.g., system steady-state availability). It is worth explaining that the function *Build_Function*, on line 5 has two possible return values. It returns a generic function for non-RBD sub-models, that serves to indicate which parameters that sub-model depends on (e.g, $f(p_1, p_2, p_3)$). In case of simple RBD blocks or RBD sub-models, the corresponding availability function will be returned. The procedure *Compute_Sensitivity_Ranking*, on line 8, may have a specific implementation for each kind of sub-model, depending on the available sensitivity analysis techniques. Partial derivatives, percentage difference, and DoE analysis are the techniques tested in this thesis and validated throughout the case studies.

4.2.2 Composition of Sensitivity Indices with SPN as top-level model

When modeling through composition of SPN and CTMC models, sensitivity analysis might use partial derivatives for all models. If the SPN model has no constraint that hinders generation of the embedded Markov chain, one can apply the partial derivative methods presented in (MUPPALA; TRIVEDI, 1990) for SPN sensitivity analysis, in conjunction with equivalent methods for CTMCs. Figure 4.6 depicts this process, for an SPN with two places (P0 and P1) and two transitions (T0 and T1). The firing delay of transition T0 comes from the mean time to absorption (MTTA) of a CTMC sub-model. The sensitivity computation begins by obtaining the matrix for the embedded Markov chain of the SPN. The matrix elements (i.e., transition rates) are denoted as symbolic expression using the SPN parameters. We compute the partial derivatives of each expression with respect to every parameter. The sensitivity indices computed for the CTMC parameters are further used to convert the symbolic derivative matrix of the corresponding upper-level model (e.g., main model) in a numerical matrix. The main model sensitivity indices are finally obtained using the numerical derivative matrix and equations described in (BLAKE; REIBMAN; TRIVEDI, 1988)(MUPPALA; TRIVEDI, 1990).

Some systems might require non-exponential distributions for best fitting the transition delays in SPNs. The models with non-exponential delays are usually solved through simulation instead of numerical analysis (i.e., solution of systems of equations). A simulation approach does not allow computation of partial derivatives, therefore other sensitivity analysis methods must be used. For such cases, it might be required combining indices that were obtained through one method with indices computed with another method.

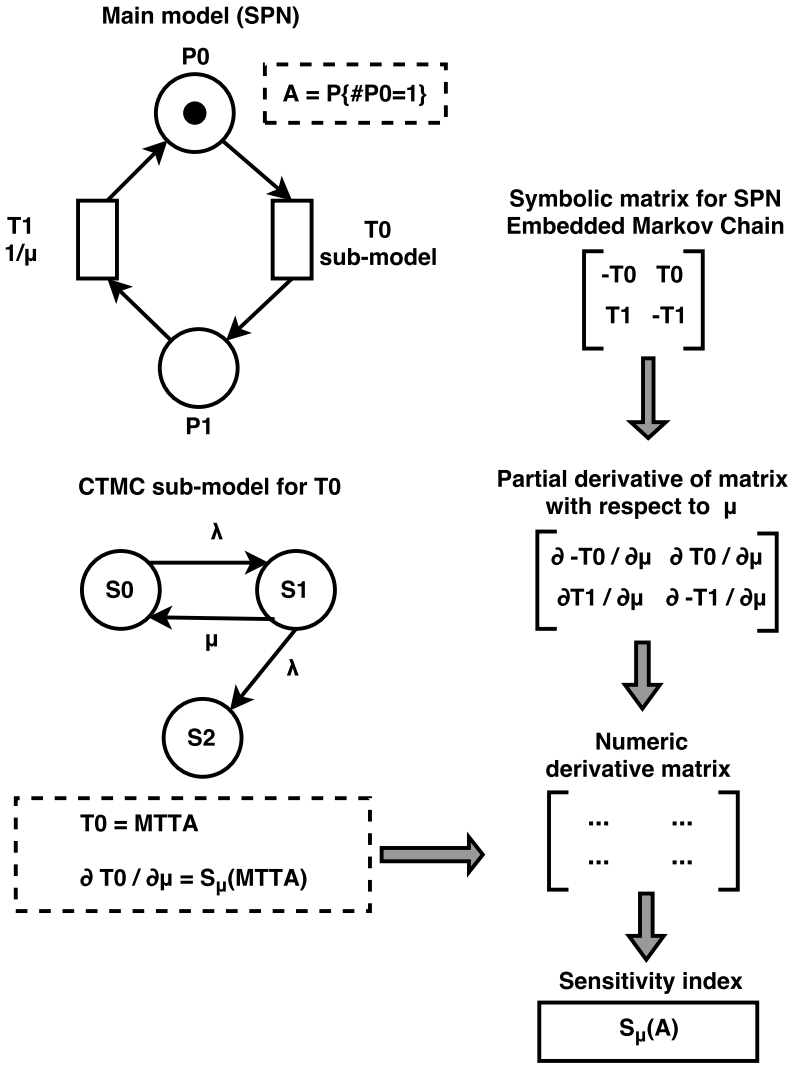


Figure 4.6: Sensitivity computation with SPN as main model and a CTMC sub-model

This thesis proposes a method for the sensitivity analysis of hierarchical models with SPN as top-level model, that is described in pseudo-code in Algorithm 2 . It has similarities, but also remarkable differences, in relation to that proposed for the case of RBD top-level model. The following steps comprise the algorithm for an SPN that might be solved by any of both methods: numerical analysis or simulation. This is accomplished by employing the sensitivity analysis technique known as percentage difference (see Section 2.3).

```
   Data: MainModel, SubModels, TransitionsWithSubModels, RangeOfParamValues
   Result: Unified sensitivity ranking R
 1 R ← ∅;
 2 MainModelParams ← Get_List_of_Parameters(MainModel);
 3 foreach Param ∈ MainModelParams do
 4 |   OutputValues ← Solve(MainModel, Param, RangeOfParamValues[Param]);
 5 |   Minimum ← min(OutputValues);
 6 |   Maximum ← max(OutputValues);
 7 |   Sensitivity[Param] ← (Maximum - Minimum) / Maximum;
 8 |   R ← R ∪ Sensitivity[Parameter];
 9 end
10 foreach Model ∈ SubModels do
11 |   SensitivitySubModels[Model] ← Compute_Sensitivity_Ranking(Model);
12 end
13 foreach Transition ∈ TransitionsWithSubModels do
14 |   SubModelParams ← Get_List_of_Parameters(Transition.SubModel);
15 |   foreach Param ∈ SubModelParams do
16 |   |   Sensitivity[Parameter] ← SensitivitySubModels[Transition.SubModel,
   |   |   Parameter];
17 |   |   Sensitivity[Parameter] ← Sensitivity[Parameter] × Sensitivity[Transition];
18 |   |   R ← R∪ Sensitivity[Parameter];
19 |   end
20 end
21 R ← Sort(R);
22 Return R;
```

Algorithm 2: Algorithm for sensitivity analysis of hierarchical model with SPN as top-level model

The main steps of the algorithm are explained as follows:

- Identify the parameters (including transitions) of the top-level model. (line 2)
- For each parameter, solve the model for the values in the respective range, and compute its percentage difference sensitivity index. (lines 3 to 9)
- For each sub-model, compute sensitivity indices with respect to its own parameters. (lines 10 to 12)
- For each transition which has a sub-model, do: (line 13)

- Identify the parameters of the respective sub-model. (line 14)
- For each parameter of the sub-model, multiply its sensitivity index by the sensitivity index of the corresponding transition in the top-level model. (lines 15 to 18)

- Sort sensitivity indices to obtain the unified sensitivity ranking. (line 21)

The approach proposed here computes the product of the sensitivity indices from the main model and the corresponding sensitivity indices from the sub-model. This way, the impact of a parameter for the sub-model metric is weighted by the impact that the respective transition has on the SPN metric. The result is an estimate of the impact of a sub-model parameter on the metric of interest in the main model.

The product of the sensitivity indices is also a way of following the chain rule of differential calculus, that may be written as in Equation (4.3):

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}, \tag{4.3}$$

where z is a function of the variable y, and y is a function of variable x. In a hierarchical model, z is the metric of interest in the main model, y is a parameter from the main model that is computed through a sub-model output metric, and x is an input parameter from the sub-model. Therefore, the derivative of the metric z with respect to x is achieved by multiplying the derivative of z with respect to y by the derivative of y with respect x.

It is worth to highlight that Algorithm 2 depends on the assumption that a given parameter is not used in two or more models. If such an assumption is not valid, the algorithm must be adapted to sum up the composite indices computed for each sub-model, as indicated by the chain rule for functions of two or more variables.

Despite a percentage difference is not exactly a partial derivative, the similar nature of both indices makes the proposed approach especially suitable. This research did not evaluate combinations using other types of indices, but it is reasonable considering that the Algorithm 2 can be adapted for other cases.

Figure 4.7 depicts an example of situation where the Algorithm 2 can be applied, comprising an SPN as main model and a CTMC as sub-model. The SPN model contains one deterministic transition (denoted by black thick rectangle). This model requires solution through simulation because an embedded Markov chain cannot be generated for analytical solution of the SPN. The delay of the other transition is exponential, and is computed by the CTMC sub-model. If ones wants to perform sensitivity analysis in the main model, a suitable approach is computing percentage difference indices. Such a technique does not depend on partial differentiation of equations.

Figure 4.7 shows that results of simulations for distinct parameter values enable computing $S_{T0}(A)$ and $S_{T1}(A)$: the sensitivity of the metric A to the delay of transitions T0 and T1,

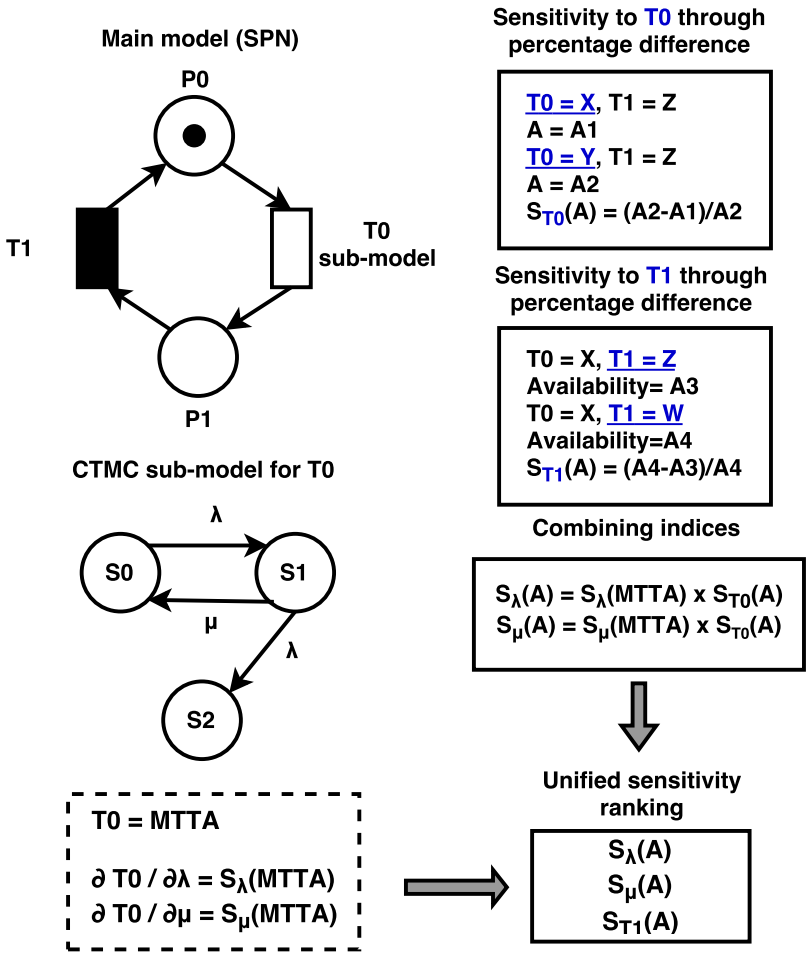


Figure 4.7: Sensitivity computation with SPN as main model (for simulation) and a CTMC sub-model

respectively. The sensitivity of metric A to the parameter λ ($S_\lambda(A)$) depends on combination of indices from CTMC sub-model $S_\lambda(MTTA)$ and SPN main model ($S_{T0}(A)$). Similarly, for computing $S_\mu(A)$, it is essential combining $S_\mu(MTTA)$ and $S_{T0}(A)$.

4.2.3 Implementation on Mercury Tool

The sensitivity analysis methods presented here provided the basis for the development of features in the Mercury tool (MERCURY, 2016)(SILVA et al., 2015). That software package helps to automatize the methodology described in Section 4.1. Especially the possibility of assembling models in a hierarchical manner, and the sensitivity analysis of single and hierarchical models were included in most recent versions.

Details about Mercury and the implementation of the proposed algorithms in that tool are presented in the Appendix A.

4.3 Optimization Guided by Sensitivity Ranking

For many systems, it is difficult to find a setup that maximizes (or minimizes) a desired metric while meeting a given constraint, such as financial costs or architectural limitations. Optimization techniques are usually employed for reaching an alternative that, at least, is close to best possible solution. The proposed methodology is also valuable in such cases. We employ the sensitivity ranking for iteratively performing parameter changes that will lead the system metrics to an optimal or nearly optimal solution.

We may handle many performance, dependability, and capacity planning problems as specific cases of the assignment problem. Consider that a model for a given system under study has a set N of parameters, here called as services, and a set M of possible providers to be assigned to each of those services. A provider means a possible value for each parameter in the model. The value assigned to each parameter may depend on the choice of an specific equipment manufacturer, a certain software configuration, or a third-party service quality. The optimization process must find the assignment of providers (i.e, parameter values) that minimizes (or maximizes) an objective measure θ, which might be reliability, availability, response time, or even a composition of many measures.

The optimization process must provide an assignment matrix $\mathbf{Z}$ which minimizes θ. This assignment matrix is the solution of the optimization problem. Each element of matrix $\mathbf{Z}$ is a binary variable z_{ij}, which is set to 1 if the provider j is assigned to service i, and otherwise is set to 0. As the row index i is related to the services and the column index j is related to the providers, the matrix $\mathbf{Z}$ has dimension $||N|| \times ||M||$. Since a provider j may not offer all services, a matrix $\mathbf{A}$ is used to indicate all the available services in a given provider. The elements a_{ij} of matrix $\mathbf{A}$ are binary variables which have a value of 1 if the provider j offers the service i, and a value of 0 otherwise. The QoS characteristics of the providers are represented in matrix

B, where each component b_{ij} is a 2-tuple (t,r) indicating the mean response time t and the reliability r of the provider j when offering service i.

Considering the above-mentioned objective function, variables and parameters, this optimization problem can be written as:

$$\min \quad \theta(\mathbf{Z}) \tag{4.4}$$

subject to

$$\sum_{j \in M} z_{ij} = 1 \ , \ \forall i \in N, \tag{4.5}$$

$$z_{ij} \leq a_{ij} \ , \ \forall i \in N \ , \ \forall j \in M. \tag{4.6}$$

The first constraint (Equation (4.5)) indicates that only one provider must be selected for each service, and all services must have one provider assigned to it. The second constraint (Equation (4.6)) requires that the provider j cannot be selected for service i if it is not able to perform that service, as indicated in matrix **A**.

We implement an instantiation of the GRASP metaheuristic (FEO; RESENDE, 1989) to solve the assignment problem we just described. GRASP is suitable to find approximate solutions for combinatorial optimization problems such as the service-provider assignment problem just described. The problems handled by GRASP are usually formulated as

$$\min f(x) \text{ subject to } x \in X, \tag{4.7}$$

where $f(\cdot)$ is an objective function to be minimized and X is a discrete set of feasible solutions. Feo and Resende (1989) proposed GRASP as a probabilistic heuristic for the set covering problem. Further developments have been made on GRASP (FEO; RESENDE, 1995) (RESENDE; RIBEIRO, 2003), and it has been applied in many distinct areas (FESTA; RESENDE, 2002) (MATEUS; SILVA; RESENDE, 2011).

GRASP is a multi-start heuristic (FEO; RESENDE, 1989) where a greedy randomized solution is constructed at each iteration to be used as a starting solution for local search. The combination of greediness and randomization is a key feature of GRASP that shall be adjusted according to the problem characteristics. Components of the solution are ranked using a greedy criterion, and the best ones are selected for a restricted candidate list (RCL), from where a random element is chosen. Local search repeatedly substitutes the current solution with a better one in its neighborhood. Such a replacement is called a move. If there is no improvement in the neighborhood, the current solution is declared as a local minimum and the search stops. The best local minimum found over all GRASP iterations is the output of the procedure.

Algorithm 3 shows the pseudo-code of our GRASP-based procedure, which returns a solution vector Z^* that is the best among the solution vectors found throughout many iterations. In this thesis, we adapt the algorithm presented in (MATOS; MACIEL; SILVA, 2013)

```
   Data: N, M, MaxPool, MaxIt, Greediness
   Result: Solution Z*
1  C* ← ∞;
2  while Stopping criteria is not satisfied do
       /* The method below is the adaptation made to original GRASP */
3      Z ← Sensitive_Construction(N,M,Greediness);
4      Z* ← Approximate_Local_Search(Z,MaxPool,MaxIt);
5      if cost(Z*) < C* then
6          Z* ← Z;
7          C* ← cost(Z*);
8      end
9  end
10 Return Z*;
```

Algorithm 3: Algorithm for Sensitive GRASP

by redefining the construction of initial solutions to incorporate the sensitivity analysis (method **Sensitivity_Construction** on line 3, it was only **Construction** originally). We name the overall algorithm as Sensitive GRASP. The sensitive construction procedure, which is our contribution, is described in Algorithm 4. This procedure starts with a random selection of providers for each service (lines 1–3), enabling the use of very different starting points in each iteration. Further, the sensitivity analysis is used for the identification of most impacting service in the current solution (line 5). According to the *Greediness* parameter, we select the best g providers for the most impact service, composing the RCL from where a random provider k is chosen (lines 6–8). This provider k replaces the previous element in the solution Z returned to main algorithm. The steps of sensitive construction procedure are also presented in Figure 4.8. Notice that greediness is a value chosen between 0 and 1. If the greediness is zero, any suitable provider can be chosen, so the improvement is completely random. If the greediness is 1, then only the best provider can be chosen. In Figure 4.8, the greediness is 0.5, so out of eight providers, the four with smallest response times are put in the RCL, and a random provider chosen from there.

```
   Data: N, M, Greediness
   Result: Initial solution Z
1  for Each service i ∈ N do
2      Randomly select provider k ∈ M ;
3      Z_i ← k ;
4  end
5  t ← Get top service on sensitivity ranking for Z;
6  g ← Ceiling(Greediness × Number of providers for service T) ;
7  RCL ← Best g providers for service t;
8  Randomly select provider k ∈ RCL;
9  Z_t ← k ;
10 Return Z;
```

Algorithm 4: Algorithm for Sensitive Construction procedure

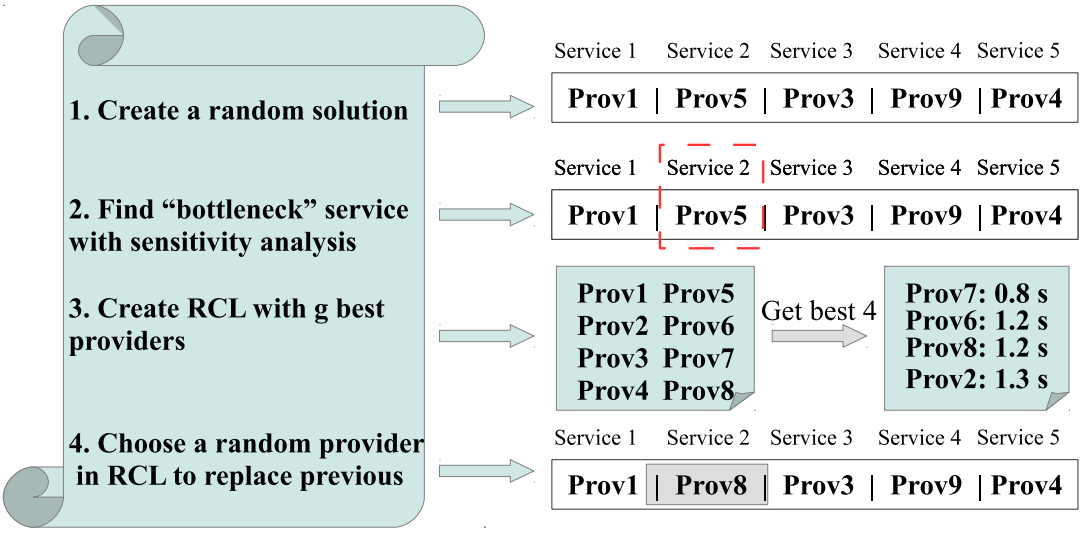


Figure 4.8: Example of sensitive construction of initial solution

The next step in GRASP is the approximate local search, presented in Algorithm 5. This procedure follows the same approach presented in (MATOS; MACIEL; SILVA, 2013). It builds a pool of good solutions from the neighborhood of the current solution. The procedure "**Move**" (line 4) is responsible for providing one neighbor of a given solution vector Z. In our assignment problem, the neighbor solutions correspond to all combinations that have only one different provider in relation to Z. Figure 4.9 illustrates an example of three neighbors that may be found for a given solution Z. Algorithm 5 searches throughout the neighborhood until it fulfills the pool of solutions or until it reaches the defined maximum number of iterations. A random solution from the pool is used as the pivot of a new neighborhood inspection. When no neighbor outperforms the pivot, the pool remains empty and the current approximate local search terminates.

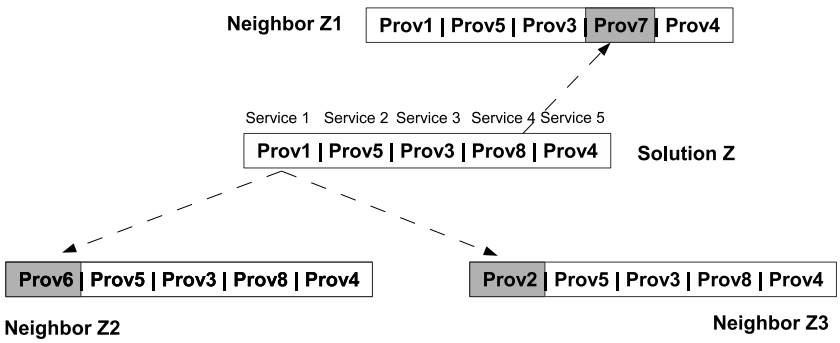


Figure 4.9: Example of neighborhood for a solution

The cost of each solution found in GRASP iterations (see **cost()** function in lines 5 and 7 of Algorithm 3 and line 5 of Algorithm 5) may be computed by means of analytical models, such as Fault Trees, RBDs, CTMCs or SPNs. The performance and dependability characteristics of each provider can be used to determine the values of transition rates in a CTMC, for example. A modeling tool, such as Mercury (MERCURY, 2016; SILVA et al.,

```
   Data: Z, MaxPool, MaxIt
   Result: Approximate local minimum Z
1  repeat
2      Count ← 0; Pool ← ∅;
3      repeat
4          Z′ ← Move(Z);
5          if cost(Z′) < cost(Z) then
6              Pool ← Pool ∪ Z′;
7          end
8          Count ← Count + 1;
9      until (|Pool| ≥ MaxPool) or (count ≥ MaxIt);
10     if Pool ≠ ∅ then then
11         Z ← Randomly selected solution from Pool;
12     end
13 until Pool = ∅;
14 Return Z;
```

Algorithm 5: Algorithm for the approximate local search used in GRASP

2015) and SHARPE (TRIVEDI; SAHNER, 2009), may be used to compute metrics as well as the sensitivity indices from the analytical model.

The optimization code requires a specific module to interact with the analytical model solver, enabling the fully automated evaluation of large benchmarks. A benchmark, in this case, is composed of various possible values for each parameter. For each new possible solution (i.e., a selected configuration of parameters values), the analytical model input file is changed to match the current providers' parameters. The corresponding objective measure is then computed and used for comparisons in the optimization process. The sensitivity analysis during the construction procedure also uses the module for interacting with the model solver.

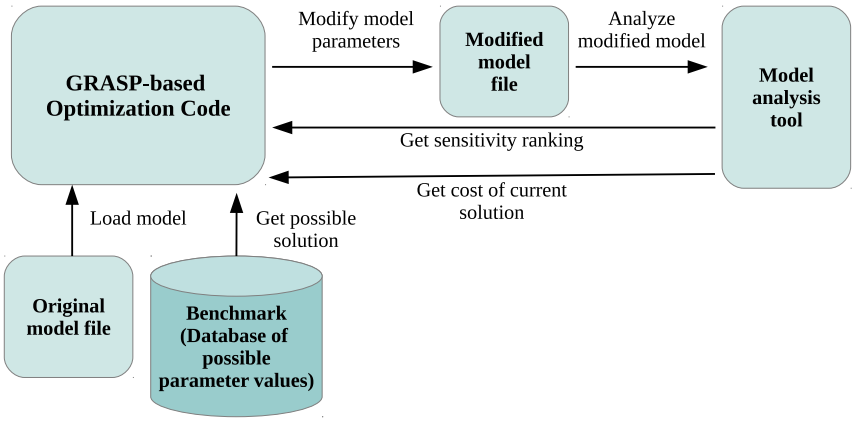


Figure 4.10: Computation of cost for a possible solution using a model analysis tool

The feasibility and accuracy of this optimization framework is verified in a case study presented in Section 5.4.

4.4 Concluding Remarks

The methods and algorithms proposed in this chapter aim at aiding the identification of points for performance or dependability improvement in cloud computing systems. The automation of those methods in a modeling tool makes them more accessible for researchers and systems administrators. The Sensitive GRASP algorithm enables a step forward, by using the bottlenecks to guide an optimization process for selected system metrics. It is worth mentioning that the methods presented here were successfully applied in various case studies, demonstrated in Chapter 5.

5

Case studies

This chapter presents four case studies that demonstrate the proposed methods in the analysis of distinct cloud computing scenarios: (i) the availability evaluation of private clouds with redundant components, where only typical IaaS components are involved and the method of Section 4.2.1 is applied; (ii) the availability evaluation of mobile clouds, which includes the evaluation of wireless communication issues, battery lifetime and mobile application availability; this scenario also serves to compare the results of our methods with different kinds of sensitivity indices; (iii) the performance evaluation of a composite web service powered by autoscaling mechanisms in a private cloud, which integrates the analysis of user application and cloud infrastructure components and applies the method of Section 4.2.2; and (iv) the optimization of performance and reliability of a composite web service by means of the Sensitive GRASP algorithm presented in Section 4.3. The results achieved in those case studies provide evidence on the usefulness and efficacy of this approach.

5.1 Availability of Redundant Private Clouds

This case study aims at demonstrating the proposed methodology for the availability evaluation of redundant private cloud architectures. The composition of sensitivity indices described in Section 4.2.1 is especially useful here.

Hierarchical analytical models were created to describe the behavior of private cloud architectures structured according to the general stack of components for open-source cloud platforms described in Section 2.1.3. The proposed models aid in analyzing how the system availability can be improved, by means of differential sensitivity analysis. Figure 5.1 shows an architecture with three clusters and redundancy on some private cloud components. The redundant components were chosen according to the study presented in (DANTAS et al., 2012a). A front-end computer is adopted as the “Cloud Manager Subsystem” and configured with the components known as Cloud Controller (CLC) and Image Repository Controller (IRC). A warm-standby host is capable of keeping the Cloud Manager subsystem working if the primary host fails. Each cluster has one machine that is hereafter called the “Cluster Subsystem”, which

runs the Cluster Controller (CC) and Storage Controller (SC) components. Warm-standby redundancy is also considered for each "Cloud Manager Subsystem". Each cluster also has three machines that run the component known as Node Controller (NC). The set of three nodes in each cluster is hereafter called a "Nodes Subsystem". The impact of implementing the redundancy in the Cloud Manager Subsystem (CLC and IRC) and in the Cluster Subsystem (CC and SC) is considered for this system.

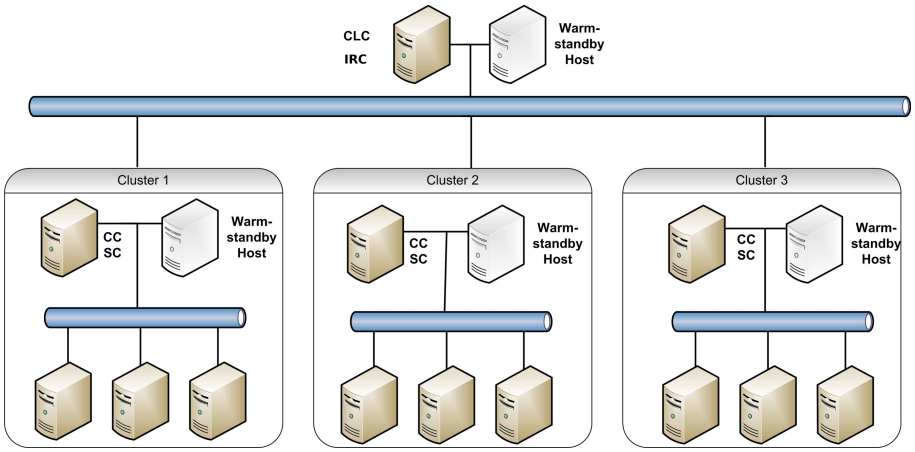


Figure 5.1: Private cloud architecture with redundant components

Due to their simplicity and efficiency of computation, Reliability Block Diagrams (RBDs) may be used to analyze the availability of the private cloud architecture described in Figure 5.1, from a high-level standpoint. However, the redundancy mechanisms used in the Cloud Manager and Cluster Subsystems require the use of state-based models, such as Markov chains, for a proper representation. Therefore, a hierarchical heterogeneous modeling approach is adopted, composing an RBD and Markov Reward Models (MRMs) (TRIVEDI, 2001) (SAHNER; TRIVEDI; PULIAFITO, 1996). The RBD describes the high-level availability view of subsystems and non-redundant components, whereas the MRM represents the detailed behavior of subsystems which employ an active redundancy mechanism. This modeling approach enables us to obtain closed-form equations for the availability of the studied architecture.

5.1.1 Creating top-level model

RBD models were created to evaluate the cloud infrastructures considering one, two, and three clusters. These RBD models are depicted in Figure 5.2, Figure 5.3, and Figure 5.4, respectively. For all cases, the system infrastructure is available if the Cloud Subsystem is running and at least one Cluster Subsystem is available with one or more nodes running in that cluster. The block named CLC represents the Cloud Manager Subsystem. Blocks CC_j represent each of the Cluster Subsystems, where $j \in 1,..,N$, and N is the number of clusters. Each block NCi_CCj represent the i^{th} node that integrates the j^{th} cluster.

The RBD models enable using closed-form expressions for computing the system steady-state availability. This step might ease obtaining sensitivity indices with respect to each model

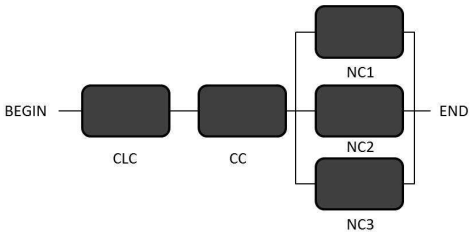


Figure 5.2: RBD model of the Cloud system with one cluster

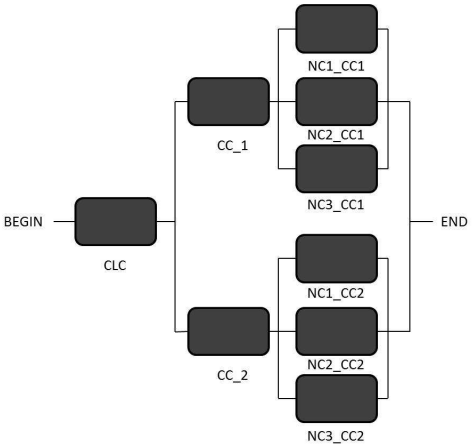


Figure 5.3: RBD model of the Cloud system with two clusters

parameter. When closed-form equations cannot be obtained, sensitivity indices may be computed numerically from the RBD and CTMC models. The numerical solution of Markov chains may depend on iterative algorithms which are prone to issues such as stiffness or long time for convergence of results. The convergence problem does not exist in the solution of closed-form equations, and stiffness is also a smaller problem, although some issues might occur yet due to computer representation of numbers.

Equation (5.1) denotes the closed-form expression for the availability of the whole cloud system (A_{Sys}). Despite the architectures evaluated here have a fixed number of clusters (up to three) and nodes per cluster (three), the Equation (5.1) applies for the general case of a cloud system with k clusters, having n nodes in each cluster. Each component of the Equation (5.1) comes from the distinct models presented in Section 5.1.2.

$$A_{Sys} = (A_{CLC}) \times (1 - \prod_{j=1}^{k} (1 - (A_{CC\_j}) \times (1 - \prod_{i=1}^{n} (1 - A_{Node\_i})))) \quad (5.1)$$

5.1.2 Creating sub-models for specific components

Figure 5.5 shows the RBD model that represents one node in each Nodes Subsystem. Each node is composed of hardware, operating system, hypervisor (e.g., KVM), and a Node Controller. The node is working only if all these components are active (non-failed).

The availability of each node, A_{Node_i}, can be computed from the Equation (5.2).

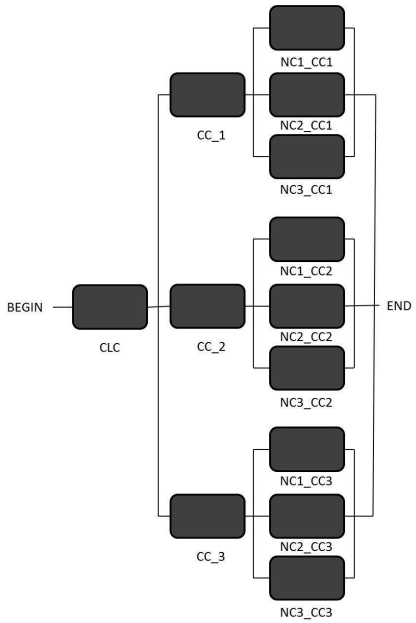


Figure 5.4: RBD model of the Cloud system with three clusters

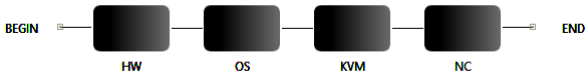


Figure 5.5: RBD model of one node

$$A_{Node\_i} = \frac{\mu_{Node}}{\lambda_{Node} + \mu_{Node}} \tag{5.2}$$

The blocks denoting the Cloud Manager Subsystem and Cluster Subsystems have their individual steady-state availability values computed through the MRM shown in Figure 5.6. The MRM has 5 states: UW, UF, FF, FU, and FW, and considers one primary and one spare server, respectively. The state UW represents primary server (S1) is functional and secondary server (S2) in standby. When S1 fails, the system goes to state FW, where the secondary server has not yet detected the S1 failure. FU represents the state where S2 leaves the waiting condition and assumes the active role, whereas S1 is failed. If S2 fails before taking the active role, or before the repair of S1, the system goes to the state FF. This study analyzes a setup where the primary server repair has priority over the secondary server repair. Therefore, when both servers have failed (state FF) there is only one possible outgoing transition: from FF to UF. If S2 fails when S1 is up, the system goes to state UF, and returns to state UW when the S2 repair is accomplished. Otherwise, if S1 also fails, the system transitions to the state FF. The failure rates of S1 and S2 are denoted by λ_s1 and λ_s2, respectively. The rate λ_i_s2 denotes the failure rate of the secondary server when it is inactive. The repair rate assigned to S2 is μ_s2. The rate sa_s2 represents the switchover rate, i.e., the reciprocal of the mean time to activate the secondary server after a failure of S1. Table 5.3 presents the values for all the mentioned parameters of the MRM. The value of μ_s1 is equal to the value of μ_s2, the rates λ_s1 and

λ_s2 also have equal values. These λ and μ values were obtained from single RBD models for each server of the Cloud Manager and Cluster Subsystems. The failure rate of the secondary server, when inactive, is assumed to be 20% smaller than the failure rate of an active server, since there is no stressing load over the inactive spare server. The value of *sa_s2* comes from default monitoring interval and activation times found in software such as Heartbeat (HEARTBEAT, 2012).

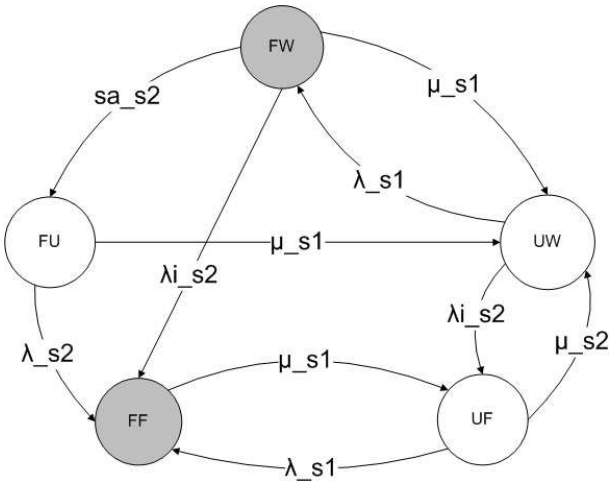


Figure 5.6: Markov chain model for a redundant subsystem with two hosts

The state reward rate $\rho(s)$ assigned to UW, UF and FU is equal to 1, since the subsystem is available in these states. The state reward rate assigned to FF and FW (shaded states) is equal to 0, since the subsystem is down in these states. The reward rate $\rho(s)$ is also defined through the function depicted in the Expression (5.3). There are no impulse rewards in this model. Therefore, the steady-state availability of the subsystem can be computed as the steady-state reward rate of the MRM, so $A = \sum_{s \in S} \pi_s \cdot \rho(s)$, where π_s is the steady-state probability of being in the state s, and $\rho(s)$ is the reward rate assigned to the state s.

$$\rho(s) = \begin{cases} 0 & \text{if } s \in \{FF, FW\}, \\ 1 & \text{otherwise.} \end{cases} \tag{5.3}$$

Each single host that composes the Cloud Manager Subsystem may be represented by a series RBD as shown in Figure 5.7. The host of Cloud Manager Subsystem consists of hardware, operating system, and the following software components: CLC (Cloud Controller) and IRC (Image Repository Controller). A similar RBD model may be used to represent a non-redundant Cluster Subsystem, which is composed of hardware, operating system, Cluster Controller (CC) and Storage Controller (SC), as depicted in Figure 5.8. As this study considers redundancy in both, Cloud Manager and Cluster Subsystems, these RBDs are adopted to obtain the equivalent MTTF and equivalent MTTR values which will be parameters in the Markov Reward Model for the correspondent redundant subsystem.

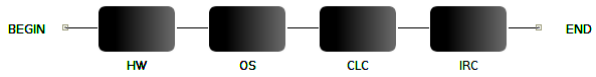


Figure 5.7: RBD model of a non-redundant Cloud Manager Subsystem

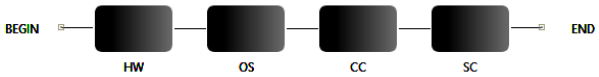


Figure 5.8: RBD model of a non-redundant Cluster Subsystem

The model presented in Figure 5.6 enables obtaining a closed-form equation for the availability of the redundant Cloud Manager Subsystem (see Equation (5.4)) and a similar equation for the Cluster Subsystem (see Equation (5.5)). Both equations, (5.4) and (5.5), can be used in conjunction with Equation (5.1), previously presented, to compute the overall availability, as well as to derive sensitivity measures of the redundant cloud system.

$$A_{CLC} = \frac{\mu_{CLC} \times (\lambda_{CLC} \times \lambda_{CLC_i} + \alpha_1^2 + sa \times \alpha_2)}{sa \times (\lambda_{CLC}^2 + \lambda_{CLC} \times \alpha_1 + \mu_{CLC} \times \alpha_1) + \alpha_3 \times (\lambda_{CLC} \times \lambda_{CLC_i} + \alpha_1^2)}, \tag{5.4}$$

where

$$\begin{aligned} \alpha_1 &= \lambda_{CLC_i} + \mu_{CLC}, \\ \alpha_2 &= \lambda_{CLC} + \lambda_{CLC_i} + \mu_{CLC}, \text{ and} \\ \alpha_3 &= \lambda_{CLC} + \mu_{CLC}. \end{aligned}$$

$$A_{CC} = \frac{\mu_{CC} \times (\lambda_{CC} \times \lambda_{CC_i} + \beta_1^2 + sa \times \beta_2)}{sa\,(\lambda_{CC}^2 + \lambda_{CC} \times \beta_1 + \mu_{CC} \times \beta_1) + \beta_3 \times (\lambda_{CC} \times \lambda_{CC_i} + \beta_1^2)} \tag{5.5}$$

where

$$\begin{aligned} \beta_1 &= \lambda_{CC_i} + \mu_{CC}, \\ \beta_2 &= \lambda_{CC} + \lambda_{CC_i} + \mu_{CC}, \text{ and} \\ \beta_3 &= \lambda_{CC} + \mu_{CC}. \end{aligned}$$

5.1.3 Definition of input parameters

Table 5.1 presents the parameter values for the hardware, operating system, KVM, and Node Controller blocks of the RBD. These values were obtained from published studies (KIM; MACHIDA; TRIVEDI, 2009a) (HU et al., 2010) about availability of virtualized systems. Par-

ticularly, the Node Controller values are based on studies about the availability of web services, since most components of open-source cloud platforms are built as web services. The values shown in Table 5.1 were used in the RBD model, and its analysis provided an equivalent MTTF (mean time to failure) (WANG; TRIVEDI, 2005) of 481.82 hours and an equivalent MTTR (mean time to repair) (WANG; TRIVEDI, 2005) of 0.91 hours for each node.

Table 5.1: Input Parameters for the nodes

Component	MTTF	MTTR
HW	8760 h	100 min
OS	2893 h	15 min
KVM	2990 h	1 h
NC	788.4 h	1 h

Table 5.2 presents the MTTF and MTTR used for the Cloud Manager Subsystem and Cluster Subsystem models. This study considers that the hardware and operating system of these subsystems are equivalent to those ones adopted in the nodes, so the MTTF and MTTR values are the same. The MTTF and MTTR of software components (CLC, IRC, CC, and SC) are based on values found in (HU et al., 2010), since these components are usually built as web services on platforms such as Eucalyptus and OpenStack. The results obtained from the RBD models are an equivalent MTTF of 333.71 h and an equivalent MTTR of 0.93 h for both Cloud Manager and Cluster Subsystems.

Table 5.2: Input Parameters for the Cloud Manager and Cluster Subsystems

Component	MTTF	MTTR
HW	8760 h	100 min
SO	2893 h	15 min
CLC, IRC, CC and SC	788.4 h	1 h

Table 5.3: Parameter values for the Markov chain model

Parameter	Description	Value (h)
$1/\lambda\_s1 = 1/\lambda\_s2 = 1/\lambda_{CC} = 1/\lambda_{CLC}$	Mean time for host failure	333.71
$1/\lambda_i\_s2 = 1/\lambda_{CLC_i} = 1/\lambda_{CC_i}$	Mean time for inactive host failure	400.45
$1/\mu\_s1 = 1/\mu\_s2 = 1/\mu_{CLC} = 1/\mu_{CC}$	Mean time for host repair	0.93
$1/sa\_s2 = 1/sa$	Mean time for spare host activation	0.005

5.1.4 Solution of hierarchical model

We computed availability measures using the mentioned input parameters on the hierarchical model. Both, MRMs and RBDs were solved numerically, first obtaining the availability for each sub-model: Cloud Manager and Cluster subsystems MRMs, and Nodes subsystem

RBD. So the corresponding availability values of each subsystem were used in the high-level RBD model.

Table 5.4 presents the availability measures of the cloud system considering the architectures with one, two, and three clusters, which hereinafter are called A1, A2, and A3. Besides the steady-state availability, Table 5.4 shows the number of nines (MARWAH et al., 2010), which constitutes a logarithmic view of the availability, and the downtime, which better denotes the impact of service unavailability from the user's standing point.

These results reveal that the architectures A2 and A3 decrease system downtime in about 50% when associating to A1. When comparing A2 and A3, some small availability differences are traceable. Nevertheless, the similarity indicates that increasing the number of clusters beyond three will have negligible impact on the availability, and it is an action which would be justified only by a need to increase capacity and performance. Therefore, a parametric sensitivity analysis is required to identify the availability bottlenecks in these architectures, guiding further improvements to the system availability.

Table 5.4: Availability and downtime measures of the cloud system

	Architectures		
Measure	**A1**	**A2**	**A3**
Steady-state Availab.	0.999938749	0.999969376	0.999969377
Number of 9's	4.21288	4.51394	4.51395
Annual downtime	32.194 min	16.096 min	16.095 min

5.1.5 Sensitivity analysis on sub-models and high-level models

Considering the specific case when the cloud system is composed of 3 identical clusters, and each cluster has 3 identical nodes, i.e., architecture A3, Equation (5.1) is rewritten shortly as Equation (5.6). Using this representation, the sensitivity of the system availability with respect to each parameter θ_i in the system (i.e., failure and repair rates) can be computed as described in Equation (5.7). Otherwise, considering a system with any number k of clusters, and n nodes per cluster, the sensitivity is computed through Equation (5.8). For the sake of conciseness, the equations that describe the sensitivity of the subsystems ($\frac{\partial A_{CLC}}{\partial \theta_i}$, $\frac{\partial A_{CC}}{\partial \theta_i}$, and $\frac{\partial A_{Node}}{\partial \theta_i}$) are not described here, but the reader can see them in the Appendix B.

$$A_{Sys} = A_{CLC} \times (1 - (1 - A_{CC} \times (1 - (1 - A_{Node})^3))^3) \tag{5.6}$$

$$
\begin{aligned}
S_{\theta_i}(A_{Sys}) &= \frac{\partial A_{Sys}}{\partial \theta_i} \\
&= \frac{\partial A_{CLC}}{\partial \theta_i} \times (1-(1-A_{CC}\times(1-(1-A_{Node})^3))^3) \\
&+ A_{CLC} \times \frac{\partial (1-(1-A_{CC}\times(1-(1-A_{Node})^3))^3)}{\partial \theta_i} \\
&= \frac{\partial A_{CLC}}{\partial \theta_i} \times (1-(1-A_{CC}\times(1-(1-A_{Node})^3))^3) \\
&+ A_{CLC} \times (-3\times(A_{CC}\times A_{Node}\times(A_{Node}^2-3\times A_{Node}+3)-1)^2\times \\
&(-A_{Node}\times(A_{Node}^2-3\times A_{Node}+3)\times\frac{\partial A_{CC}}{\partial \theta_i}-3\times A_{CC}\times(A_{Node}-1)^2\times\frac{\partial A_{Node}}{\partial \theta_i}))
\end{aligned}
\tag{5.7}
$$

$$
\begin{aligned}
S_{\theta_i}(A_{Sys}) &= \frac{\partial A_{Sys}}{\partial \theta_i} \\
&= \frac{\partial A_{CLC}}{\partial \theta_i} \times (1-\prod_{j=1}^{k}(1-(A_{CC\_j})\times(1-\prod_{i=1}^{n}(1-A_{Node\_i})))) \\
&+ A_{CLC} \times \frac{\partial (1-\prod_{j=1}^{k}(1-(A_{CC\_j})\times(1-\prod_{i=1}^{n}(1-A_{Node\_i}))))}{\partial \theta_i}
\end{aligned}
\tag{5.8}
$$

Aiming to reduce the influence of different magnitude orders in the analysis, the scaled sensitivity index $SS_k(A_{Sys})$ has been used, and may be computed as indicated in Equation (2.6). This scaled sensitivity index is necessary because whereas some events are defined in terms of thousands hours (e.g., MTTF of one host), other ones are in the range of minutes (e.g., MTTR of one host), or even seconds. Therefore, we scale the sensitivity index so that its magnitude order may be considered as the expected magnitude order of the change in the steady-availability. For instance, if a sensitivity index is in the order of 10^{-5}, this implies that percent changes in that respective parameter will cause variations in the fifth decimal place on the system availability.

Table 5.5 shows the sensitivity rankings for architectures A1, A2, and A3. It is worth mentioning that positive sensitivity values indicate a direct relationship and negative sensitivity values indicate that parameters have an inverse impact on the availability. Notice from Table 5.5 that the all sensitivity indices related to failure rates (i.e., λ_{CLC}, λ_{CLC_i}, λ_{CC}, λ_{CC_i}, λ_{Node}) are negative, so the growth of failure rates cause a decrease in the availability and vice versa. Sensitivity indices for the repair rates are negative, so the system availability increases when a repair rate is increased.

The scaled sensitivity indices, $SS_\theta(A_{Sys})$ are listed in decreasing order of importance, for each architecture. The order of importance vary depending on the architecture, so each one has

Table 5.5: Sensitivity rankings for architectures A1, A2, and A3

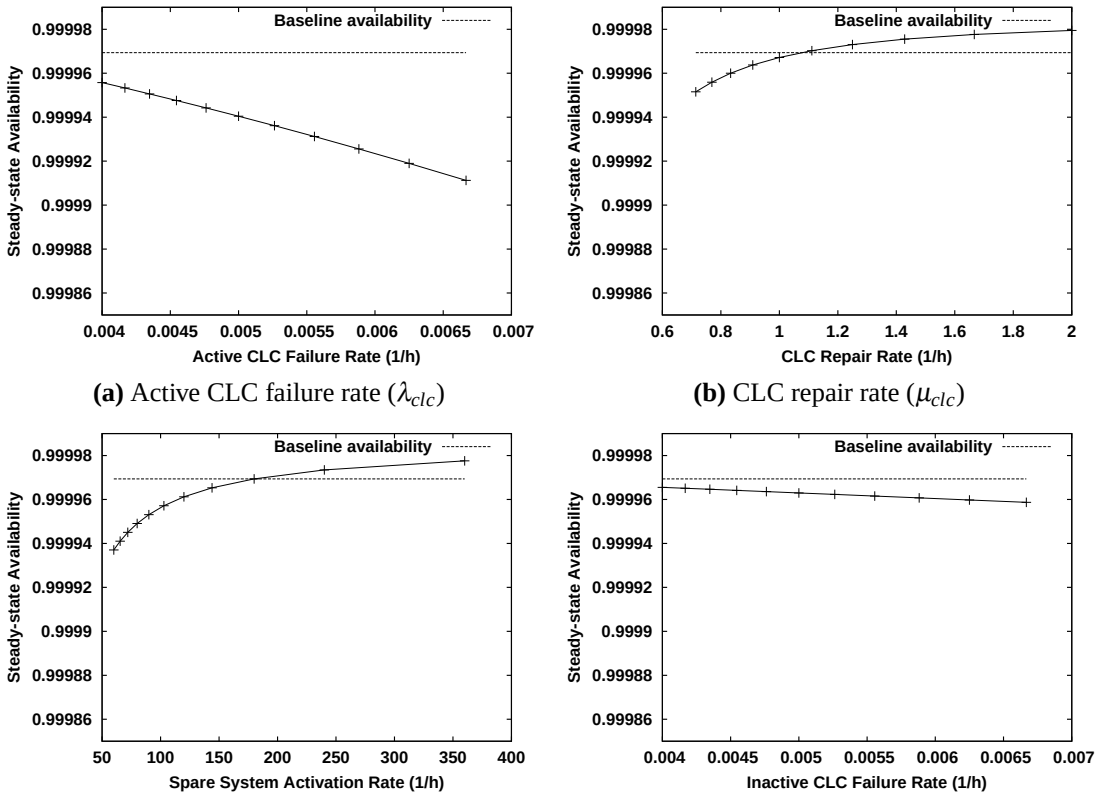
A1		A2		A3	
Param.	$\mathbf{SS_{\theta_i}(A_{Sys})}$	**Param.**	$\mathbf{SS_{\theta_i}(A_{Sys})}$	**Param.**	$\mathbf{SS_{\theta_i}(A_{Sys})}$
λ_{CLC}	-3.82×10^{-5}	λ_{CLC}	-3.82×10^{-5}	λ_{CLC}	-3.82×10^{-5}
λ_{CC}	-3.82×10^{-5}	μ_{CLC}	2.83×10^{-5}	μ_{CLC}	2.83×10^{-5}
sa	3.27×10^{-5}	sa	1.64×10^{-5}	sa	1.63×10^{-5}
μ_{CLC}	2.83×10^{-5}	λ_{CLC_i}	-6.41×10^{-6}	λ_{CLC_i}	-6.41×10^{-6}
μ_{CC}	2.82×10^{-5}	λ_{CC}	-2.34×10^{-9}	λ_{CC}	-1.08×10^{-13}
λ_{CLC_i}	-6.41×10^{-6}	μ_{CC}	1.73×10^{-9}	μ_{CC}	7.96×10^{-14}
λ_{CC_i}	-6.41×10^{-6}	λ_{CC_i}	-3.92×10^{-10}	λ_{CC_i}	-1.80×10^{-14}
λ_{Node}	-2.01×10^{-8}	λ_{Node}	-1.23×10^{-12}	λ_{Node}	-5.66×10^{-17}
μ_{Node}	2.01×10^{-8}	μ_{Node}	1.23×10^{-12}	μ_{Node}	5.66×10^{-17}

its distinct points of improvement. It is also worth noticing that most of the scaled sensitivity indices in Table 5.5 reduce from A1 to A2, and then to A3 (except by the parameter λ_{CLC}), indicating that the larger is the number of clusters, the smaller is the impact of these parameters. All three rankings show that the parameter which deserves priority for system enhancement is the failure rate of the primary Cloud Controller server (λ_{CLC}). Therefore, upgrading the Cloud Manager Subsystem replication (e.g., increasing from two to three servers) or investing in more reliable hardware and software for this specific component would be the most effective approach to reach higher system availability. Preventive maintenance for avoiding the failures of some specific hardware components (e.g., disks, memory modules) is another action that might reduce the failure rate of the Cloud Controller server, and therefore improve the availability effectively. Specially, for architecture A1, the highest importance is shared with the failure rate of the primary Cluster Controller server (λ_{CC}). Thus, improvements on A1 should also consider improving the Cluster Controller server, as well as extending with another cluster (composed of other Cluster and Storage Controllers and their respective nodes), transforming A1 into A2.

The second most important parameter, for A2 and A3, is the repair rate of the Cloud Manager Subsystem. This endorses that the Cloud Manager server is the "availability bottleneck" for these architectures. So, enhancing the efficiency of repair process for the Cloud Manager Subsystem is an action to be considered with high priority.

Considering the analyzed private cloud architectures, Table 5.5 also highlights that possible changes in the availability of the nodes have small impact on the system availability. The existence of multiple nodes in the clusters reduces the impact of a single node failure as well as minimizes the need for optimizing the node repair activity. The sensitivity analysis of A2 and A3 is also useful to determine that changes in the failover rate of the redundant subsystems (sa) affect the system availability more than all the other parameters related to the Cluster Subsystem do (i.e., λ_{CC}, λ_{CC_i}, and μ_{CC}).

The differential sensitivity ranking enables the comparison of parameters' impact more accurately than with the visual inspection of plots, varying one parameter at a time. Figure 5.9



(a) Active CLC failure rate (λ_{clc})

(b) CLC repair rate (μ_{clc})

(c) Spare server activation rate (sa)

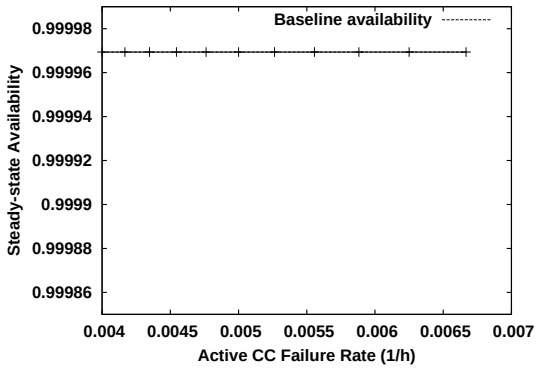
(d) Inactive CLC failure rate (λ_{clc_i})

Figure 5.9: Sensitivity analysis - Plots of most impacting parameters

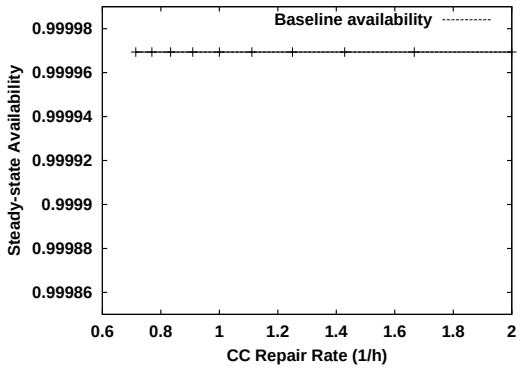
and Figure 5.10 present the results for the system availability when we vary the values of each parameter in the architecture A3. The value of the measure in the base case is drawn as the horizontal dashed line in each plot. Similar plots for A1 and A2 were not drawn for the sake of conciseness. The conclusions obtained from the plots confirm the results provided through the sensitivity indices. Figure 5.9a shows that the parameter λ_{CLC} has the highest influence on the measure of interest, since the slope of the line in this plot is higher than all the other ones. The next plots with high slopes are for μ_{CLC} (see Figure 5.9b) and sa (see Figure 5.9c), also confirming the analysis result obtained, since these parameters are the second and third ones, respectively, in the differential sensitivity ranking. Although the difference of impact between μ_{CLC} and sa is not so evident in the plots as it is in the numerical ranking.

Figure 5.9d shows a line with small slope, indicating therefore the little effect that changes in λ_{CLC_i} (fourth in the ranking of Table 5.5) produce on the system availability. For the least impacting parameters (i.e., λ_{CC}, μ_{CC}, λ_{CC_i}, λ_{Node}, and μ_{Node}), it is not possible to distinguish the levels of importance through the graphical representation. Such a dissimilarity would require using a different scale for each graph, adapted for each availability interval. This procedure may be too time-consuming and confuse the person who analyses the data, when evaluating scenarios with many components and subsystems. Therefore adopting differential sensitivity indices is recommended for such scenarios.

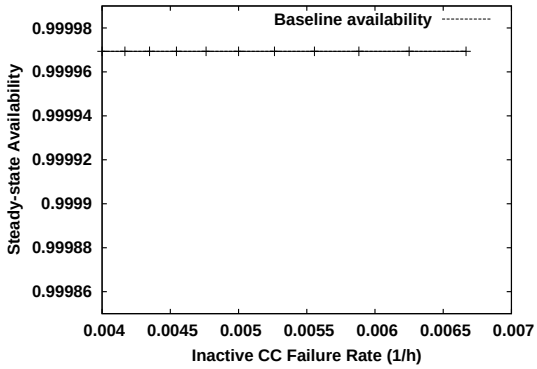
The variation of one parameter at a time allows noticing that the impact of CLC repair



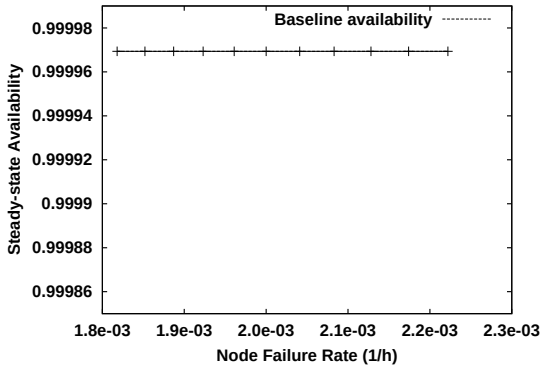
(a) Active CC failure rate (λ_{cc})



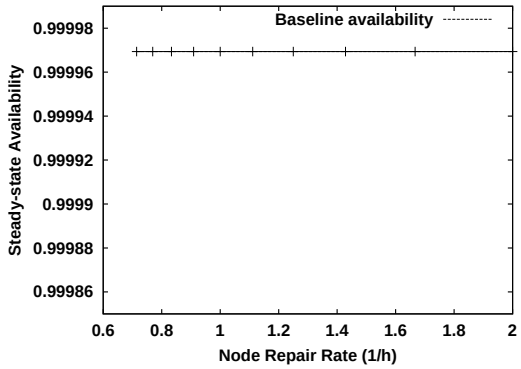
(b) CLC repair rate (μ_{cc})



(c) Inactive CC failure rate (λ_{cc_i})



(d) Node failure rate (λ_{node})



(e) Node repair rate (μ_{node})

Figure 5.10: Sensitivity analysis - Plots of least impacting parameters

rate (μ_{CLC}) and spare server activation rate (*sa*) decreases as they reach highest values. This indicates that after significant enhancements on the current repair and spare activation processes, only architectural changes and reduction of CLC failure rates would be effective to improve system availability. However, the latter actions imply in costs which may not be affordable for the cloud infrastructure owners, requiring careful analysis of other alternatives indicated by the sensitivity analysis and their trade-off between cost and effectiveness.

5.2 Availability of a Mobile Cloud System

This case study employs the methods described in Section 4.2.1 for identification of bottleneck in mobile cloud systems. It also uses the comparison of results from three distinct sensitivity analysis techniques to validate the proposed approach.

Despite the recent advances, mobile computing suffers from resource scarcity, even on the most modern devices. The most common problems are interruption of wireless connectivity, lack of security, hand-off delay, battery discharge and limited computational power (QI; GANI, 2012). In this context, a new paradigm named Mobile Cloud Computing (MCC) was introduced recently. This paradigm aims to utilize cloud computing resources to overcome the limitations of mobile computing, allowing delivery of more sophisticated and innovative applications to the user. The mobile cloud computing market is expected to reach 45 billion dollars in revenues by 2016 (KOOPMAN, 2012). Considering this financial impact level, it is essential to provide services that can be justifiably trusted, that is, *dependable* services. When a company's workforce needs to move around remote areas to accomplish their duties, the time wasted due to system unavailability may also imply low productivity of its employees.

The availability modeling and analysis of mobile clouds require the investigation of a large number of possible events in client, communication, and server domains. This section presents the analysis of mobile cloud availability based on hierarchical analytical models and distinct sensitivity analysis techniques to assess the impact of each input parameter. This analysis aims to identify the bottlenecks for system improvement. We also use a combined evaluation of results from three techniques which complement each other to deal with the analysis of this system. The results show that the system availability may be improved effectively by focusing on a reduced set of factors which produce large variation on steady-state availability.

The mobile cloud architecture considered for this study is depicted in Figure 5.11, and is an adaptation of the system analyzed in (OLIVEIRA et al., 2013). The architecture is divided into three high-level subsystems: *Mobile Client*, *Mobile Communication*, and *Cloud Infrastructure*. The system is only available if all the three subsystems are working properly. The availability of the *Mobile Client* may be affected by events on four components: *Mobile Hardware*, *Mobile Operating System*, *Battery*, and *Mobile Application*. The event that may cause battery unavailability is its full discharge. We assume that a fully charged spare battery is used to replace the discharged one. The availability of the mobile application is affected by software

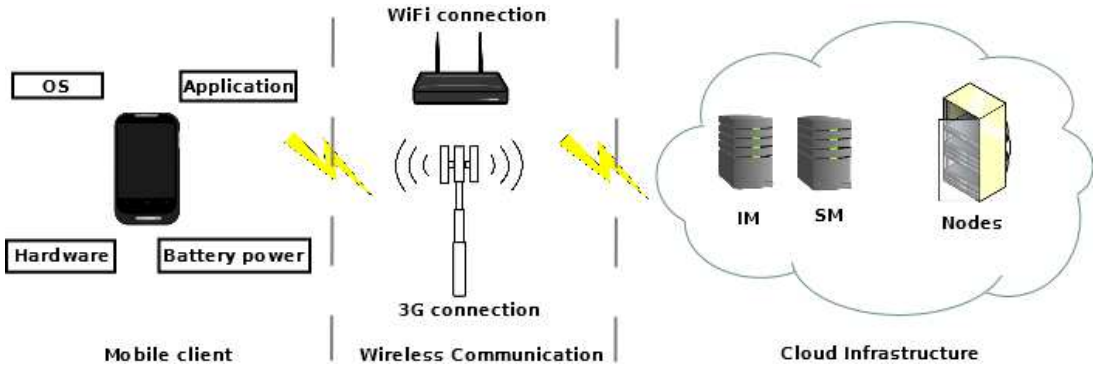


Figure 5.11: Mobile cloud architecture.

faults, or by installation of software updates.

The *Mobile Communication* subsystem has the *WiFi* and *3G* components; it was represented directly in the main system with two blocks in parallel. When both WiFi and 3G fail, the communication subsystem is down, and subsequently the entire system too. The cloud infrastructure has one *Infrastructure Manager* (IM), one *Storage Manager* (SM), and five *nodes*. These components are based on the common building blocks found in frameworks such as Eucalyptus, OpenStack, and OpenNebula, which may be used to implement IaaS clouds (PENG et al., 2009).

A 1:N redundancy is used for the IM. This means that there is one spare machine for N active machines playing the IM role. The same technique is employed for the SM. We consider the number, N, of active servers as a tunable parameter of the system, in order to keep the flexibility of our analysis.

5.2.1 Creating top-level model

Following the proposed methodology, we create a model to represent the high-level view of the mobile cloud architecture illustrated in Figure 5.11. The RBD model in Figure 5.12 has blocks representing the *MobileDevice*, *Battery*, *MobileApp*, *Cloud_IM*, and *Cloud_SM* subsystems, which have their availabilities computed through CTMC sub-models. The blocks representing *WiFi*, *3G*, and the cloud nodes are not expanded into sub-models.

For the top-level RBD from Figure 5.12, a closed-form equation for system availability is expressed by (5.9), derived from standard equations for series-parallel RBDs, as shown in (MACIEL et al., 2011). Each component A_x in this equation is computed from the evaluation of the respective sub-model $x \in$ {MobileDev, Battery, MobileApp, WiFi, 3G, IM, SM, Node}, which can also be done through closed-form equations, if it is possible to obtain them, or through numerical solution. The closed-form equations will also be used to get partial derivatives needed to compute the sensitivities.

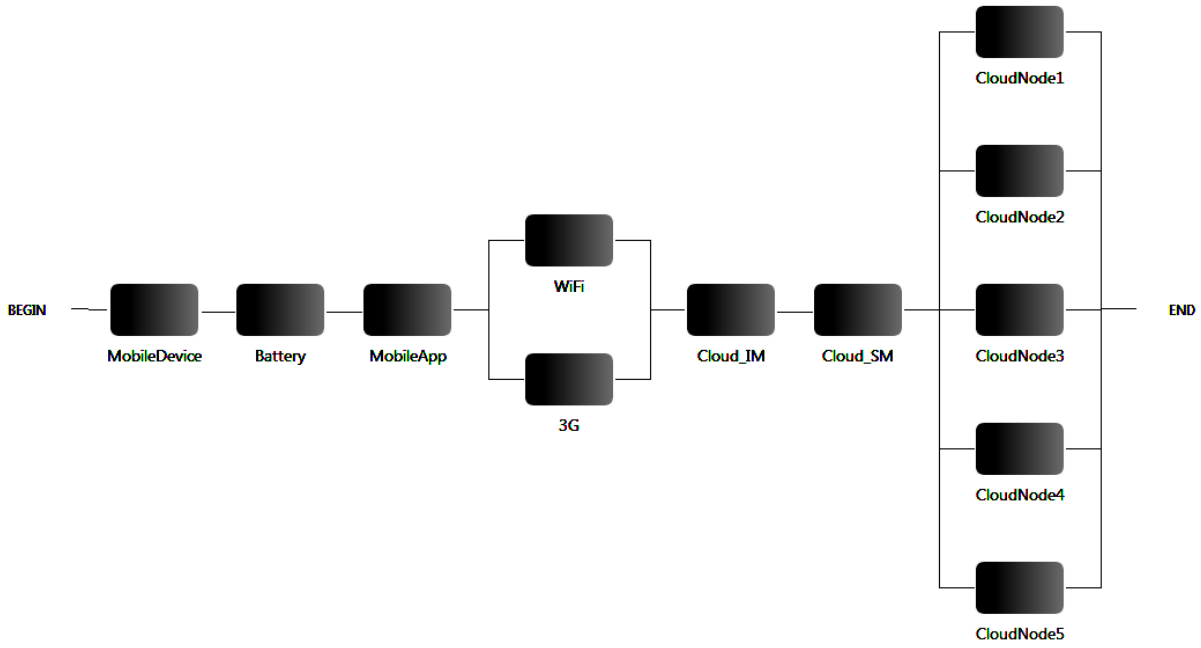


Figure 5.12: RBD model for the mobile cloud.

$$A_{System} = A_{MobileDev} \times A_{Battery} \times A_{MobileApp} \times (1-(1-A_{WiFi}) \times (1-A_{3G})) \times A_{IM} \times A_{SM} \times (1-(1-A_{Node})^5) \quad (5.9)$$

5.2.2 Creating sub-models for specific components

The CTMC that represents the mobile device is depicted in Figure 5.13. In *Up* state, the device works properly. A hardware failure may occur, with rate λh, leading to the device unavailability indicated by the gray colored *HD* state. The hardware repair has a rate μh, taking the system to the *Up* state again. If the operating system fails, the device goes to the *SD* state with rate λs. The correspondent repair happens with rate μs. We assume that the time between failures and the time between repairs (reciprocal of mentioned transition rates) are exponentially distributed. The same assumption is made in the other models presented hereafter.

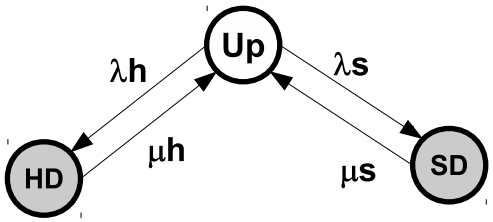


Figure 5.13: CTMC for the mobile device.

Equation (5.10) can be used to compute the availability for the CTMC depicted in Figure 5.13.

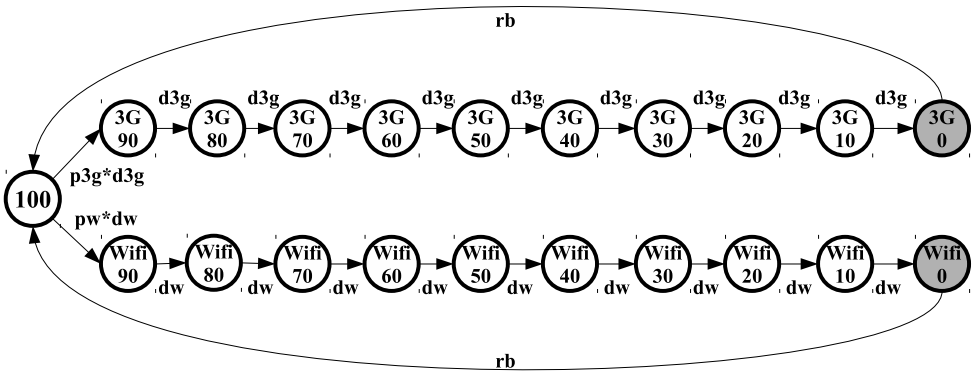


Figure 5.14: CTMC for the battery discharge.

$$A_{MobileDev} = \frac{(\mu_h \times \mu_s)}{(\lambda_s \times \mu_h + (\lambda_h + \mu_h) \times \mu_s)} \tag{5.10}$$

Figure 5.14 presents the CTMC state diagram for the battery discharge process. The energy consumption of a mobile device when communicating through a WiFi interface is different from the consumption when a 3G network is used (BALASUBRAMANIAN; BALASUBRAMANIAN; VENKATARAMANI, 2009). Due to this reason, the CTMC in Figure 5.14 represents the discharge process through two different ways. In state *100* the battery is full and may begin to discharge with WiFi interface enabled, with probability *pw*, or with 3G interface enabled, with probability *p3g*. The discharge is modeled in steps of 10%, so *d3g* represents the discharge rate of this amount of energy when using 3G, and *dw* represents the corresponding rate when using WiFi. We assume that once the battery begins to discharge, there will not be vertical handoff, i.e. the change of network interface. In the states *0 3g*, and *0 Wifi* the battery is fully discharged, so it becomes unavailable. We assume that the user always has a spare battery available, therefore, he only needs to replace the battery and turn the device on gain. The rate of transitions from states *0 3g*, and *0 Wifi* to the state *100* is *rb*. It is also important to highlight that the entire discharging time is known to have a nearly deterministic behavior. We approximate the deterministic behavior by a 10-stage Erlang random variable, as in (TRIVEDI, 2001).

Considering the model depicted in Figure 5.14, the availability of the battery is computed through Equation 5.11.

$$A_{Battery} = \frac{((1 + 9 \times p3g + 9 \times pw) \times rb)}{(d3g \times p3g + dw \times pw + rb + 9 \times (p3g + pw) \times rb)} \tag{5.11}$$

The mobile application has two possible causes of outage: software failure or software update. Figure 5.15 shows the state diagram for mobile application CTMC. In the gray states, *Updating* and *App Failed*, the application is unavailable. The software failure rate is λapp and is assigned to the transition between *App Up* and *App Failed*. The repair rate is labeled as μapp.

The transition from *App Up* to *Update Ready* means that the application developer released a new version. Such a transition occurs with rate λu. When the user decides to install the updated version of the application, the system goes to the state *Updating*. The rate βu is the reciprocal of the mean time passed since the update release and the installation. With rate μu, the application finishes the update process, and becomes available again.

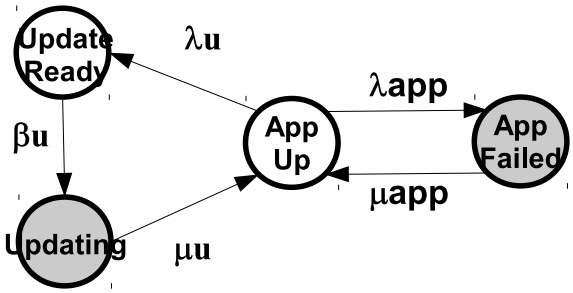


Figure 5.15: CTMC for the mobile application.

The closed-form expression for the mobile application availability is shown in Equation (5.12).

$$A_{MobileApp} = \frac{((\beta_u + \lambda_u) \times \mu_{app} \times \mu_u)}{(\beta_u \times \lambda_u \times \mu_{app} + \lambda_u \times \mu_{app} \times \mu_u + \beta_u \times (\lambda_{app} + \mu_{app}) \times \mu_u)} \quad (5.12)$$

The WiFi and 3G components of our top-level model are blocks which are not expanded in sub-models, therefore, their availabilities are computed using Equations (5.13) and (5.14), respectively. The parameters μ_{wifi} and λ_{wifi} are the repair and failure rates of the WiFi connection, whereas μ_{3g} and λ_{3g} are the repair and failure rates of the 3G connection.

$$A_{WiFi} = \frac{\mu_{wifi}}{\lambda_{wifi} + \mu_{wifi}} \quad (5.13)$$

$$A_{3G} = \frac{\mu_{3g}}{\lambda_{3g} + \mu_{3g}} \quad (5.14)$$

The infrastructure manager (IM) of the cloud environment is composed of N active hosts, and 1 standby spare host. The infrastructure manager needs N hosts to be available, so if one of the active hosts fails, the spare host shall be activated, bringing the system to working condition again. The CTMC for this 1:N redundancy is depicted in Figure 5.16.

The state *US* represents the initial condition, where all N active servers are working properly. The failure of one active host may bring the system to the state *DS*, which indicates that a covered failure occurred, and the spare server is being activated. The transition rate between *US* and *DS* is $N_{im}\lambda_{im}ca_{im}$, where N_{im} is the number of active hosts, λ_{im} is the failure rate of a single host, and ca_{im} is the coverage factor for failures of active servers, i.e., the

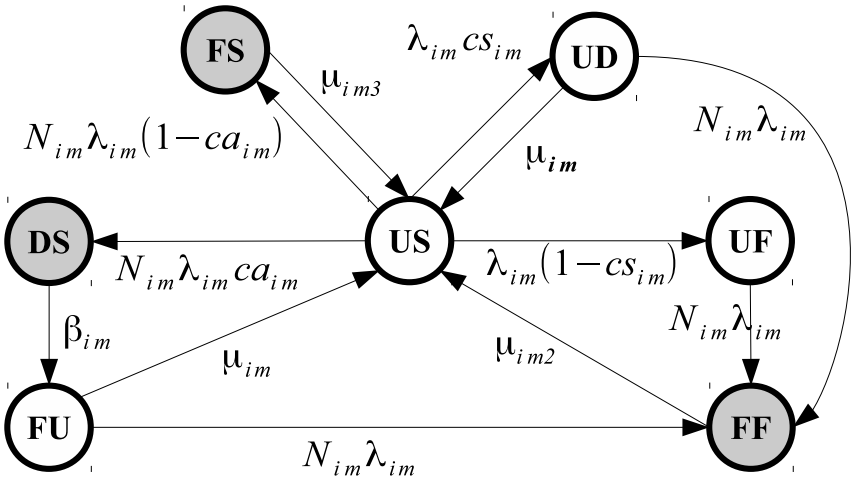


Figure 5.16: CTMC for the Infrastructure Manager.

probability that the failure may be covered by spare host activation. From state *DS*, the state FU is reached with rate β_{im}, indicating that the spare server was activated, and the system is up. In this condition, the failed host may be repaired with rate mu_{im}, in which case the system returns to state *US*, where the spare server is in standby again. If another host fails, with rate $N_{im}\lambda_{im}$, the system goes to state *FF*. In such a condition, a complete repair is executed, so the system goes to the operational state *US* with rate μ_{im2}.

From state *US*, when a not-covered failure occurs in one active host, the activation of the spare host is not triggered, and the system goes to non-operational state *FS*. The rate of such a transition is $N_{im}\lambda_{im}(1-ca_{im})$. In the state *FS*, the failed host is repaired and the system reconfigured to return to an available condition. This repair process is executed with rate mu_{im3} leading the system to state *US*.

The state *UD* is reached due to a detectable failure in the spare host when it is in standby condition and the system is up (state *US*). The rate of this transition is $\lambda_{im}cs_{im}$, and this single event does not affect system availability. The parameter cs_{im} is the coverage factor of spare host failures, i.e., the probability of that failure be detectable while the host is in standby condition. If the spare host is repaired, with rate mu_{im}, the system goes back to state *US*. Although, in state *UD* another host may fail, with rate $N_{im}\lambda_{im}$, bringing the system down (state *FF*).

The state *UF* represents that the spare host has failed while the system is available, but that failure is not detectable while in standby condition. This event occurs with rate $\lambda_{im}(1-cs_{im})$. From this point, one of the active hosts may fail, leading the system to failure (state *FF*) because the spare host cannot be activated. This transition occurs with rate $N_{im}\lambda_{im}$.

Equation (5.15) provides the closed-form expression for the availability of the Infrastructure Manager (IM).

$$A_{CloudIM} = 1 - (\beta_{im}(N_{im}(1+N_{im}+ca_{im}N_{im})\lambda_{im}+(1-cs_{im}+N_{im})\mu_{im})\mu_{im2}\mu_{im3})/((-1+ca_{im})N_{im}^2\beta_{im}\lambda_{im}(N_{im}\lambda_{im}+\mu_{im})\mu_{im2}-((1-cs_{im})\beta_{im}\mu_{im}\mu_{im2}+ca_{im}N_{im}^3\lambda_{im}^2(\beta_{im}+\mu_{im2})+N_{im}\beta_{im}(\mu_{im}\mu_{im2}+\lambda_{im}(\mu_{im}-cs_{im}\mu_{im}+\mu_{im2}))+N_{im}^2\lambda_{im}(ca_{im}\mu_{im}\mu_{im2}+\beta_{im}(\lambda_{im}+\mu_{im2}+ca_{im}\mu_{im2})))\mu_{im3}) \quad (5.15)$$

The Storage Manager (SM) of the cloud infrastructure uses a redundancy mechanism that is similar to that presented for the IM. Therefore, the CTMC models of both IM and SM have the same structure of states and transitions, and their differences are just the values of failure, repair, and coverage parameters used in the transition rates. Due to such similarity the CTMC for the SM is not depicted here, but we present Equation (5.16) used to compute the availability of this component.

$$A_{CloudSM} = 1 - (\beta_{sm}(N_{sm}(1+N_{sm}+ca_{sm}N_{sm})\lambda_{sm}+(1-cs_{sm}+N_{sm})\mu_{sm})\mu_{sm2}\mu_{sm3})/((-1+ca_{sm})N_{sm}^2\beta_{sm}\lambda_{sm}(N_{sm}\lambda_{sm}+\mu_{sm})\mu_{sm2}-((1-cs_{sm})\beta_{sm}\mu_{sm}\mu_{sm2}+ca_{sm}N_{sm}^3\lambda_{sm}^2(\beta_{sm}+\mu_{sm2})+N_{sm}\beta_{sm}(\mu_{sm}\mu_{sm2}+\lambda_{sm}(\mu_{sm}-cs_{sm}\mu_{sm}+\mu_{sm2}))+N_{sm}^2\lambda_{sm}(ca_{sm}\mu_{sm}\mu_{sm2}+\beta_{sm}(\lambda_{sm}+\mu_{sm2}+ca_{sm}\mu_{sm2})))\mu_{sm3}) \quad (5.16)$$

Similar to the WiFi and 3G components of our top-level model, each node in the cloud environment is a single block which is not expanded in a sub-model. Therefore, the availability of each node can be computed through Equation (5.17), where μ_{node} and λ_{node} are the repair and failure rates of each node composing the cloud nodes available for hosting the applications that support the mobile cloud in the server side.

$$A_{Node} = \frac{\mu_{node}}{\lambda_{node}+\mu_{node}} \quad (5.17)$$

5.2.3 Definition of input parameters

Table 5.6 shows the input parameters for the mobile device, battery, and mobile application models. The failure rate of mobile hardware, λ_h, comes from the annual failure rate (AFR) reported in (SQUARETRADE, 2010), but it was reduced by a factor of 10. This reduction aims to yield a more realistic value of the MTTF, based on (SCHROEDER; GIBSON, 2007), which

states that the time between replacements of failed hard drives may be 10 times shorter than the MTTF published by manufacturers, which are also based on AFR values. The application failure rate (λ_{app}) is based on the estimate found in (KIM; MACHIDA; TRIVEDI, 2009b) for web application. The rate of update releases (λ_u) is an average of values found in (APPBRAIN, 2013). The application repair, μ_{app}, and update installation, μ_u, rates are considered to be equivalent to the inverse of the mean time to restart a mobile application.

Table 5.6: Input parameters for the mobile device and mobile application models

Parameter	Value (h^{-1})
λ_h	0.00004452
λ_s	0.00069401
μ_h	0.6
μ_s	3.0
λ_{app}	0.0029700
μ_{app}	120
λ_u	0.00157828
μ_u	120
β_u	0.2

Table 5.7 shows the parameter values for the blocks representing the WiFi and 3G communication networks in the RBD of Figure 5.12, as well as the values for the battery discharge model. The failure rates of WiFi and 3G networks, due to signal blocking and similar problems, are found in (D-LINK WIRELESS N150 ROUTER, 2012) and (COOPER; FARRELL, 2007), respectively. The battery discharging rate is obtained in the specifications published by major smartphone manufacturers (PHONEARENA, 2013).

Table 5.7: Input parameters for the WiFi, 3G, and battery models

Parameter	Value
λ_{wifi}	0.0001 (h^{-1})
μ_{wifi}	0.6 (h^{-1})
λ_{3g}	0.000012 (h^{-1})
μ_{3g}	0.083 (h^{-1})
$p3g$	0.7
pw	0.3
$d3g$	1.4 (h^{-1})
dw	1.1 (h^{-1})
rb	60 (h^{-1})

Table 5.8 depicts the parameters for the RBD of Figure 5.16, representing the IM, and also for the SM and the nodes in the cloud infrastructures. The failure and repair rates are found in the study of private cloud systems presented in (DANTAS et al., 2012b).

Table 5.8: Input parameters for the IM, SM, and nodes

Parameter	Value
λ_{im}	0.0029979 (h^{-1})
μ_{im}	1.0650684 (h^{-1})
μ_{im2}	0.5325342 (h^{-1})
μ_{im3}	0.7100456 (h^{-1})
β_{im}	120 (h^{-1})
N_{im}	1
ca_{im}	0.95
cs_{im}	0.9
λ_{sm}	0.0039984006 (h^{-1})
μ_{sm}	0.89212433 (h^{-1})
μ_{sm2}	0.44606216 (h^{-1})
μ_{sm3}	0.59474626 (h^{-1})
β_{sm}	120 (h^{-1})
N_{sm}	1
ca_{sm}	0.95
cs_{sm}	0.9
λ_{node}	0.003678
μ_{node}	1.1367382

Table 5.9: Availability results

Steady-state availability	Number of Nines	Downtime (h/yr)
0.99553119	2.349808	39.147

5.2.4 Solution of hierarchical model

We computed availability measures for the mobile cloud architecture, using the mentioned input parameters on the hierarchical model. Both, CTMCs and RBDs were solved numerically, first obtaining the availability for each CTMC sub-model, and then using the corresponding values in the RBD model. The results are shown in Table 5.9, including steady-state availability, number of nines, and annual downtime.

The expected annual downtime of 39.147 hours indicates that there is room for improvements, because this value corresponds to more than 1 day of total outage throughout a year. The same conclusion is also drawn by looking directly on steady-state availability and its number of nines. Many companies consider that their systems must reach at least three nines of availability. Amazon EC2 SLA defines 99.95% as its minimum monthly uptime percentage, which once violated makes the customer eligible to receive service credits, as compensation (AMAZON, 2014b). Such a metric equals 3.3 nines of availability, whereas the mobile cloud system evaluated here has about 2.35 nines, confirming that we cannot consider the evaluated measures as satisfactory. Therefore, we must carry out enhancements on this system according to the proposed methodology.

5.2.5 Sensitivity analysis on sub-models and high-level models

This section shows how distinct sensitivity analysis techniques applied to our hierarchical model can detect bottlenecks of this system's availability out of the many parameters that may impact the model results.

As mentioned in Chapter 2, the direct method—based on partial derivatives—is the backbone of many sensitivity analysis techniques. We use such a technique to begin our exploration of model parameters, in order to assess those with significant impact on model results, i.e., system steady-state availability. The sensitivity ranking obtained through the computation of partial derivatives will also enable us to justifiably ignore parameters that have less impact on the measure of interest.

Partial derivatives Table 5.10 presents the sensitivity ranking computed using the partial derivate of Equation (5.9), and subsequent derivatives of equations for each sub-model. The derivative expressions are not shown for the sake of conciseness. The sensitivity ranking in Table 5.10 uses scaled sensitivity indices to remove undesired influences of units, because parameters with very different orders of magnitude are used in this model. The parameters are presented in decreasing order of the sensitivity index.

The top-ranked parameters are the coverage factors of the Storage Manager and Infrastructure Manager. All these coverage factors (cs_{sm}, ca_{sm}, cs_{im}, ca_{im}) have a direct impact on how likely a single host failure will cause the failure of the IM, or of the SM, and a subsequent failure of the whole system. The parameters cs_{sm} and cs_{im} may be improved through the employment of more accurate failure detection mechanisms, while ca_{sm} and ca_{im} may require architectural solutions to enable the spare host to be activated for nearly all possible types of failures in the active hosts. Most parameters ranked in the next highest positions are related to the battery model or the IM and SM models. Therefore, these components should receive higher priority than others to achieve effective improvements in system availability.

The last set of parameters in the ranking are related to the WiFi, 3G, and cloud nodes. The reduced impact of failure and repair rates of these components is expected due to the parallel structures in which they are involved. A single failure of WiFi, or 3G, network does not bring the system down. The same thing happens for a single cloud node. The other parts of the mobile cloud system are prone to single points of failure, or at most to dynamic redundancy mechanisms which do not avoid some downtime before their activation.

Percentage difference Table 5.11 presents the sensitivity ranking based on percentage difference, as shown in Equation (2.7). For this analysis we used a range of values approximately between -50% and +50% of the baseline value for each parameter. For the probability values, we used 0.1 as the minimum value and 1.0 as the maximum value. The minimum value for N_{im} and N_{sm} was 1 and the maximum value was 10.

Table 5.10: Sensitivity ranking based on partial derivatives

Parameter	Description	$\lvert SS(A)\rvert$
cs_{sm}	Coverage factor - Spare SM	0.006242686368
ca_{sm}	Coverage factor - Active SM	0.005679147263
cs_{im}	Coverage factor - Spare IM	0.003946053118
ca_{im}	Coverage factor - Active IM	0.003579911742
rb	Battery replacement	0.002173576439
$d3g$	Battery discharge - 3G	0.001626034283
λ_{sm}	Failure - Storage Manager	0.001259313897
μ_{sm2}	Repair 2 - Storage Manager	0.000871670267
λ_{im}	Failure - Infrastructure Manager	0.000767725345
dw	Battery discharge - WiFi	0.000547542156
N_{sm}	Number of active SMs	0.000534594283
μ_{im2}	Repair 2 - Infrastructure Manager	0.000533421136
N_{im}	Number of active IMs	0.000316933771
μ_{sm3}	Repair 3 - Storage Manager	0.000301953425
$p3g$	Prob. of 3G connection	0.000256681126
μ_s	Repair Mobile OS	0.000230303097
λ_s	Failure Mobile OS	0.000230303097
μ_{im3}	Repair 3 - Infrastructure Manager	0.000190158647
μ_h	Repair - Mob. dev. hardware	0.000073869450
λ_h	Failure - Mob. dev. hardware	0.000073869450
μ_{sm}	Repair 1 - Storage Manager	0.000057255808
pw	Prob. of WiFi connection	0.000039323482
β_{sm}	Activation of spare SM	0.000028434398
μ_{app}	Repair of Mobile App	0.000024446504
λ_{app}	Failure of Mobile App	0.000024446504
μ_{im}	Repair - Infrastructure Manager	0.000022767188
β_{im}	Activation of spare IM	0.000021378375
μ_u	Update installation	0.000012991063
λ_u	Update release	0.000012697941
β_u	Update selection	0.000000293122
μ_{wifi}	Repair - Wifi	0.000000023918
μ_{3g}	Repair - 3G	0.000000023918
λ_{wifi}	Failure - WiFi	0.000000023918
λ_{3g}	Failure - 3G	0.000000023918
μ_{node}	Repair - Node	0.000000000002
λ_{node}	Failure - Node	0.000000000002

Table 5.11: Sensitivity ranking from percentage difference

Parameter	$\|S(A)\|$
N_{sm}	0.0068660546
rb	0.0058528304
N_{im}	0.003583039
ca_{sm}	0.0026962696
cs_{sm}	0.0026128065
ca_{im}	0.0017010243
cs_{im}	0.0016524309
$d3g$	0.0014956874
dw	0.0006411687
μ_s	0.0006241302
$p3g$	0.0005661573
λ_{sm}	0.0004926414
μ_{sm2}	0.0003510625
μ_{im2}	0.0002565759
λ_{im}	0.0001975189

Note that N_{sm} and N_{im} are at higher positions in this ranking, as well as rb—the rate of replacement for the battery—and the coverage parameters of SM and IM (ca_{sm}, cs_{sm}, ca_{im}, cs_{im}) are ranked just after. This occurs because N_{im} and N_{sm} are not supposed to vary in a continuous domain, as assumed in the sensitivity analysis based on partial derivatives. N_{sm} and N_{im} are in the integer numbers domain. Such a fact highlights the usefulness of complementing the sensitivity evaluation by comparing distinct methods. It is important to highlight that increases in N_{sm} and N_{im} decrease the system availability. Thus, the higher position in the ranking denotes that the redundancy mechanism shall be modified to a more adequate approach as the required number of active servers rises.

The battery discharge rate with 3G enabled ($d3g$), and the discharge rate with WiFi (dw), have similar positions in both rankings, and therefore also deserve attention when searching for improvements to system availability. Both rates may be reduced by adjusts in high level protocols of the application running in the mobile device. These adjusts can include compressing transmitted data, and decreasing synchronization frequency. The choice of the 3G provider with best coverage (i.e., strongest signal in most areas) is another action capable of reducing the discharge rate too.

It is also worth noticing that only one parameter in the top 15 from Table 5.10 does not appear in Table 5.11, despite the changes in the order of the other 14 parameters. mu_{sm3} is not one of the 15 most impacting parameters in this percentage difference ranking. Table 5.11 instead indicates mu_s—repair rate of mobile OS—as one of the top 15 parameters.

Design of Experiments We performed the analysis of a factorial design of experiments to provide another point of view on the sensitivity of mobile cloud availability with respect to each parameter. This analysis is performed on the 15 parameters shown in the ranking based on

Table 5.12: Sensitivity ranking from 2^k experiment analysis

Parameter	Effect
rb	0.010521
ca_{sm}	0.007525
N_{sm}	-0.006857
μ_{sm2}	0.004468
λ_{sm}	-0.004225
N_{im}	-0.003982
cs_{im}	0.003141
mu_{sm3}	0.003115
cs_{sm}	0.002748
λ_{im}	0.002174
$p3g$	0.001510
dw	-0.001095
$d3g$	-0.001015
ca_{im}	0.000861
μ_{im2}	-0.000096

partial derivatives. Two levels are considered for each parameter: the minimum and maximum values used in the percentage difference analysis. This 2^k factorial experiment (JAIN, 1991) is evaluated according to the individual effects for the system availability, and these values are shown in Table 5.12.

Parameters rb, ca_{sm} and N_{sm} have the largest effect values, similarly to seen in the analysis with percentage differences. The failure rate of the SM (λ_{sm}), and the repair rate of SM when two hosts are failed (μ_{sm2}), are among the highest in this ranking, as they are also in the partial derivatives ranking. Note that the effect of many parameters related to the SM are among the highest ones in Table 5.12, confirming the importance of this component in the search for system availability improvements.

A refined analysis combining the three rankings—2^k DoE, percentage difference indices, and partial derivative indices—may provide a reduced list of parameters which deserve the highest priority to improve the system availability. We perform such a combined analysis by checking the parameters which appear among the first five positions in at least two out of the three rankings. The parameters which match such a criterion are the coverage factors of failures in the SM (ca_{sm}, cs_{sm}), the required number of active hosts for the SM (N_{sm}), and the rate of battery replacement after discharge (rb). These four parameters can be considered major availability bottlenecks extracted from the 36 parameters comprising the complete hierarchical model.

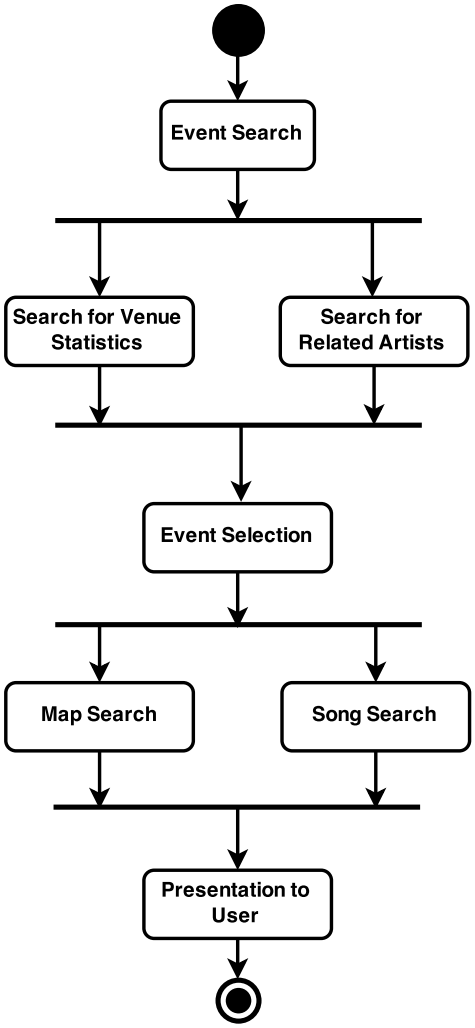


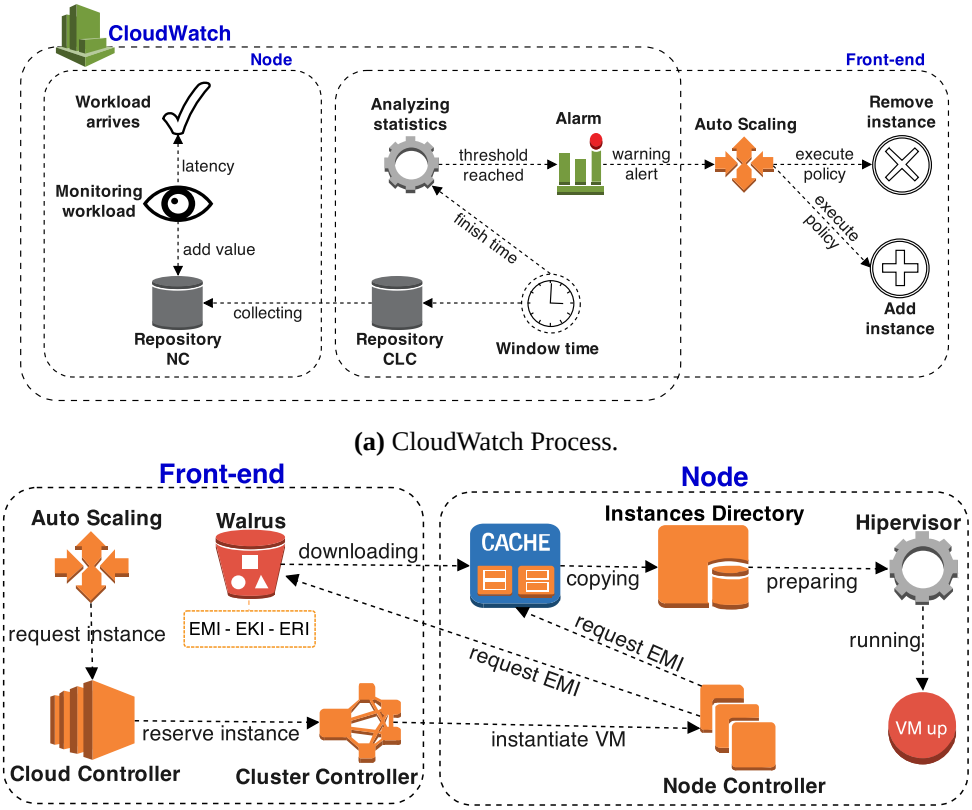
Figure 5.17: Activity Diagram of Event Recommendation Mashup

5.3 Performance of Composite Web Services on Private Cloud

This case study demonstrates the application of our methodology in the context of performance evaluation, what it makes especially distinct from both case studies presented in Section 5.1 and Section 5.2, that carried out availability evaluation instead. It also exercises specifically the method proposed in Section 4.2.2, which addresses SPN as top-level model in the hierarchical composition.

The system evaluated in this section is an event recommendation mashup (MATOS; MACIEL; SILVA, 2013), i.e.: a composite web service, which is hosted on a private cloud. This mashup receives the location (city or neighborhood) from the user and combines data from publicly available web services in order to recommend a musical event that will occur nearby.

Figure 5.17 depicts a Unified Modeling Language (UML) activity diagram for such a service. The first activity is the search for musical events around the current location of the user. The location data might be acquired by communicating with a Global Positioning System (GPS)



(a) CloudWatch Process.

(b) Instantiation Process.

Figure 5.18: Detailed representation of Eucalyptus Auto Scaling process.

application, or manually provided by the user. After obtaining a list of nearby musical event, the application issues concurrent calls to two distinct services: search on venue statistics, such as average users rating of previous events in that concert place; and search for similarities between the lineup of artists in the event and the user's preferences. When data from both services are acquired, the mashup selects the best event based on the venue and artist criteria. Once the event is selected, the application searches for map directions from the user current place to the event venue, and gets a link for one sample song from the main artist in that event. The last activity is the presentation of all gathered information to the user.

The mashup application must take advantage of elasticity mechanisms to avoid performance degradation even in sudden bursts of users requests. The elasticity also avoids wasting system resources in low workload periods. The Eucalyptus Auto Scaling mechanism is responsible for adapting the number of VMs that run the web service. As explained in Section 2.1, the Auto Scaling interacts with the CloudWatch and Elastic Load Balancer components to avoid performance degradation. This is accomplished by creating new VM instances when a given metric reaches a threshold predefined by the system administrator.

CloudWatch monitors information that is used by Auto Scaling to add or remove instances. Figure 5.18 (a) details the operation of this process. Note that while workload arrives in a VM, the CloudWatch, after a certain latency, monitors a metric (e.g.: average CPU uti-

lization of all VMs) and adds the value in the Node Controller (NC) repository, along with the time stamp of data collection. This information is collected from repositories of all NCs each 5 minutes (by default), and sent to a unified repository on Cloud Controller (CLC) that gathers data from all clusters. At the end of a specified period of time (time window), CloudWatch aggregates the metric values from the CLC repository, which were added within the range of the time window. Statistics (e.g.: minimum, maximum, average) are computed from the aggregate data, and if the result reaches the specified threshold, CloudWatch Alarm would modify its state from *ok* to *alarm*. If an alarm state is maintained over a predefined number of time windows, the CloudWatch Alarm triggers the warning threshold for the Auto Scaling, which performs actions (i.e.: add or remove instances) based on policies previously determined (EUCALYPTUS, 2014b).

Every request for the creation of a new VM instance takes some time to be fully serviced. That time depends on factors of the Eucalyptus instantiation process, which are explained as follows. As shown in Figure 5.18 (b), when the auto scaling mechanism —or a user— calls for a new VM instance, the CLC checks the existence of available resources for creating such a VM. This is accomplished through queries to the Cluster Controller (CC), which stores information about its nodes. If there are enough resources, the CLC reserves a unique identification number for the instance, the CC assigns the node where the VM should be instantiated, and the NC starts copying three images: Eucalyptus Machine Image (EMI); Eucalyptus Kernel Image (EKI); and Eucalyptus Ramdisk Image (ERI). These images can be downloaded from Walrus or copied from a local cache, maintained by NC (EUCALYPTUS, 2014a).

The cache will not be used if it is not enabled in system configuration or if the requested EMI had never been instantiated on that node before. In the latter case, the CLC transfers EMI from Walrus to the cache and to the NC instance directory. Note that EKI and ERI are also downloaded if they are not in the NC cache. When the cache is not enabled, the EMI is transferred directly to the NC instance directory and is not cached for later use. When the cache is working and the node already has a copy of the EMI, the CLC does not download from Walrus, but only copies the EMI from the cache to the instance directory (EUCALYPTUS, 2014a).

After obtaining the EMI, EKI, and ERI, the NC interacts with the hypervisor (KVM, Xen, or VMware) to prepare the disk space required by the VM instance, according to the chosen VM type (e.g.: m1.small, c1.xlarge, etc.). Such a procedure usually requires creating, partitioning and formatting virtual block devices. The hypervisor, then, starts the current VM, completing the instantiation process (EUCALYPTUS, 2014a).

In a previous work (CAMPOS et al., 2015), we identified three main phases that occur for instantiating a VM in a Eucalyptus cloud: (i) resource and instance reservation; (ii) copy (or download) of the VM image files (EMI, EKI, and ERI); and (iii) VM preparation and deployment. These three phases are considered in a CTMC model for VM instantiation that is presented further.

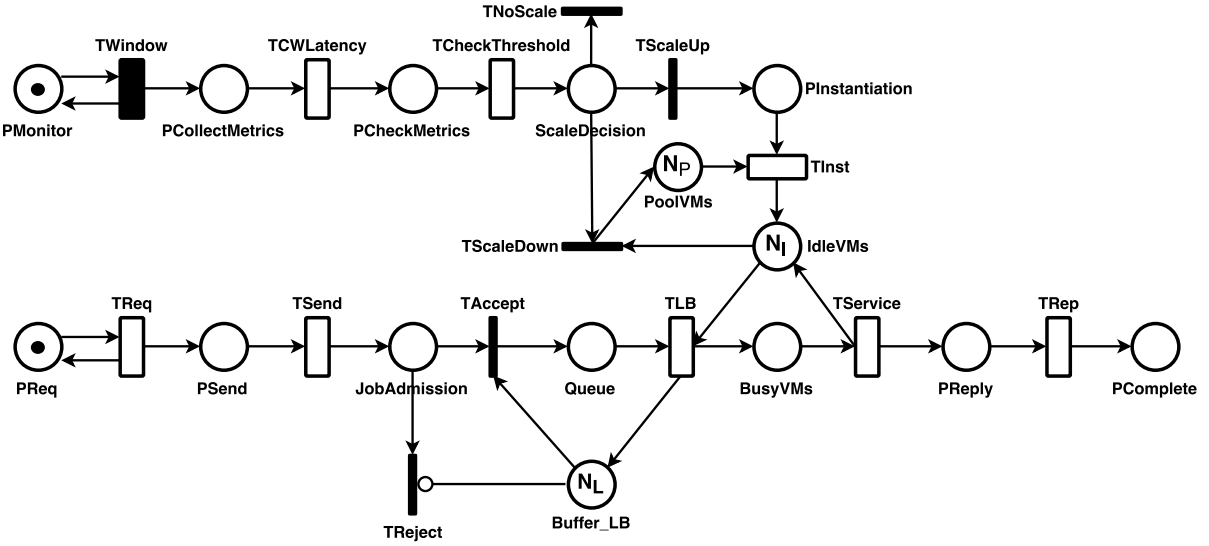


Figure 5.19: SPN model for the scalable web service on private cloud

A hierarchical model is proposed to evaluate the whole system just described in this section. The approach enables representing details of specific processes, such as the instantiation of VMs and the calls for the providers of specific web services that compose the mashup application. The hierarchical model comprises an SPN as main model and CTMCs as sub-models.

5.3.1 Creating top-level model

The SPN, depicted in Figure 5.19, is a performance model of the web service deployed in a private cloud with the auto scaling mechanism. This model captures the main activities of the system, from client requests to service completion. It also represents the creation and termination of VM instances through the autoscaling mechanism.

A token in place **PReq** represents a user request for the mashup. The firing delay of transition **TReq** corresponds to the mean time between arrivals of requests. When **TReq** fires, it stores one token in the place **PSend**, which denotes the transmission of requests through the network. The network latency between client and server is assigned to transition **TSend**. A token in place **JobAdmission** represents user request arrival in the cloud. Such a request may be admitted by the Load Balancer if its buffer is not full (immediate transition **TAccept**), or discarded otherwise (immediate transition **TReject**). If the request is admitted, it waits in the place **Queue** for being assigned to one of the VMs hosting the mashup application. The time spent by the Load Balancer to forward the request is represented by transition **TLB**. Notice that **TLB** requires one token from place **IdleVMs**, which initially has two tokens, denoting the number of available VM instances we defined for initial cloud setup.

The VMs that are busy processing a user request are represented by tokens in place **BusyVMs**. The time that one VM takes to serve a request is assigned to transition **TService**, which is refined by a CTMC submodel presented further. Notice that transition **TService** has an infinite server firing policy in order to properly represent the parallel execution of all requested

Table 5.13: Immediate transitions of the SPN model for scalable web service on private cloud

Transition	Description	Enabling function	Priority
TNoScale	Decision of keeping the current number of VMs	#IdleVMs≥1	1
TScaleUp	Decision of increasing the current number of VMs	#IdleVMs<1	1
TScaleDown	Decision of decreasing the current number of VMs	(#IdleVMs>4) AND (#Queue<1)	2
TAccept	Decision of accepting the user request	–	1
TReject	Decision of rejecting the user request	–	1
TComplete	Completion of user request	–	1

VMs. After processing a request, a response is sent back to the client. This activity is denoted by transition **TRep**. Place **PComplete** represents the client response arrival. The transition **TComplete** firing consumes all tokens in **PComplete**, avoiding accumulation of tokens in that place and subsequent problems for model solution.

The auto scaling mechanism is modeled by places and transitions in the upper part of the SPN. The place **PMonitor** and transition **TWindow** denote the periodic trigger of CloudWatch monitor. **TWindow** is a deterministic transition, so it properly represents the fixed time interval (window) at which the metrics are requested. A token in the place **PCheckMetrics** enables the transition **TCWLatency**, which denotes the time for collecting data from nodes and summarizing collected data. When **TCWLatency** fires, one token is stored in **PCheckMetrics**. The transition **TCheckThreshold** delay represents the time for analyzing summarized data according to the predefined thresholds. **TCheckThreshold** stores a token in place **ScaleDecision**, which has two outgoing transitions: **TNoScale** and **TScaleUp**, that denote the options of holding or increasing the current number of VM instances, respectively. Table 5.13 presents the enabling functions for those two transitions.

If the transition **TNoScale** fires, it consumes a token from place **ScaleDecision**. Otherwise, if the transition **TScaleUp** fires, it consumes a token from **ScaleDecision** and stores a token in place **PInstantiation**. The transition **TInst** delay represents the time required for instantiating one VM in the private cloud. A CTMC submodel was developed to represent the VM instantiation process in detail, hence the delay of **TInst** is computed from that submodel. Notice that **TInst** requires one token available in the place **PoolVMs**. Such a place denotes the maximum capacity (in number of VMs) that might be added to the mashup application. For the current analysis, there are five tokens in **PoolVMs**, indicating that the autoscaling might create up to five new VMs. When transition **TInst** fires, it stores one token in the place **IdleVMs**. The immediate transition **TScaleDown** represents the activity of terminating VMs in periods of low workload, in order to avoid underutilization of resources.

Table 5.13 shows the enabling functions and priorities for each immediate transition of the SPN model. **TNoScale** is enabled whenever there is at least one token in the place **IdleVMs**.

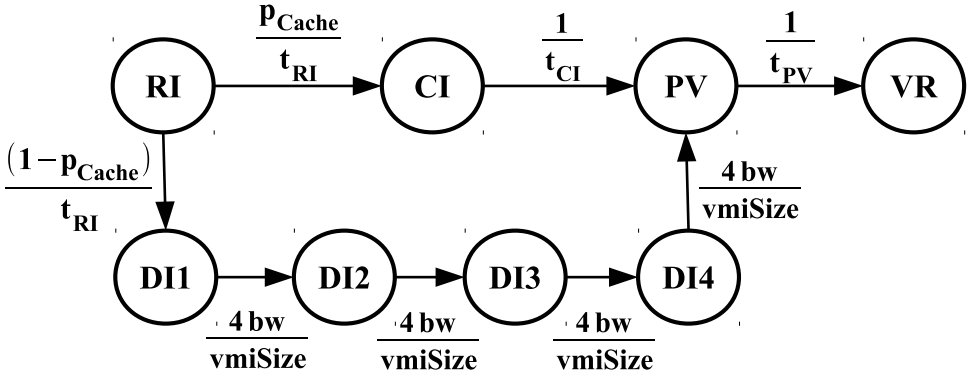


Figure 5.20: CTMC model for the VM instantiation performance.

TScaleUp is enabled if there is less than one token in **IdleVMs**, i.e.: if the place is empty. **TScaleDown** requires more than four tokens in **IdleVMs**. Considering only the enabling functions, **TNoScale** and **TScaleDown** could be simultaneously enabled, but **TScaleDown** was assigned the highest priority of both, so it always fires first. The transitions **TAccept** and **TReject** do not have any enabling functions because they depend only on the existence of tokens in place **Buffer_LB**. There is an inhibitor arc in **TReject** coming from **Buffer_LB**, so **TReject** can only fire if **Buffer_LB** is empty. The arc from **Buffer_LB** to **TAccept** only enables transition **TAccept** if there is at least one token in **Buffer_LB**. **TComplete** is a single sink transition which does not need any enabling function.

It is important to highlight that this SPN shall not be solved through numerical analysis, but only through simulation, due to the existence of a non-exponential timed transition (**TWindow** is deterministic). Next sections deal with the two CTMC models that also comprise our hierarchical modeling approach.

5.3.2 Creating sub-models for specific components

Two CTMC submodels were created to compute the time spent in the VM instantiation process and the response time of the composite web service mentioned in Section 5.3.1. The results from those provide the mean delay values for **TInst** and **Tservice** transitions, respectively, of the SPN main model.

VM instantiation submodel

Figure 5.20 depicts the CTMC created to represent the instantiation process of a VM in a Eucalyptus private cloud. This model comprises the states **RI**, **CI**, **DI1**, **DI2**, **DI3**, **DI4**, **PV**, and **VR**. State RI represents the reservation of a VM instance on Cluster Controller. State **CI** denotes the copy of VMI (virtual machine image) files –i.e, EMI, EKI, and ERI – to the directory of instances in Node Controller. States **DI1**, **DI2**, **DI3**, and **DI4** represent the download of VMI to the Node Controller cache. State **PV** means that the hypervisor is formatting the virtual block

device and configuring the VM. Finally, **VR** represents the VM running, so the instantiation is complete.

We have used average results from an experimental testbed as input for the VM instantiation model. Statistical analysis on experimental data does not reject exponential distributions as good fit for the time of every activity in VM instantiation, except by VMI download $t\_DI = vmiSize/bw$. For such a reason, our model represents t_DI by means of a 4-stage Erlang distribution, using a moment matching method described in (WATSON J.F.; DESROCHERS, 1991). The stages are denoted by **DI1**, **DI2**, **DI3**, and **DI4** with rate $1/(t\_DI/4) = (4 \times bw)/vmiSize$ for the output transition of each state.

The VM instantiation process begins at **RI** state. The model goes from **RI** to **CI** with rate $p\_cache \times (1/t\_RI)$, denoting the case when VMI is already in the node's cache. In **CI** state, the transition to **PV** state occurs with rate $1/t\_CI$. The transition from **RI** to **DI1** occurs with rate $(1 - p\_cache) \times (1/t\_RI)$ and represents that the node needs to download the VMI from Walrus. From **DI1**, the model continues to **DI2**, **DI3**, and **DI4**, with rate $(4 \times bw)/vmiSize$ in each transition until it reaches the **PV** state , indicating the end of VMI download and the beginning of VM preparation. The last step of instantiation process occurs when the model goes from **PV** state to **VR** state with $1/t\_PV$ rate.

Mashup application submodel

A CTMC model was created to evaluate the performance of the single mashup application, without the cloud or other infrastructure aspects. The CTMC input data are the response times of each individual service depicted in the UML diagram of Figure 5.17. Each state in the CTMC, depicted in Figure 5.21 denotes a service request, the only exception is the final state. The transition rates are estimated as the reciprocal of mean response time for each web service $(1/mrt_X)$. All response times are assumed to be exponentially distributed. The state "**Event Analysis**" represents the execution of concurrent calls to the "Search for Venue Statistics" and "Search for Related Artists" services. The model might go to state "**Venue Stats Finished**", with a rate of $1/mrt_{VS}$, or to "**Similar Artists Finished**", with a rate of $1/mrt_{SA}$, indicating which web service replied first. The state "**Top Event Selection**" indicates that the responses of both services, i.e.: "Search for Venue Statistics" and "Search for Related Artists" were received. "**Top Event Selection**" also denotes the analysis of all events using the previously collected data, which is completed with rate $1/mrt_{TS}$. Then the model goes to state **"Additional info search"**, representing the concurrent execution of queries to "**Map Search**" and "**Song Search**" services. The model reaches " textbfMap Search Finished" with rate $1/mrt_{MS}$, and "**Song Search Finished**" with rate $1/mrt_{SS}$, denoting which service replied first. After finishing both services, the CTMC finally reaches state "**Complete**", an absorbing state that indicates the end of mashup execution.

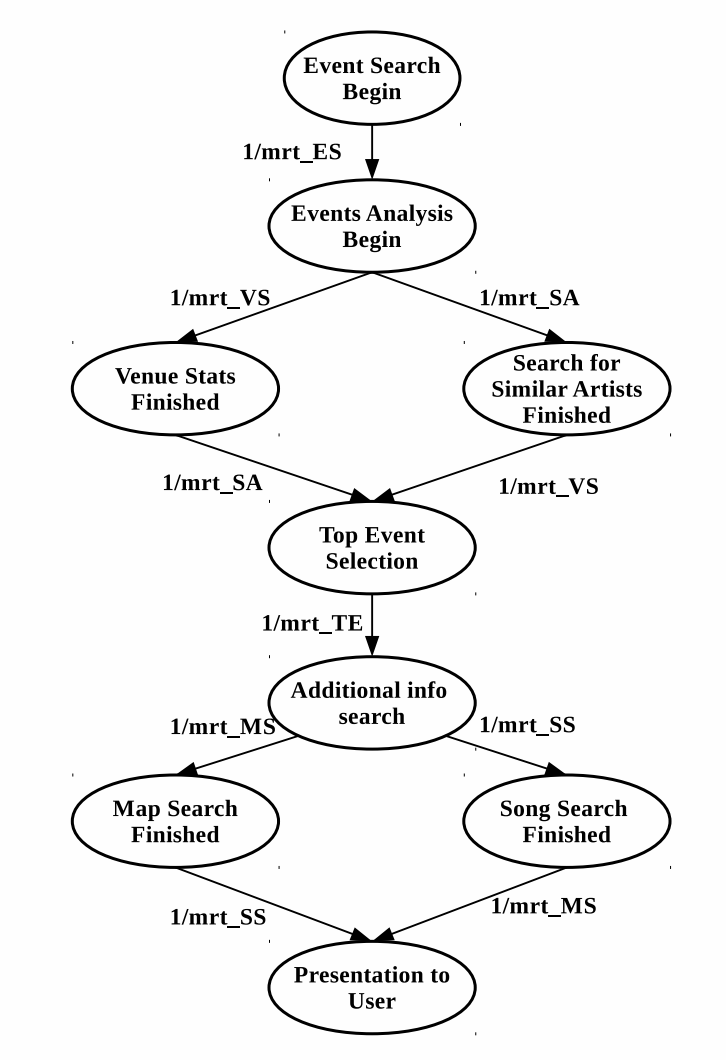


Figure 5.21: CTMC model for the event recommendation mashup

Table 5.14: Timed transitions of the SPN model for scalable web service on private cloud

Transition	Description	Value (s)
TWindow	Time window for CloudWatch	60.0
TCWLatency	Delay for metrics collection on CloudWatch	60.0
TCheckThreshold	Time for computing metrics and compare to thresholds	1.0
TInst	Time for instantiation of a new VM	37.2
Treq	Time between user requests	4.0
Tsend	Network latency to send request	0.2
TLB	Time for Load Balancer forward request	1.0
Tservice	Response time of mashup	6.9
Trep	Network latency to send response	0.2

5.3.3 Definition of input parameters

Table 5.14 presents the delays assigned to all timed transitions of the SPN main model. The values for delays of **TWindow**, **TCWLatency**, **TCheckThreshold**, **TLB**, **Treq**, and **Trep** were obtained in a Eucalyptus private cloud testbed, using default configuration parameters for the Eucalyptus CloudWatch (EUCALYPTUS, 2014a). The values for **TInst** and **Tservice** delays come from distinct CTMC submodels presented further, which represent the VM instantiation process, and the event recommendation composite web service. Therefore the evaluation of our system relies on a hierarchical heterogeneous model.

Table 5.15 presents a description of VM instantiation model parameters and their values. The parameter **pCache** is assigned to 0.9 to represent a scenario with 90% of nodes containing the VMI in their cache. All other parameter values were collected in testbed experiments on a Eucalyptus private cloud. The machines in the cloud had the following hardware configuration: Intel(R) Core(TM) i7-3770 3.4 GHz CPU, 4 GB RAM DDR3, 500 GB SATA HD. One machine was configured as front-end for execution of CLC, CC, SC, and Walrus. Other five machines are the nodes running the NCs. All hosts run the CentOS Linux 6 operating system and the Eucalyptus 3.4 platform. The VMs run the Ubuntu Server 14.04.01 LTS operating system. A Fast Ethernet network was adopted to connect the PCs through a single switch. This environment has all that is needed for the purposes of this study, since the VM instantiation is a process involving only the front-end and the specific node where the VM is allocated. Therefore, the limited size enabled accurately monitoring every stage of the instantiation process.

In order to obtain the value for *t_CI*, the VM was previously instantiated in all nodes, so their caches stored the VMI. Therefore in the following experiments for measuring *t_CI*, the nodes would only copy the VMI from local cache. On the other hand, for obtaining *bw* –the effective bandwidth used during VMI download– the cache of every node was erased in the beginning of each experiment run. Therefore, the nodes always had to download the VMI from Walrus and *bw* was measured properly. The average value for each parameter was computed after 50 experiment replicas.

Table 5.15: Parameter values for the CTMC model of VM instantiation.

Parameter	Description	Value
p_Cache	Probability that VMI is already in cache	0.9 (90%)
t_RI	Mean instance reservation time	0.280 s
t_CI	Mean VMI local copy time	7.624 s
bw	Network bandwidth	10.5 MB/s
vmiSize	VMI size	2048 MB
t_PV	Mean VM preparation time	10.603 s

Table 5.16: Parameter values for the mashup CTMC model.

Parameter	Description	Value (s)
mrt_ES	Mean resp. time of Event Search	2.333
mrt_VS	Mean resp. time of Venue Search	0.324
mrt_SA	Mean resp. time of Similar Artists	2.286
mrt_TS	Mean resp. time of Top Event Selection	0.226
mrt_MS	Mean resp. time of Map Search	0.452
mrt_SS	Mean resp. time of Song Search	1.909

Table 5.16 presents the values assigned to parameters of the mashup application model. Each mean response time was obtained through measurements on a real mashup application, calling specific web services provided by Foursquare, Google Maps, Last.FM, and Eventful.

5.3.4 Solution of hierarchical model

The models presented in Section 5.3.1 and Section 5.3.2 were evaluated to assess the performance of the scalable web service system. Stationary simulation of the SPN depicted in Figure 5.19 enabled obtaining measures such as mean queue size, average number of busy VMs, average utilization of VMs, and mean response time to the user.

The simulation was executed for a confidence level of 95%, maximum relative error of 5%, warm-up period of 50 runs, run size (i.e.: number of times each transition fires) of 1000, and maximum simulation time of 120 seconds. The CTMC sub-models were solved through stationary analysis, providing the values to be assigned as delays of **TInst** and **TService** transitions.

Table 5.17: Performance measures

Measure	Expression	Value
Utilization of VMs (%)	E{#BusyVMs})/(E{#IdleVMs}+E{#BusyVMs})	38.3 %
Average number of busy VMs	E{#BusyVMs}	1.716
Average number of idle VMs	E{#IdleVMs}	2.773
LB queue size (#of requests)	E{#Queue}	0.432
Mean response time - Rsp - (s)	NRequests/(P{#PReply>0}×(1/TReply))	9.029 s

Table 5.17 presents the performance measures computed considering the baseline con-

figuration of parameters values shown throughout Sections 5.3.1 and 5.3.2. The average utilization of VMs is around 38.3%, what shows that the system has enough capacity to serve user requests with the allocated resources. Such a capacity is partially provided by means of additional VMs created by the auto scaling mechanism. This is confirmed by the sum of average number of busy VMs and average number of idle VMs, that is equal to approximately 4.48. If the auto scaling mechanism were not working –and we had only two VMs– the average utilization could reach about 85%, incurring in risks of bad performance for the users, mainly during high workload bursts. The average load balancer queue size is another measure that shows the system is not overloaded, since the requests do not wait in the queue for being distributed to the VMs.

The mean response time of the system (*Rsp*) is 9.029 seconds. This is the round-trip time interval elapsed from the dispatch of user request to response arrival. The measure expression on the SPN is $Rsp = NRequests/(P\{\#PReply > 0\} \times (1/TReply))$, where *NRequests* is the average number of requests in the system. *NRequests* is computed through the expression $E\{\#PSend\}) + (E\{\#JobAdmission\}) + (E\{\#Queue\}) + (E\{\#BusyVMs\}) + (E\{\#PReply\}$.

By summing up the response time of the mashup application, the request and reply network delays, and the time for load balancer distribution, the result is close to the system response time (9.029 s), indicating that requests spend little time in queue. Even then, the mean response time of this system might be shortened by tuning some of its many parameters and components. In order to identify the most effective points for a response time improvement, it is important to assess the measure sensitivity to models' parameters.

5.3.5 Sensitivity analysis on sub-models and high-level models

We employ the following technique to achieve a unified view of the response time sensitivity with respect to all parameters.

First, a sensitivity ranking is computed for the main model, considering all of its parameters, but without detailing the sensitivity with respect the parameters of the submodels. We used the percentage difference technique (see Section 2.3) to compute the sensitivity ranking of the main model. The minimum and maximum values used for computing the percentage difference index are shown in Table 5.18. Such a technique was adopted because the model can only be solved through simulation, making it not suitable for the technique of partial derivatives. Next, we computed the sensitivity rankings for the sub-models through partial derivatives, because they are solved through analytical methods instead of simulation. In such a case, sensitivity indices from partial derivatives are obtained from the underlying equations with smaller computational effort than percentage difference indices. We multiplied each sub-model sensitivity index by the corresponding index on the main model, obtaining a composite index. For instance, $S_{mrt\_ES}(Rsp) = SS_{mrt\_ES}(TService) \times S_{TService}(Rsp)$. The same rule will be applied for all parameters of the CTMC model used to compute **TService**, and similarly for the parameters of

Table 5.18: Minimum and maximum parameter values for computing the sensitivity indices for the main model

Transition	Minimum (s)	Maximum (s)
TWindow	60	240
TCWLatency	60	300
TCheckThreshold	1	5
TInst	15	60
Treq	4	8
Tsend	0.2	1
TLB	0.2	1
Tservice	5	10
Trep	0.2	1

the CTMC model used to compute **TInst**.

The unified sensitivity ranking comprises the composite indices for the parameters of sub-models, as well as indices for parameters of the main model which are not related to sub-models. We could also build sensitivity rankings for metrics such as utilization of VMs and load balancer queue size, but we focused on system mean response time (Rsp) because it is the most user-centered performance measure among those listed on Table 5.17.

Sensitivity analysis results

Table 5.19 shows the sensitivity ranking for the main model. Notice that the ranking is sorted in decreasing order of absolute value. This is because the magnitude of the index indicates how much impact it causes on the output measure. It is important to highlight that negative indices only indicate an inverse relationship between the parameter and the measure of interest, i.e.: if the input parameter value increases, the output measure decreases and vice versa.

It is possible to notice that **TService** — the execution time of the event recommendation web service — is the most impacting of the parameters, followed by **TLB**, **TRep**, and **TSend**. The other parameters have smaller impact on the system response time. **TReq** has an intermediate impact if compared to the other ones, and it is the only parameter with a negative sensitivity index. This is because the smaller is the time between users requests the higher is the load to be processed by the VMs, incurring in higher system response time too.

The ranking enables us to state that **TService**, **TLB**, **TRep**, and **TSend** are the most effective points to invest in order to decrease the response time. Although, we must also analyze the sub-models that provide *TService* and *TInst* to achieve a detailed view of the impact that each sub-component has in this system.

The VM instantiation sub-model is used to compute **TInst**, so we evaluate the sensitivity of **TInst** with respect to each parameter of the respective CTMC. It is worth mentioning that we use scaled indices in Table 5.20 due to the different units of parameters of VM instan-

Table 5.19: Sensitivity ranking for the main model

Parameter	S(Rsp)
TService	0.45763
TLB	0.13788
TRep	0.11303
TSend	0.11466
TReq	-0.05808
TWindow	0.00617
TCWLatency	0.00489
TInst	0.00256
TCheckThreshold	0.00176

Table 5.20: Sensitivity ranking for the VM instantiation submodel

Parameter	SS(TInst)
pCache	-4.52843
vmiSize	0.52363
bw	-0.52363
t_PV	0.28465
t_CI	0.18421
t_RI	0.00752

tiation submodel. The scaled indices enable us to compare those parameters fairly. As shown in Table 5.20, the probability of finding the VM image on the node's cache (**pCache**) is the factor that produces the largest impact on the time for VM instantiation. Varying **pCache** cause changes on **TInst** that are one order of magnitude larger than the effect of other parameters. This is denoted by the difference between the sensitivity indices. Note that if **pCache** increases, **TInst** will decrease. The same will occur with **bw** — the network bandwidth — but to a lesser extent.

Table 5.21 presents the sensitivity of **TService** with respect to each parameter of the mashup sub-model.

The response time of the **Event Search** and **Similar Artists** providers (**mrt_ES** and **mrt_SA**) are the parameters with largest impact on **TService**. The sensitivity with respect to the **Song Search** response time **mrt_SS** is also an important factor in this model. The other three parameters have sensitivity indices much smaller than those top three. We also use scaled sensitivities in this ranking in order to keep consistency with VM instantiation submodel ranking and enable fair comparison of parameters with distinct units when building the unified ranking. Notice that the sensitivity indices on main model do not need to be scaled because the percentage difference method already provides nondimensional values.

The next step in our analysis is to build a unified ranking considering all parameters from both sub-models as well as the parameters from the main model. We used the composition technique based on the chain rule (see Section 4.2). We multiplied each sub-model sensitivity index by the corresponding index on the main model, obtaining a composite index. For

Table 5.21: Sensitivity ranking for the mashup sub-model

Parameter	SS(TService)
mrt_ES	0.33906
mrt_SA	0.32711
mrt_SS	0.26727
mrt_TS	0.03284
mrt_MS	0.02274
mrt_VS	0.01096

Table 5.22: Unified sensitivity ranking for the general model and submodels

Parameter	S(Rsp)
mrt_ES	0.15517
mrt_SA	0.14969
TLB	0.13977
mrt_SS	0.12231
TSend	0.11466
TRep	0.11303
TReq	-0.05808
mrt_TS	0.01503
pCache	-0.01162
mrt_MS	0.01041
TWindow	0.00617
mrt_VS	0.00502
TCWLatency	0.00489
TCheckThreshold	0.00176
vmiSize	0.00134
bw	-0.00134
t_PV	0.00073
t_CI	0.00047
t_RI	0.00002

instance, $S_{mrt\_ES}(Rsp) = SS_{mrt\_ES}(TService) \times S_{TService}(Rsp)$. The same rule is applied for all parameters of the CTMC model used to compute **TService**, and similarly for the parameters of the CTMC model used to compute **TInst**. Such a method generated the sensitivity ranking presented in Table 5.22. It is worth mentioning that the impact of **TService** when considering the main model (see Table 5.19) is confirmed by the presence of **mrt_ES** and **mrt_SA** on the top of the ranking. Thus, the most important action to decrease the system response time in a significant degree is the replacement of current **Event Search** and **Similar Artists** web service providers by faster ones, or at least the tuning of performance-related configurations in their API (Application Programming Interface), such as the format and maximum length of the response.

Notice that in Table 5.19 there is an effective difference between $S_{TService}(Rsp)$ and $S_{TLB}(Rsp)$, but in this unified ranking the sensitivity with respect to **TLB** is comparable to that of **mrt_ES**, **mrt_SA**, and **mrt_SS**. Therefore, the load balancer is also a high priority component for the improvement of the overall system performance.

It is important to stress that some parameters of the sub-model related to **TService** are not so important for system improvement as other components from the main model. This is the case of **mrt_VS**, which has a sensitivity index among the half least impacting of all 19 parameters. The impact of **mrt_VS** on system response time is similar to the impact of **TCWLatency** and **TWindow** from the main model. Those system components are little relevant for the enhancement of system's response time in the current setup, and therefore they deserve low priority during system upgrades. Most parameters from the VM instantiation sub-model, with the exception of **pCache**, are in the same lower-ranking situation.

Such conclusions encompassing factors from distinct models are valuable results of the unified ranking analysis, and demonstrate the applicability and importance of our proposed approach. As an auxiliary view, we present a comparison of impact among some parameters through scatter plots. Figure 5.22 depicts the impact of parameters **mrt_ES**, **mrt_SA**, and **mrt_SS** on system response time, computed from the SPN main model. We grouped these parameters on Figure 5.22 because they belong to the same sub-model, and are among the top parameters in the sensitivity ranking of Table 5.22. The plot is generated by fixing all parameters at their baseline values (see Tables 2.8, 5.15, and 5.16), except by one parameter that is varied through a specific range in steps of about 10%, enabling the comparison of impact on the system response time. Notice that the slopes of **mrt_ES**, **mrt_SA**, and **mrt_SS** are similar, just like their sensitivity indices on Table 5.22.

Figure 5.23 presents the impact of parameters **TLB**, **TSend**, and **TRep**, which belong to the SPN main model. **TLB** shows a slightly higher impact on system response time than **TRep** and **TSend** do. Such a behavior matches the indices shown in Table 5.22, as well as the similarity of slopes to the **mrt_ES**, **mrt_SA**, and **mrt_SS** parameters. Notice that we kept the range of Y-axis (system response time) the same for all plots (Figure 5.22, Figure 5.23, and Figure 5.24), so we can compare the slopes of lines from distinct graphs.

Figure 5.24 depicts the impact of parameter **pCache** on system response time. This is one parameter from the VM instantiation sub-model, and has a much smaller sensitivity index than the previously plotted parameters. This relatively lower impact of **pCache** is noteworthy by comparing Figure 5.22 and Figure 5.23 to Figure 5.24. It is worth highlighting that even varying **pCache** with steps larger than 10%, its effect on system response time is limited to about 0.1 second throughout the plot, whereas in Figure 5.22 the impact reaches around 0.8 seconds. On the other hand, **pCache** may be one of the parameters most easily tunable in the system, if compared to specific services response times (**mrt_ES**, **mrt_SA**, and **mrt_SS**) or parameters related to network latency (**TSend**, **TRep**). **pCache** has a higher sensitivity index than many other parameters, and therefore deserves attention from system administrators.

Comparisons with other parameters could be made here, but if we evaluate systems with dozens of parameters is harder and error-prone to compare all parameters through scatter plots. The unified sensitivity ranking presented here is advantageous because it enables the fast identification of major and minor impacting parameters through accurate indices, as demonstrated.

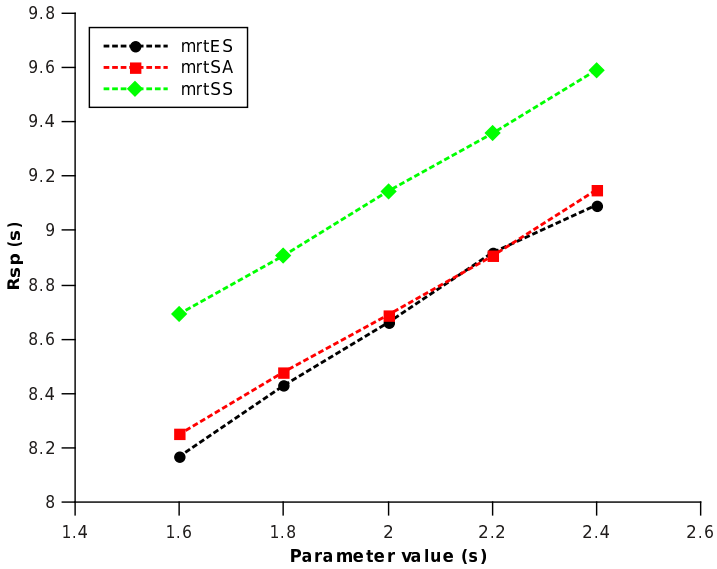


Figure 5.22: Impact of mrt_ES, mrt_SA, and mrt_SS on system response time

The sensitivity analysis approach that we proposed is tailored for hierarchical models and indicates effective points to be tuned in order to achieve even better performance in this scalable web service composition. Systems administrators can benefit of this approach, that may guide investments and help decision making on which are the high priority components during tune-up and upgrade efforts.

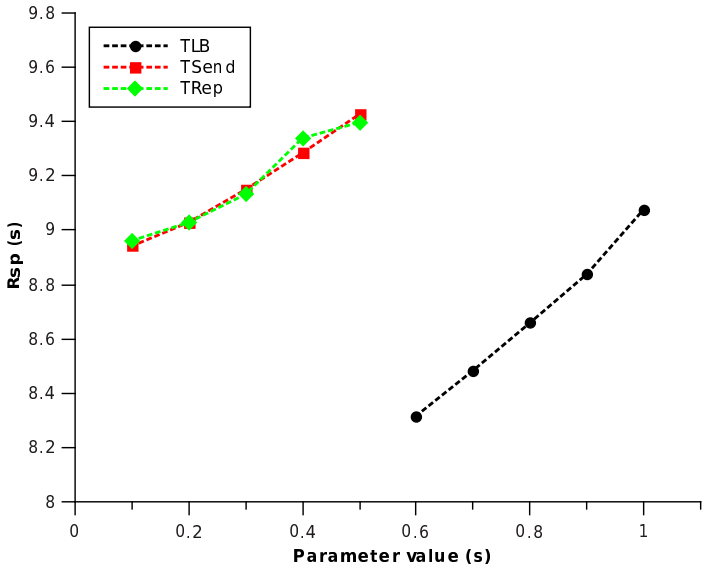


Figure 5.23: Impact of TLB, TSend, and TRep on system response time

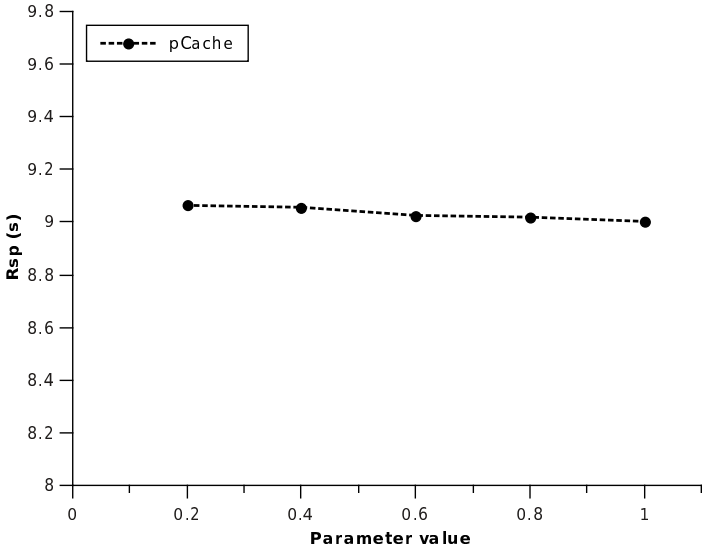


Figure 5.24: Impact of pCache on system response time

5.4 Optimization of composite web services with Sensitive GRASP

We carried out experiments for verifying the performance of Sensitive GRASP in a scenario of multiple possible providers for a composite web service. A CTMC model describes the web service composition (i.e., a mashup), and we compare the results with the approach presented in (MATOS; MACIEL; SILVA, 2013) – hereinafter called Non-sensitive GRASP for the sake of clarity. The execution time and the quality of solutions provided by each approach are the evaluation criteria.

The Event recommendation mashup evaluated here is almost identical to that presented in Section 5.3, with the exception that the mean response time of the **Top Event Selection** activity is not represented. This is because the event selection involves only internal processing, after receiving input data from the third-party providers. In this study, we try to select the optimal configuration of external providers.

Figure 5.25 depicts the continuous time Markov chain (CTMC) used in this study. Due to the similarity with Figure 5.21, we suppress the explanation of this model. The reader can find the description of states and transitions in Section 5.3.

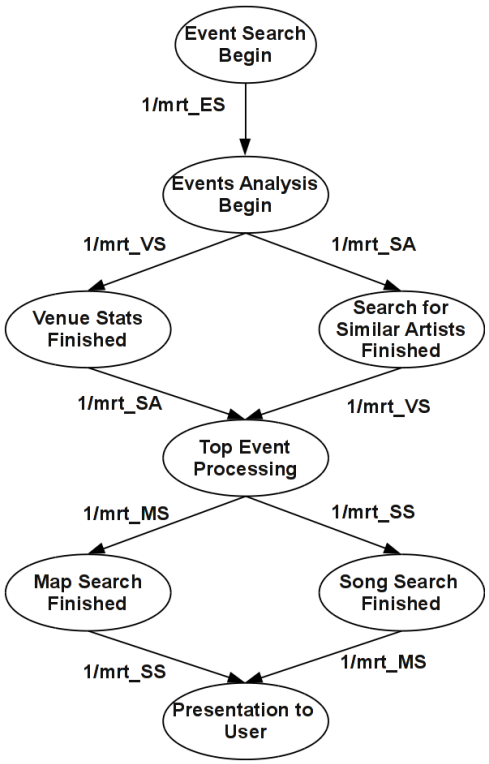


Figure 5.25: Event recommendation mashup

We implemented both algorithms, Sensitive and Non-sensitive GRASP, in Java. The tests were performed in a computer with Intel Core i7 3.0 GHz processor, with 12 GB RAM. The operating system was the Debian GNU/Linux 7.0, with kernel 3.14-1-amd64. The environment

Table 5.23: Parameters used in the benchmark with size 100^5

Provid.	Event search		Venue stats		Similar artists		Map Search		Song Search	
ID	Rel.	Rsp.	Rel.	Rsp.	Rel.	Rsp.	Rel.	Rsp.	Rel.	Rsp.
0	0.9670	3.0113	0.9837	3.5059	0.8597	1.0736	0.9268	4.0671	0.85892	1.3385
1	0.8778	2.7782	0.9311	4.6052	0.9763	3.3094	0.9929	3.4856	0.93159	4.3750
2	0.9434	2.1265	0.9810	4.3334	0.9285	4.6042	0.8835	3.4695	0.97944	3.0133
3	0.8902	3.1983	0.9357	1.9611	0.9366	4.8620	0.9857	1.3064	0.97163	4.7609
4	0.9103	3.7795	0.9652	2.6782	0.9866	4.2124	0.8885	3.7308	0.98738	2.9546
5	0.9312	3.9894	0.9744	4.5989	0.9208	4.1117	0.9056	1.2273	0.91222	1.5175
6	0.9541	3.8682	0.9549	3.7501	0.8729	2.4366	0.8936	3.5028	0.90240	2.8178
7	0.9148	1.5231	0.8522	2.0175	0.9880	3.5669	0.9702	1.7196	0.85149	1.3232
8	0.9378	3.2733	0.8625	3.2696	0.8998	2.8502	0.9222	4.3975	0.89676	4.7288
9	0.9549	1.6625	0.9350	4.5319	0.9742	2.3224	0.9830	4.9187	0.85726	1.9227
10	0.9837	3.5059	0.9268	4.0671	0.8663	1.6207	0.9300	3.9414	0.92036	3.7622
11	0.9311	4.6052	0.9929	3.4856	0.8798	3.8831	0.9386	1.9414	0.90350	1.9944
12	0.9810	4.3334	0.8835	3.4695	0.9830	2.5857	0.9669	2.6959	0.87086	1.8588
13	0.9357	1.9611	0.9857	1.3064	0.9179	4.4242	0.9579	2.1591	0.94919	1.2196
14	0.9652	2.6782	0.8885	3.7308	0.8909	4.7061	0.9737	1.5192	0.97035	2.8545
15	0.9744	4.5989	0.9056	1.2273	0.9123	3.6869	0.9397	2.3304	0.91226	1.5222
16	0.9549	3.7501	0.8936	3.5028	0.9122	1.7268	0.8708	1.9284	0.87014	4.3974
17	0.8522	2.0175	0.9702	1.7196	0.9034	1.0173	0.8654	1.4829	0.90527	2.1776
18	0.8625	3.2696	0.9222	4.3975	0.9104	3.4003	0.9208	1.3554	0.97126	4.7393
19	0.9350	4.5319	0.9830	4.9187	0.9501	1.1528	0.9664	2.3104	0.91387	3.5298

was controlled to avoid significant interferences to system performance. The Java version used was 1.7.0_51 by means of the OpenJDK Runtime Environment (IcedTea 2.4.6).

Generation of Benchmark

The benchmarks of the experiments comprised $||M||$ hypothetical providers for each service type, where $||M||$ varies from 100 to 500 depending on each specific experiment. A pseudo random number generator called MersenneTwister (MATSUMOTO; NISHIMURA, 1998) was used to generate providers parameters using a Uniform distribution with interval from [1s, 5s] for the response time and [0.85, 0.99] for the reliability. We chose a Uniform distribution to get the most diverse values for the parameters of each provider. The intervals of the generated numbers were based on real web service measurements and some values found in the literature (ZHENG et al., 2012; SATO; TRIVEDI, 2007). MersenneTwister was chosen because it has a very long period and it passes many tests for statistical randomness (L'ECUYER; SIMARD, 2007). Table 5.23 shows the values of reliability (Rel.) and response time (Rsp.) generated for 20 providers of the benchmark with size 100. For the sake of conciseness we do not show all providers' values here, but the complete benchmark data is available in a specific website containing the resources used in this study[1].

Reference solutions

For all generated benchmarks, we computed a reference solution S^* used as one of the stopping conditions of the optimization algorithm evaluated here. The procedure terminates when a solution S_i with $\delta_i \leq 0.05$ is found, or when a maximum number of GRASP iterations

[1]Resource files for this study: `http://www.cin.ufpe.br/~rsmj/sensitive-grasp/`

Table 5.24: Reference solutions used throughout experiments

Amount of providers	Reference solution		
	Rsp. time (s)	**Rel.**	θ
100	10.426	0.987	0.1281
200	7.894	0.985	0.1142
300	8.003	0.984	0.1254
400	7.376	0.986	0.1002
500	8.019	0.988	0.0945

(5000) is reached. δ_i is the relative difference between the cost of the reached solution, $\theta(S_i)$, and the cost of the reference solution, $\theta(S^*)$. Thus, $\delta_i \in \mathbb{R}$ and was computed as $\frac{\theta(S_i) - \theta(S^*)}{\theta(S^*)}$. For such a purpose, the reference solution becomes another input parameter for the Algorithm 3. When a solution S_i with $\delta_i \leq 0.05$ is not found within the defined maximum number of iterations, the algorithm provides the solution with lowest cost so far, which might be much worse than the reference solution, though.

The reference solution was found by preliminary execution of Non-sensitive GRASP configured for 5000 iterations, and executed 20 times with different seeds for random number generation. Notice that the reference solution is not necessarily the optimal solution for the benchmark. This procedure was used because an exhaustive search to determine the exact best solution would last several days due to the adopted benchmark sizes.

Table 5.24 shows the reference solutions computed for benchmarks with 100, 200, 300, 400, and 500 possible providers for each service. For each reference solution[2], Table 5.24 presents the respective response time (Rsp), reliability (Rel) and cost (θ) of the solution, computed through Equation (5.18).

$$\theta = (1 - Rel) \times Rsp \tag{5.18}$$

For the sake of simplicity, hereinafter the benchmark size will be denoted as N, instead of $||M||$, indicating the number of possible providers for each service. There are five types of service providers, so the total amount of possible solutions is N^5, with $N \in \{100, 200, 300, 400, 500\}$.

Experimental results

Non-sensitive GRASP and Sensitive GRASP were executed 50 times over each benchmark, so we could get enough data to compare the execution time and the quality of solutions provided by each approach. The list of adopted random generation seeds is available on the resources website. The parameters **MaxPool** and **MaxIt** of the approximate local search (see Algorithm 5) were set to 20 and 40, respectively, after some tests with different configurations for these parameters. The greediness parameter was set to 0.90 in Non-Sensitive GRASP, indicating that in the construction phase, only the best 10 % providers, according to response time, were considered. The Sensitive GRASP was evaluated in two distinct greediness levels: 0.90

[2]The specific configurations of providers which constitute the reference solutions are also available in `http://www.cin.ufpe.br/~rsmj/sensitive-grasp/`

and 0.95. These two levels were chosen in order to enable a comparison with Non-Sensitive GRASP, and on the other hand to asses the effect of the greediness level on the performance of Sensitive GRASP. Notice that with greediness 0.90 in a benchmark of 100 providers for each service, Sensitive GRASP randomly selects one among the best 10 providers for the most impacting service, indicated by the sensitivity analysis. When greediness is 0.95, the random selection occurs only from the best 5 providers. This might produce better initial solutions, speeding up convergence to the optimal solution, but it might also make the algorithm be trapped in a local optimum, still far from global optimum.

The sensitivity analysis of the CTMC model on Sensitive GRASP was configured to identify the impact of each service response time on the probability of reaching the *Complete* state in up to 5 seconds.

Figure 5.26 shows the average cost (θ^*) of the solutions provided by each algorithm for each benchmark size. The average cost is computed from the outputs of the 50 executions of each algorithm. We consider that the lower is θ^* the higher is the quality of the output produced by the algorithm. Sensitive GRASP, in both greediness levels, has θ^* values close to those of Non-sensitive GRASP, for almost all benchmark sizes. A small advantage for Sensitive GRASP is noticed in the largest benchmark sizes: 300, 400, and 500. A set of paired sample t-tests with confidence interval of 90% and significance level of 0.05 show that results of the three algorithms are not significantly different from one to each other. The only exception is the difference between Non-sensitive GRASP and Sensitive GRASP with greediness 0.90 for the benchmark size 300. That is the largest difference noticeable in Figure 5.26. Therefore, it is worth emphasizing that the increased greediness level did not produce significant improvements on quality of results from Sensitive GRASP.

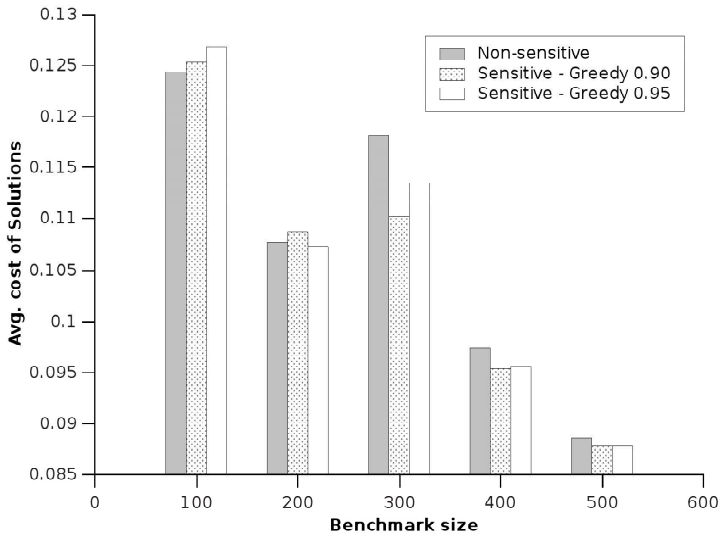


Figure 5.26: Average cost of solutions

We also analyzed the number of "hits" achieved by each algorithm, i.e., the number of solutions among the 50 executions which are better than or equal to the reference solution.

Table 5.25: Statistical summary of execution times

Amount of providers	Non-sensitive GRASP		Sensitive GRASP 0.90		Sensitive GRASP 0.95	
	Mean (s)	Std. Dev.	Mean (s)	Std. Dev.	Mean (s)	Std. Dev.
100	80.6	68.6	70.4	55.5	70.3	63.9
200	80.7	67.4	110.9	106.8	88.1	93.4
300	69.9	83.9	67.4	52.3	59.2	54.7
400	1,068.0	1,169.4	303.4	314.7	267.2	255.2
500	363.8	397.9	152.4	163.3	150.0	135.4

Figure 5.27 shows that the Sensitive GRASP with greediness 0.95 outperforms the Sensitive GRASP with greediness 0.90 and the Non-sensitive GRASP, considering benchmarks sizes 200, 300, 400, and 500. These results confirm that Sensitive GRASP is slightly better than Non-sensitive for finding the lowest cost solutions in the largest benchmarks. Notice that increasing the greediness parameter might be useful to reach more solutions that are better than those previously known.

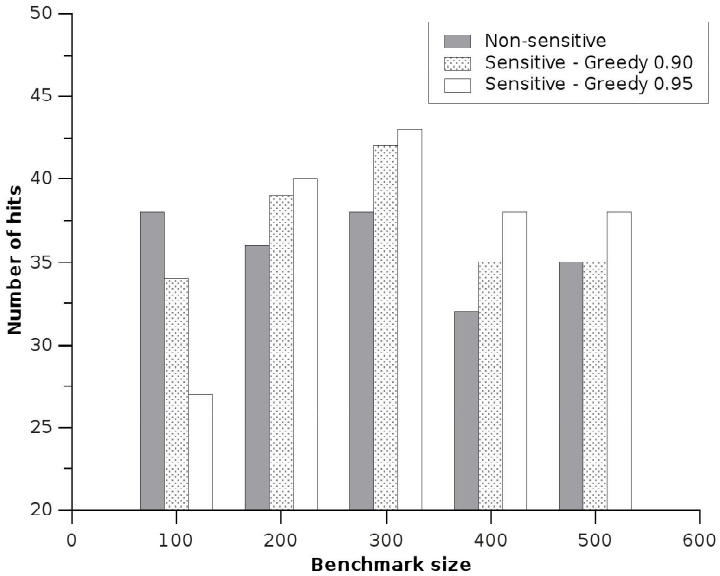


Figure 5.27: Number of hits

Besides the evaluation of quality of solutions, we also compared the execution time of the algorithms. Table 5.25 presents a statistical summary of the execution time. Sensitive GRASP (in its two variants) had shorter mean times than Non-sensitive for benchmark sizes 100, 300, 400, and 500. Especially for the two largest benchmark sizes, Sensitive GRASP is capable of obtaining better solutions than Non-Sensitive GRASP with smaller computational effort. Table 5.25 also shows that greediness 0.95 yields shorter mean execution time than greediness 0.90 for all benchmark sizes. The improvement on mean execution time is related to the increased ability of finding good solutions, so the algorithm required less iterations to reach the stopping criterion.

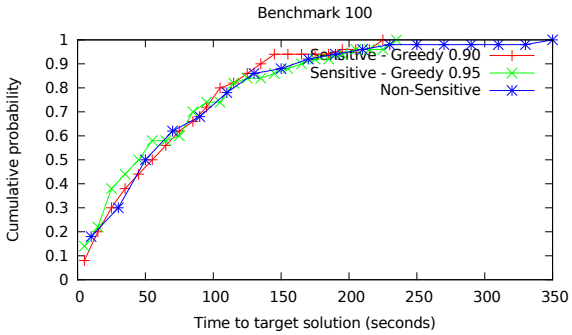
Since we notice large standard deviations on the results of Table 5.25, we performed additional analyses to determine how significant were the differences in execution times of the algorithms. A complementary view for the analysis is presented in Figures 5.28a, 5.28b, 5.28c,

5.28d, and 5.28e, which depicts the cumulative probability distribution of the execution time for the evaluated approaches, also known as time-to-target plot (AIEX; RESENDE; RIBEIRO, 2005). For benchmark sizes 100, 200, and 300, the curves of Non-sensitive, Sensitive with greediness 0.90, and Sensitive with greediness 0.95 are similar. When benchmark sizes 400 and 500 are analyzed, we notice that Sensitive GRASP is able to reach good solutions –i.e. close or better than the reference solution– in less time than Non-sensitive. In Figure 5.28d, both versions of Sensitive GRASP reach the target in up to 1500 seconds, whereas the Non-sensitive had a much poorer performance: it took up to 5500 seconds to find a suitable solution. In Figure 5.28e every execution of Sensitive GRASP spent up to 800 seconds, whereas a significant part of Non-sensitive GRASP executions took between 800 and 1800 seconds to reach a solution that satisfied the stopping criterion.

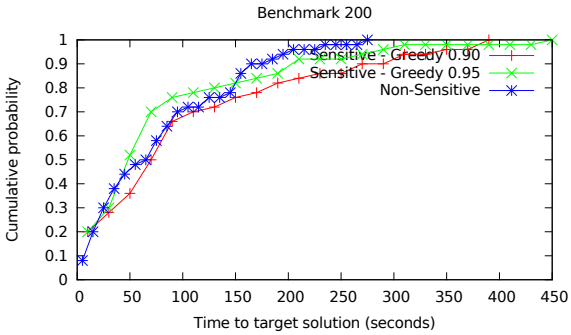
The time-to-target plots also provide interesting results on the impact of greediness parameter on the execution time of Sensitive GRASP. In general, the algorithm has similar execution time with 0.90 ("*Sensitive Greedy 0.95*" curve) and 0.95 ("*Sensitive Greedy 0.95*" curve) greediness levels, and we can observe small differences in favor of the latter one. On benchmark sizes 200 and 300, the curve "*Sensitive Greedy 0.95*" reaches higher probability values ($70\% - -80\%$) before "*Sensitive Greedy 0.90*" does. Considering those two benchmarks, the probability of sensitive GRASP completing in less than 100 seconds is larger with greediness 0.95 than with greediness 0.90. For benchmark sizes 400 and 500, it is worth to notice that the maximum execution time with greediness 0.95 is about 100 seconds shorter than with greediness 0.90.

It is also important to highlight that, in the two largest benchmarks, 70% of the executions took less than 300 seconds (5 minutes) to reach a suitable solution. This was observed in both greediness levels. For benchmark sizes 100 and 200, the solution is always achieved in less than 450 seconds. This denotes that Sensitive GRASP is also useful for dynamic compositions, where the combination of providers can be updated in runtime to satisfy a new optimal configuration every time the average performance of some provider changes.

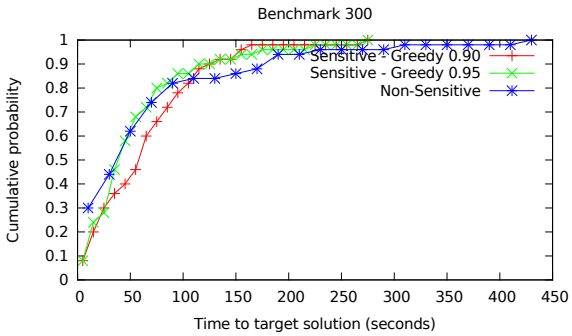
This study was based on CTMC, but other (single or hierarchical) analytical models may be used with specific changes in the integration with the optimization algorithm and sensitivity analysis tools.



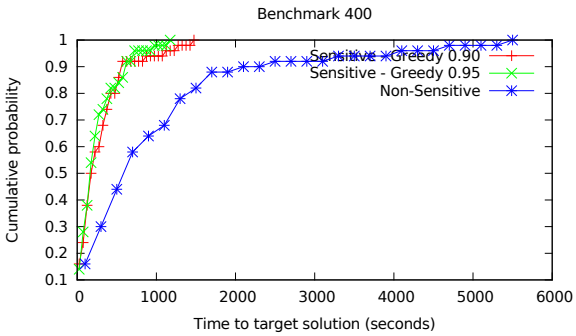
(a) Benchmark 100



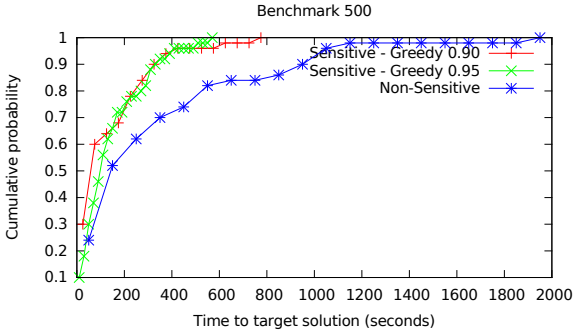
(b) Benchmark 200



(c) Benchmark 300



(d) Benchmark 400



(e) Benchmark 500

Figure 5.28: Distribution of execution time (time-to-target plots)

6

Final remarks

A large proportion of worldwide IT companies have adopted cloud computing for various purposes. Providers of public cloud services, as well as owners of private clouds need to design and manage their systems to keep up with users expectations regarding performance and dependability of their infrastructures. Analytical and simulation models are useful for planning computer systems and predicting their behavior before deployment or significant changes. Although the creation and analysis of performance and dependability models for cloud computing systems is a challenge that requires the combination of many techniques for reaching accurate and significant results. This thesis showed that hierarchical modeling is important and effective for coping with such a hard task. Moreover, it proposed methods for detecting the factors that have the largest importance to the improvement of a cloud system.

This Ph.D. research achieved a number of results in the areas that it has explored, and the major contributions are the methods for identifying performance and dependability bottlenecks of cloud computing systems. The supporting methodology also provides guidance for administrators of IaaS systems that intend to detect points for improvement in their infrastructures. The methods can also be applied in an integrated manner with the optimization process described in this thesis. The methods for bottleneck identification were tested throughout distinct case studies and produced other noteworthy results. The contributions of this thesis are summarized as follows.

6.1 Contributions

The hierarchical models for dependability evaluation of private clouds, as well as mobile clouds, allowed obtaining important conclusions regarding tune-up and architectural choices for those systems. Many research studies benefited from the proposed approach and have shown how useful is the methodology presented here.

The models for performance evaluation of a scalable composite web service enable capacity planning for that application, including aspects of the elasticity mechanisms of cloud systems such as autoscaling and load balancing features. The proposed hierarchical model is

adjustable, so the composite web service model might be replaced by other one to represent a different application running on top of the same cloud infrastructure.

This research introduced methods to build unified sensitivity rankings when RBD and CTMC models are combined, as well as for the composition of SPN and CTMC models. Such methods are a contribution to the state of the art, to the best of our knowledge. Hierarchical models created for various types of systems may also be analyzed using the proposed composition methods, because they are not restricted to the cloud computing domain. Even the supporting methodology is general enough to be adapted in distinct contexts, guiding the hierarchical modeling and bottleneck identification regardless the system under assessment.

The composition methods are embedded in Mercury software tool, providing an automated sensitivity analysis framework for hierarchical models. The features developed for Mercury in the scope of this thesis include: sensitivity analysis of RBD, SPN, and CTMC models; parameterization and assignment of rewards for CTMC; computation of mean time to absorption in CTMC and SPN; hierarchical modeling with RBD and CTMC, as well as with SPN and CTMC; sensitivity ranking computation for the hierarchical models. Such set of features allow users to create and analyze models following the methodology described in Section 4.1.

Other original contribution in this work is the Sensitive GRASP algorithm, that is targeted at optimizing performance and dependability of cloud-hosted services and their infrastructures. This algorithm may also be useful for other computing infrastructures that cannot stand the exploration of all architectural and configuration possibilities to find the best quality of service.

Significant effort was dedicated for reviewing the documentation of cloud computing platforms, and to properly set up testbed infrastructures used in the case studies. The experience acquired through the configuration of every component in such private clouds was essential for a detailed understanding of the systems evaluated here. Moreover, the employment of these infrastructures on correlated research projects generated other important results in fields such as: software aging and rejuvenation; and fault injection, which represented complimentary views of the cloud design and management issues addressed in this thesis.

This thesis did not tackled specific problems of public clouds, mainly because the access to information about their internal infrastructure is very limited, in comparison to private cloud platforms. But the methods developed here are not tied to the domain of private clouds, so they should also be useful for administrators of public clouds.

6.2 Future works

The knowledge about formal models for performance and dependability evaluation is not widespread among computer systems administrators, so auxiliary tools that convert semi-formal to formal models (ANDRADE et al., 2013) are helpful and could be integrated to the proposed methodology in future works. This would enable the representation of the cloud

systems by means of UML and similar modeling languages that are well known by software engineers and other people in the cloud infrastructure technical staff.

Other limitation of this work is not addressing models such as Queueing Networks, Stochastic Automata Networks, Fault Trees, and Reliability Graphs. The usage of sensitivity analysis techniques such as Regression, Correlation, and Perturbation Analysis is not taken into account in this thesis too. Future extensions might include sensitivity analysis methods that comprehend other modeling formalisms or different kinds of compositions for sensitivity indices.

The application of the proposed methods to scenarios with software-defined networks (SDN) is also a prominent possibility for future research, since SDNs are an important part of current datacenters. Analytical models for representing those infrastructures might be connected to the hierarchical models presented in the case studies here, with proper modifications. Such an integration should create expanded models to provide even more accurate results and verify the importance of some network issues that were not handled in any of the assessments this thesis (e.g., behavior of communication protocols, packets discard, and queuing issues on routing and switching equipment).

Other works might prospect improvements for the Sensitive GRASP algorithm and employment of other optimization meta-heuristics for the parameter-value assignment problem in analytical models. Genetic algorithms, particle swarm optimization, and ant colony algorithms are some options that might be assessed in such a direction.

References

ABDALLAH, H.; HAMZA, M. On the sensitivity analysis of the expected accumulated reward. **Performance Evaluation**, Amsterdam, The Netherlands, The Netherlands, v.47, n.2, p.163–179, 2002.

AIEX, R. M.; RESENDE, M. G. C.; RIBEIRO, C. C. **tttplots - a perl program to create time-to-target plots**. Available in `http://www2.research.att.com/~mgcr/tttplots/`. Accessed on 2012-11-13.

AMAZON. **Amazon Elastic Block Store (EBS)**. [S.l.]: Amazon.com, Inc., 2012. Available in: http://aws.amazon.com/ebs.

AMAZON. **Auto Scaling**. Available on `http://aws.amazon.com/autoscaling/`.

AMAZON. **Amazon EC2 SLA)**. Available in: http://aws.amazon.com/ec2/sla/, Amazon.com, Inc.

ANDRADE, E. C. et al. Openmads: an open source tool for modeling and analysis of distributed systems. In: **Computer Safety, Reliability, and Security**. [S.l.]: Springer, 2013. p.277–284.

APPBRAIN. **AppBrain Android Market**. Available on http://www.appbrain.com/. Accessed in 2013-07-26.

ARAUJO, J. et al. Software aging issues on the eucalyptus cloud computing infrastructure. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (SMC), 2011., Anchorage. **Proceedings...** [S.l.: s.n.], 2011. p.1411–1416.

ARMBRUST, M. et al. **Above the clouds**: a berkeley view of cloud computing. [S.l.: s.n.], 2009.

ARMBRUST, M. et al. A view of cloud computing. **Commun. ACM**, New York, NY, USA, v.53, n.4, p.50–58, Apr 2010.

AVIZIENIS, A. et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. **IEEE Trans. Dependable Sec. Comput.**, [S.l.], v.1, n.1, p.11–33, 2004.

BALASUBRAMANIAN, N.; BALASUBRAMANIAN, A.; VENKATARAMANI, A. Energy Consumption in Mobile Phones: a measurement study and implications for network applications. In: ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT CONFERENCE, 9., New York, NY, USA. **Proceedings...** ACM, 2009. p.280–293. (IMC '09).

BARHAM, P. et al. Xen and the art of virtualization. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.37, p.164–177, oct 2003.

BLAKE, J. T.; REIBMAN, A. L.; TRIVEDI, K. S. Sensitivity analysis of reliability and performability measures for multiprocessor systems. In: SIGMETRICS '88: PROCEEDINGS OF THE 1988 ACM SIGMETRICS CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, New York, NY, USA. **Anais...** ACM, 1988. p.177–186.

BOLCH, G. et al. **Queuing Networks and Markov Chains**: modeling and performance evaluation with computer science applications. 2.ed. [S.l.]: John Wiley and Sons, 2001.

BONDAVALLI, A.; MURA, I.; TRIVEDI, K. S. Dependability Modelling and Sensitivity Analysis of Scheduled Maintenance Systems. In: THIRD EUROPEAN DEPENDABLE COMPUTING CONFERENCE ON DEPENDABLE COMPUTING, London, UK. **Proceedings...** Springer-Verlag, 1999. p.7–23. (EDCC-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**, [S.l.], v.25, n.6, p.599–616, 2009.

CALLOU, G. et al. Sustainability and dependability evaluation on data center architectures. In: IEEE INT. CONF. ON SYSTEMS, MAN, AND CYBERNETICS (SMC), 2011. **Proceedings...** [S.l.: s.n.], 2011. p.398–403.

CAMPOS, E. et al. Performance Evaluation of Virtual Machines Instantiation in a Private Cloud. In: IEEE WORLD CONGRESS ON SERVICES. SERVICES 2015, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.319–326.

CAO, X.-R. Uniformization and performance sensitivity estimation in closed queueing networks. **Mathematical and Computer Modelling**, [S.l.], v.23, n.11-12, p.77 – 92, 1996. Elsevier.

CHAISIRI, S.; LEE, B.-S.; NIYATO, D. Optimization of resource provisioning cost in cloud computing. **Services Computing, IEEE Transactions on**, [S.l.], v.5, n.2, p.164–177, 2012.

CHOI, H.; MAINKAR, V.; TRIVEDI, K. S. Sensitivity Analysis of Deterministic and Stochastic Petri Nets. In: INTERNATIONAL WORKSHOP ON MODELING, ANALYSIS, AND SIMULATION ON COMPUTER AND TELECOMMUNICATION SYSTEMS, San Diego, CA, USA. **Proceedings...** Society for Computer Simulation International, 1993. p.271–276. (MASCOTS '93).

CHUOB, S.; POKHAREL, M.; PARK, J. S. Modeling and Analysis of Cloud Computing Availability Based on Eucalyptus Platform for E-Government Data Center. In: FIFTH INTERNATIONAL CONFERENCE ON INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 2011. **Anais...** [S.l.: s.n.], 2011. p.289 –296.

CIARDO, G. et al. Automated generation and analysis of Markov reward models using stochastic reward nets. In: MEYER, C.; PLEMMONS, R. (Ed.). **Linear Algebra, Markov Chains and Queuing Models**. [S.l.]: Springer, 1993. v.48, p.145–191.

CLOTH, L. et al. Model Checking Markov Reward Models with Impulse Rewards. In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS, 2005., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2005. p.722–731. (DSN '05).

CLOUDSTACK. **Apache CloudStack**: open source cloud computing. Available on `https://cloudstack.apache.org/`. Accessed in 2016-01-25.

COOPER, T.; FARRELL, R. Value-chain engineering of a tower-top cellular base station system. In: VEHICULAR TECHNOLOGY CONFERENCE, 2007. VTC2007-SPRING. IEEE 65TH. **Anais...** [S.l.: s.n.], 2007. p.3184–3188.

CUOMO, A. et al. An SLA-based Broker for Cloud Infrastructures. **Journal of Grid Computing**, [S.l.], v.11, n.1, p.1–25, 2013.

D-LINK Wireless N150 Router. Available on `http://www.dlink.com/us/en/home-solutions/connect/routers/dir-601-wireless-n-150-home-router`.

DANTAS, J. et al. An Availability Model for Eucalyptus Platform: an analysis of warm-standy replication mechanism. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (IEEE SMC 2012), 2012., Seoul, Korea. **Proceedings...** [S.l.: s.n.], 2012.

DANTAS, J. et al. Models for Dependability Analysis of Cloud Computing Architectures for Eucalyptus Platform. **International Transactions on Systems Science and Applications**, [S.l.], v.8, p.13–25, Dec 2012.

DANTAS, J. et al. Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud. **Computing**, [S.l.], p.1–20, 2015.

DOWNING, D.; GARDNER, R.; HOFFMAN, F. An Examination of Response-Surface Methodologies for Uncertainty Analysis in Assessment Models. **Technometrics**, [S.l.], v.27, n.2, p.151–163, May 1985.

EUCALYPTUS. **Eucalyptus Open-Source Cloud Computing Infrastructure - An Overview**. Goleta, CA: Eucalyptus Systems, Inc., 2009.

EUCALYPTUS. **Cloud Computing and Open Source**: IT Climatology is born. Goleta, CA: Eucalyptus Systems, Inc., 2010.

EUCALYPTUS. **Official Documentation for Eucalyptus Cloud**. Available on `https://www.eucalyptus.com/docs/eucalyptus/4.0/`.

EUCALYPTUS. **CloudWatch Troubleshooting**. Available on `https://github.com/eucalyptus/eucalyptus/wiki/CloudWatch-Troubleshooting/`.

EUCALYPTUS. **HPE Helion Eucalyptus**. [S.l.]: Hewlett Packard Enterprise, 2016. Available in: http://www.eucalyptus.com/.

FEO, T.; RESENDE, M. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, [S.l.], v.8, p.67–71, 1989.

FEO, T.; RESENDE, M. Greedy randomized adaptive search procedures. **J. of Global Optimization**, [S.l.], v.6, p.109–133, 1995.

FESTA, P.; RESENDE, M. **Essays and Surveys on Metaheuristics**. [S.l.]: Kluwer Academic Publishers, 2002. v.6, p.325–367.

FRANK, P. M. **Introduction to System Sensitivity Theory**. [S.l.]: Academic Press Inc, 1978.

FURHT, B.; ESCALANTE, A. (Ed.). **Handbook of Cloud Computing**. [S.l.]: Springer Science & Business Media, 2010.

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

GERMAN, R.; MITZLAFF, J. Transient Analysis of Deterministic and Stochastic Petri Nets with TimeNET. In: LECTURE NOTES IN COMPUTER SCIENCE VOL. 977: QUANTITATIVE EVALUATION OF COMPUTING AND COMMUNICATION SYSTEMS. **Anais...** [S.l.: s.n.], 1995. p.209–223.

GHOSH, R. et al. End-to-end performability analysis for infrastructure-as-a-service cloud: an interacting stochastic models approach. In: DEPENDABLE COMPUTING (PRDC), 2010 IEEE 16TH PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2010. p.125–132.

GHOSH, R. et al. Modeling and performance analysis of large scale iaas clouds. **Future Generation Computer Systems**, [S.l.], v.29, n.5, p.1216–1234, 2013.

GINAC. **GiNaC is Not a CAS**. Available in: `http://www.ginac.de`.

GOOGLE. **Google App Engine**: platform as a service. Available on `https://cloud.google.com/appengine/`. Accessed in 2016-01-28.

HAMBY, D. M. A Review Of Techniques For Parameter Sensitivity Analysis Of Environmental Models. **Environmental Monitoring and Assessment**, [S.l.], p.135–154, 1994.

HAVERKORT, B.; MEEUWISSEN, A. Sensitivity and uncertainty analysis of Markov-reward models. **IEEE Transactions on Reliability**, [S.l.], v.44, n.1, p.147–154, Mar. 1995.

HAVERKORT, B. R. **Markovian models for performance and dependability evaluation**. New York, NY, USA: Springer-Verlag New York, Inc., 2002. p.38–83.

HEARTBEAT. **Linux-HA Project**. Available: http://www.linux-ha.org.

HEIDELBERGER, P.; GOYAL, A. Sensitivity Analysis of Continuous Time Markov Chains Using Uniformization. In: INTERNATIONAL WORKSHOP ON APPLIED MATHEMATICS AND PERFORMANCE/RELIABILITY MODELS OF COMPUTER/COMMUNICATION SYSTEMS, 2. **Proceedings...** [S.l.: s.n.], 1987. p.93–104.

HIREL, C.; TU, B.; TRIVEDI, K. S. **SPNP**: Stochastic Petri Nets. version 6.0. 2010.

HOFFMAN, F.; GARDNER, R. Evaluation of Uncertainties in Environmental Radiological Assessment Models. In: TILL, J.; MEYER, H. (Ed.). **Radiological Assessments**: a textbook on environmental dose assessment. Washington, DC: U.S. Nuclear Regulatory Commission, 1983. Report No. NUREG/CR-3332.

HU, T. et al. MTTF of Composite Web Services. In: INT. SYMP. ON PARALLEL AND DISTRIBUTED PROCESSING WITH APPLICATIONS (ISPA), 2010. **Anais...** [S.l.: s.n.], 2010. p.130 –137.

IOSUP, A. et al. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. **Parallel and Distributed Systems, IEEE Transactions on**, [S.l.], v.22, n.6, p.931–945, June 2011.

JAIN, R. **The Art of Computer Systems Performance Analysis**: techniques for experimental design, measurement, simulation and modeling. New York: Wiley-Interscience, 1991.

JOHNSON, D. et al. **Eucalyptus Beginner's Guide – UEC Edition**. 2.ed. [S.l.: s.n.], 2010.

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. Availability Modeling and Analysis of a Virtualized System. In: IEEE PACIFIC RIM INT. SYMP. ON DEPENDABLE COMPUTING. PRDC '09., 15. **Anais...** [S.l.: s.n.], 2009. p.365–371.

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

KLEINROCK, L. **Queueing Systems**. New York: Wiley, 1975. v.1.

KOLMOGOROV, A. Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung (in German). **Mathematische Annalen**, [S.l.], 1931. Springer-Verlag.

KOOPMAN, S. **The mobile cloud computing market will generate 45 billion dollars in revenues by 2016 | ASD Reports**. Acessible on `https://www.asdreports.com/news.asp?pr_id=200`.

KUO, W.; ZUO, M. **Optimal reliability modeling**: principles and applications. [S.l.]: John Wiley & Sons, 2003.

L'ECUYER, P.; SIMARD, R. TestU01: a C library for empirical testing of random number generators. **ACM Trans. Math. Softw.**, New York, NY, USA, v.33, n.4, Aug. 2007.

LIU, Z.; NAIN, P. Sensitivity results in open, closed and mixed product form queueing networks. **Performance Evaluation**, [S.l.], v.13, n.4, p.237 – 251, 1991.

LONGO, F. et al. A scalable availability model for Infrastructure-as-a-Service cloud. In: IEEE/IFIP 41ST INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS NETWORKS (DSN), 2011. **Proceedings...** [S.l.: s.n.], 2011. p.335 –346.

MA, Y.; HAN, J.; TRIVEDI, K. Composite performance and availability analysis of wireless communication networks. **IEEE Transactions on Vehicular Technology**, [S.l.], v.50, n.5, p.1216–1223, Sept. 2001.

MACIEL, P. et al. Dependability Modeling. In: **Performance and Dependability in Service Computing**: concepts, techniques and research directions. Hershey: IGI Global, 2011.

MALHOTRA, M.; TRIVEDI, K. S. Power-Hierarchy of Dependability-Model Types. **IEEE Trans. Reliability**, [S.l.], v.43, n.3, p.493–502, Sept. 1994.

MARIE, R. A.; REIBMAN, A. L.; TRIVEDI, K. S. Transient analysis of acyclic markov chains. **Performance Evaluation**, [S.l.], v.7, n.3, p.175 – 194, 1987.

MARINO, S. et al. A Methodology For Performing Global Uncertainty And Sensitivity Analysis In Systems Biology. **J Theor Biol**, [S.l.], Sept 2008.

MARSAN, M. A.; CONTE, G.; BALBO, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. **ACM Trans. Comput. Syst.**, New York, NY, USA, v.2, p.93–122, May 1984.

MARWAH, M. et al. Quantifying the sustainability impact of data center availability. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.37, p.64–68, March 2010.

MATEUS, G.; SILVA, R.; RESENDE, M. GRASP with path-relinking for the generalized quadratic assignment problem. **J. of Heuristics**, [S.l.], v.17, p.527–565, 2011.

MATOS JÚNIOR, R. d. S. **An automated approach for systems performance and dependability improvement through sensitivity analysis of Markov chains**. 2011. Dissertação (Mestrado em Ciência da Computação) — Universidade Federal de Pernambuco – UFPE, Recife, Brazil.

MATOS JUNIOR, R. et al. Sensitivity Analysis of Availability of Redundancy in Computer Networks. In: THE FOURTH INT. CONF. ON COMMUNICATION THEORY, RELIABILITY, AND QUALITY OF SERVICE (CTRQ 2011), Budapest, Hungary. **Proceedings...** [S.l.: s.n.], 2011. p.115–121.

MATOS, R. S.; MACIEL, P. R.; SILVA, R. M. QoS-driven optimisation of composite web services: an approach based on grasp and analytical models. **International Journal of Web and Grid Services**, [S.l.], v.9, n.3, p.304–321, 2013.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. **ACM Trans. Model. Comput. Simul.**, New York, NY, USA, v.8, n.1, p.3–30, Jan. 1998.

MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. [S.l.]: NIST - National Institute of Standards and Technology, 2011. NIST Special Publication 800-145.

MENASCÉ, D. A.; ALMEIDA, V. A.; DOWDY, L. W. **Performance by Design**: computer capacity planning by example. [S.l.]: Prentice Hall PTR, 2004.

MENDEZ MUNOZ, V. et al. Rafhyc: an architecture for constructing resilient services on federated hybrid clouds. **Journal of Grid Computing**, [S.l.], v.11, n.4, p.753–770, 2013.

MERCURY. **Mercury Tool**. Available in: `https://sites.google.com/site/mercurytooldownload/`, MoDCS Research Group.

MICROSOFT. **What is PaaS? Platform as a service | Microsoft Azure**. Available on `https://azure.microsoft.com/en-us/overview/what-is-paas/`. Accessed in 2016-01-20.

MICROSOFT. **Microsoft Azure**: cloud computing platform and services. Available on `https://azure.microsoft.com/en-us/documentation/scenarios/web-app/`. Accessed in 2016-01-28.

MOLLOY, M. K. Performance Analysis Using Stochastic Petri Nets. **IEEE Trans. Comput.**, Washington, DC, USA, v.31, n.9, p.913–917, Sept. 1982.

MUPPALA, J. K.; TRIVEDI, K. S. GSPN models: sensitivity analysis and applications. In: ACM-SE 28: PROCEEDINGS OF THE 28TH ANNUAL SOUTHEAST REGIONAL CONFERENCE, New York, NY, USA. **Anais...** ACM, 1990. p.25–33.

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

O'CONNOR, P. P.; KLEYNER, A. **Practical Reliability Engineering**. 5th.ed. [S.l.]: Wiley Publishing, 2012.

OLIVEIRA, D. et al. Availability and Energy Consumption Analysis of Mobile Cloud Environments. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS (IEEE SMC 2013), Manchester. **Proceedings...** [S.l.: s.n.], 2013.

OPDAHL, A. L. Sensitivity analysis of combined software and hardware performance models: open queueing networks. **Performance Evaluation**, [S.l.], v.22, n.1, p.75 – 92, 1995. 6th International Conference on Modelling Techniques and Tools for Computer Performance Evalution.

OPENNEBULA. **OpenNebula | Flexible Enterprise Cloud Made Simple**. Available on `http://opennebula.org/`. Accessed in 2016-01-20.

OPENSTACK. **OpenStack Open Source Cloud Computing Software**. Available on `https://www.openstack.org/`. Accessed in 2016-01-25.

OU, Y.; DUGAN, J. B. Approximate Sensitivity Analysis for Acyclic Markov Reliability Models. **IEEE Transactions on Reliability**, [S.l.], n.2, June 2003.

PENG, J. et al. Comparison of Several Cloud Computing Platforms. In: INT. SYMP. ON INFORMATION SCIENCE AND ENGINEERING (ISISE), 2., Shanghai. **Proceedings...** IEEE Press, 2009. p.23–27.

PHONEARENA. **Phone Arena - Phone News, Reviews and Specs**. Online. Available on `http://www.phonearena.com/`. Acessed 20-May-2013.

QI, H.; GANI, A. Research on mobile cloud computing: review, trend and perspectives. In: DIGITAL INFORMATION AND COMMUNICATION TECHNOLOGY AND IT'S APPLICATIONS (DICTAP), 2012 SECOND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.195–202.

R-PROJECT. **The R Project for Statistical Computing**. Available in: `https://www.r-project.org`, Home page.

RESENDE, M.; RIBEIRO, C. **Handbook of Metaheuristics**. [S.l.]: Kluwer Academic Publishers, 2003. v.6, p.219–249.

RIGHTSCALE. **RightScale 2015 - State of the Cloud Report**. Available on http://assets.rightscale.com/uploads/pdfs/RightScale-2015-State-of-the-Cloud-Report.pdf. Acessed on 2016-02-05.

RIMAL, B. et al. Architectural Requirements for Cloud Computing Systems: an enterprise cloud approach. **Journal of Grid Computing**, [S.l.], v.9, n.1, p.3–26, 2011.

ROSS, S. **Introductory Statistics**. [S.l.]: Elsevier Science, 2010.

SAHNER, R. A.; TRIVEDI, K. S.; PULIAFITO, A. **Performance and reliability analysis of computer systems**: an example-based approach using the sharpe software package. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

SATO, N.; TRIVEDI, K. S. Stochastic Modeling of Composite Web Services for Closed-Form Analysis of Their Performance and Reliability Bottlenecks. In: ICSOC. **Anais...** [S.l.: s.n.], 2007. p.107–118.

SCHROEDER, B.; GIBSON, G. A. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In: USENIX CONFERENCE ON FILE AND STORAGE TECHNOLOGIES, 5., Berkeley. **Proceedings...** [S.l.: s.n.], 2007. (FAST '07).

SEMPOLINSKI, P.; THAIN, D. A comparison and critique of eucalyptus, opennebula and nimbus. In: CLOUD COMPUTING TECHNOLOGY AND SCIENCE (CLOUDCOM), 2010 IEEE SECOND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.417–426.

SILVA, B. et al. Mercury: an integrated environment for performance and dependability evaluation of general systems. In: INDUSTRIAL TRACK AT 45TH DEPENDABLE SYSTEMS AND NETWORKS CONFERENCE, Rio de Janeiro. **Proceedings...** IEEE, 2015. (DSN 2015).

SOUSA, E.; MACIEL, P.; ARAUJO, C. Performability evaluation of EFT systems using expolinomial stochastic models. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2009., Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2009. p.3328–3333. (SMC'09).

SQUARETRADE. **Cell Phone Comparison Study Nov 10**. Online. Accessed 20-May-2013, `http://www.squaretrade.com/cell-phone-comparison-study-nov-10`.

SUN. **Introduction to Cloud Computing Architecture**. [S.l.]: Sun Microsystems, Inc., 2009. 1 ed.

SUN, D. et al. A Dependability Model to Enhance Security of Cloud Environment using System Level Virtualization Techniques. In: FIRST INT. CONF. ON PERVASIVE COMPUTING, SIGNAL PROCESSING AND APPLICATIONS (PCSPA 2010), Harbin. **Proceedings...** [S.l.: s.n.], 2010.

SYMJA. **Symja**: Java Computer Algebra Library. Available in: `https://bitbucket.org/axelclk/symja_android_library`.

TRIVEDI, K. S. **Probability and Statistics with Reliability, Queuing, and Computer Science Applications**. 2.ed. [S.l.]: John Wiley and Sons, 2001.

TRIVEDI, K. S.; SAHNER, R. SHARPE at the age of twenty two. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.36, n.4, p.52–57, 2009.

VON LASZEWSKI, G. et al. Comparison of multiple cloud frameworks. In: CLOUD COMPUTING (CLOUD), 2012 IEEE 5TH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.734–741.

VOORSLUYS, W. et al. Cost of Virtual Machine Live Migration in Clouds. **Lecture Notes in Computer Science – Springer-Verlag**, [S.l.], 2009.

WANG, D.; TRIVEDI, K. Computing steady-state mean time to failure for non-coherent repairable systems. **Reliability, IEEE Transactions on**, [S.l.], v.54, n.3, p.506–516, 2005.

WATSON J.F., I.; DESROCHERS, A. Applying generalized stochastic Petri nets to manufacturing systems containing nonexponential transition functions. **Systems, Man and Cybernetics, IEEE Transactions on**, [S.l.], v.21, n.5, p.1008–1017, Sep 1991.

WEI, B.; LIN, C.; KONG, X. Dependability Modeling and Analysis for the Virtual Data Center of Cloud Computing. In: IEEE INTERNATIONAL CONFERENCE ON HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS, 2011., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p.784–789. (HPCC '11).

WOLFRAM. **Wolfram Mathematica**: modern technical computing. Available in: `https://www.wolfram.com/mathematica`, Home page.

YIN, B. et al. Sensitivity analysis and estimates of the performance for M/G/1 queueing systems. **Perform. Eval.**, Amsterdam, The Netherlands, The Netherlands, v.64, n.4, p.347–356, 2007.

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

ZHENG, Z. et al. Collaborative Web Service QoS Prediction via Neighborhood Integrated Matrix Factorization. **Services Computing, IEEE Transactions on**, [S.l.], v.PP, n.99, p.1, 2012.

Appendix

A

Development of Mercury Tool Features

The sensitivity analysis methods proposed in this thesis were integrated in the Mercury tool. Mercury is a software developed by MoDCS (Modeling of Distributed and Concurrent Systems) Group at the Federal University of Pernambuco (UFPE), Brazil. The tool has been developed to evaluate performance, dependability, and energy flow models. It provides graphical interfaces for modeling and evaluating Stochastic Petri Nets (SPN), Reliability Block Diagrams (RBD), Energy Flow Models (EFM) and Continuous Time Markov Chains (CTMC). Figure A.1 illustrates the formalisms and evaluation methods available in Mercury.

The Algorithm 1 was used to produce the feature of Mercury tool depicted in Figure A.2. That dialog window shows a sensitivity analysis of a hierarchical model comprising an RBD and a CTMC. It first presents the partial derivatives expressions that denote the sensitivity of system availability to each parameter. The window also presents the numerical sensitivity indices for each parameter. In that case, **MTTFb1** and **MTTRb1** are parameters from the RBD model (corresponding to block b1 MTTF and MTTR, respectively), whereas **mu** and **lambda** are parameters from the CTMC model.

Notice that a user may choose between computing scaled sensitivity indices or unscaled ones. The ranking might also be presented ordered or not. If the user decides by ordering the

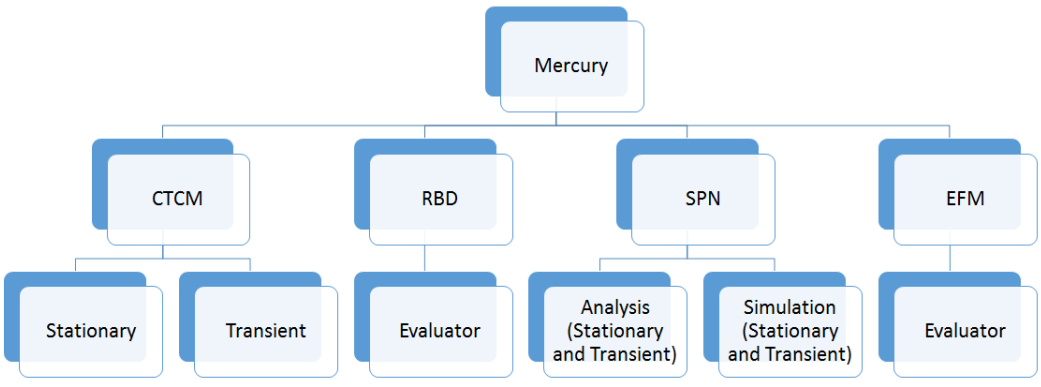


Figure A.1: Overview of Mercury features

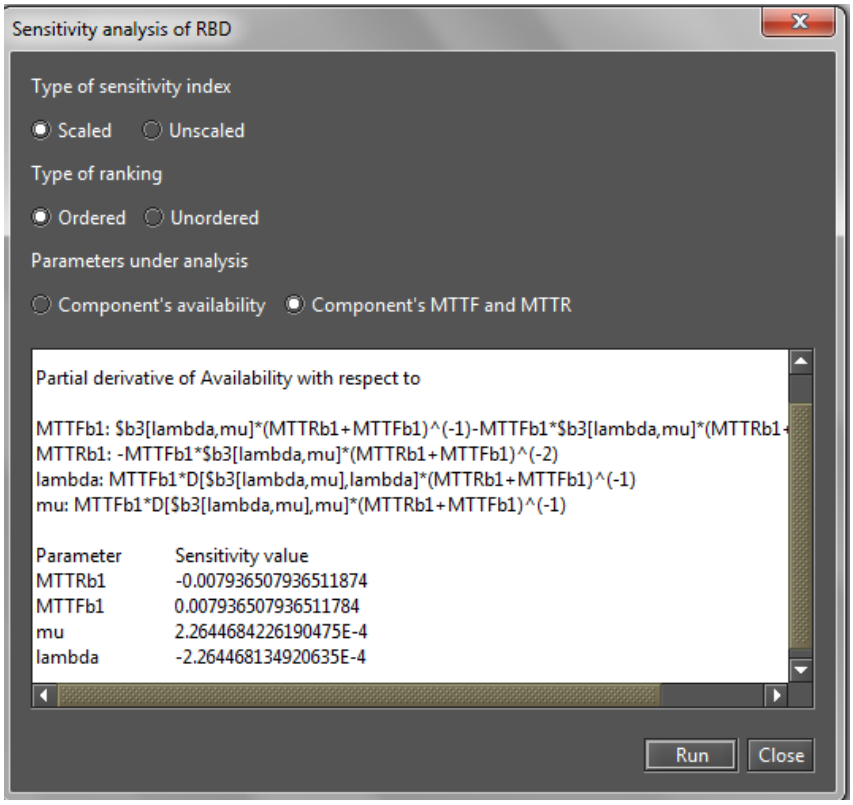


Figure A.2: Dialog window for RBD sensitivity analysis on Mercury

ranking, the absolute values of the indices are used for sorting them in decreasing order. There is also an option for considering the availability of each block as the target parameters in the sensitivity analysis, of their MTTF and MTTR values.

In Mercury source code, five classes interact to accomplish the proposed sensitivity analysis steps in hierarchical composition of RBD and CTMCs. Figure A.3 shows a UML sequence diagram that describes the execution of methods for that activity. The user triggers the sensitivity analysis in **JDialogSensitivityRBD**, that is the class of the dialog window shown in Figure A.2. The **RBDModel** class is responsible for returning the first three pieces of information for this analysis: the list of parameters, the structural function that represents the RBD, and the transformed structural function. This corresponds to lines from 1 to 6 of the proposed algorithm. Next, the class **SensitivityAnalysisRBD** provides the symbolic derivative function for each parameter. The class **SensitivityAnalysisCTMC** is responsible for computing the sensitivity indices for each CTMC sub-model. Those indices and the RBD parameter values are then used by **SensitivityAnalysisRBD** class to evaluate the former symbolic derivative expressions, yielding the numeric sensitivity indices. The method *sort()* from the **Collections** class is called when an ordered ranking is requested, finishing the procedure.

It is important to mention that intermediate steps are suppressed here because they are not the exclusive focus of this work. Some of those steps include the symbolic computation methods for obtaining partial derivatives and the symbolic-to-numeric conversion of expressions. The Symja framework provided many methods for implementing such tasks.

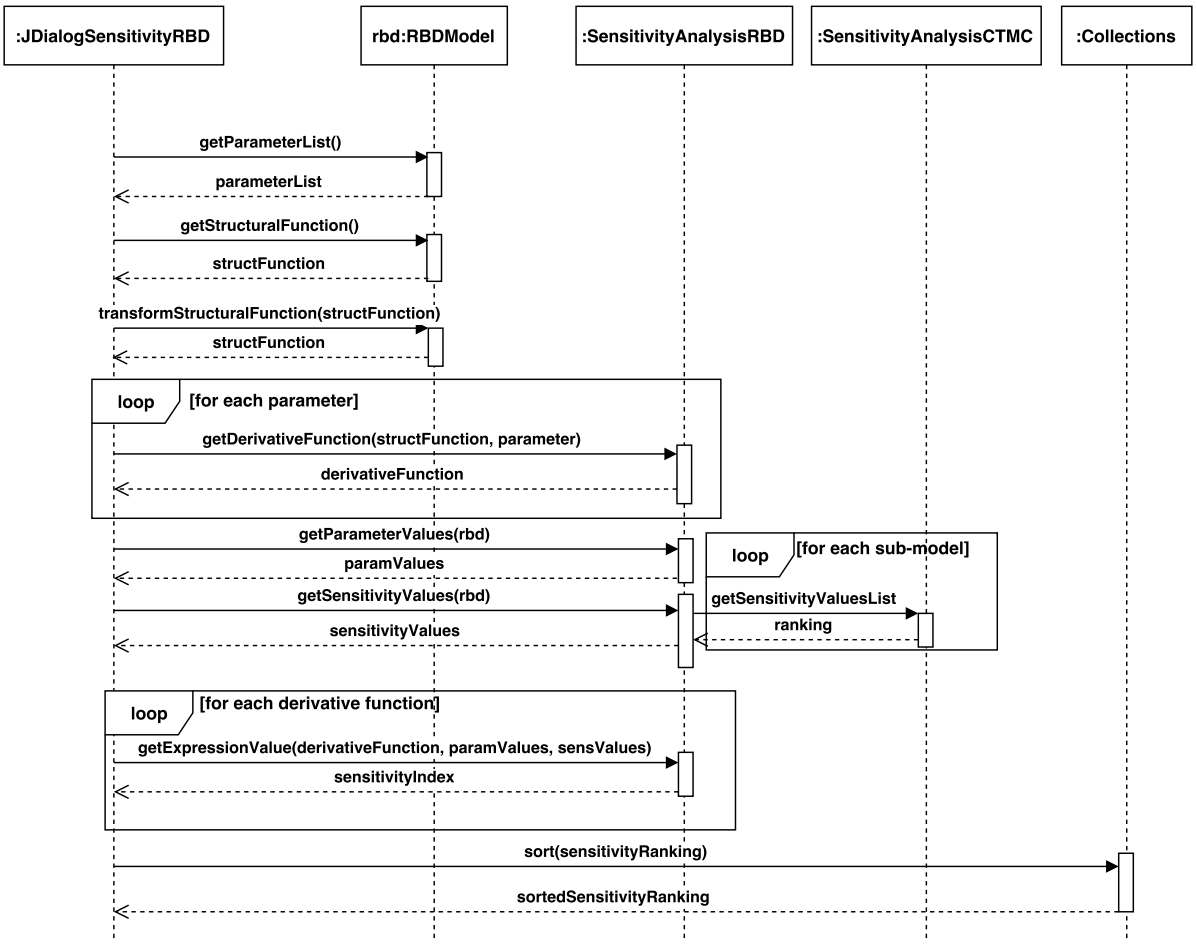


Figure A.3: UML sequence diagram for sensitivity computation with RBD main model and CTMC sub-models

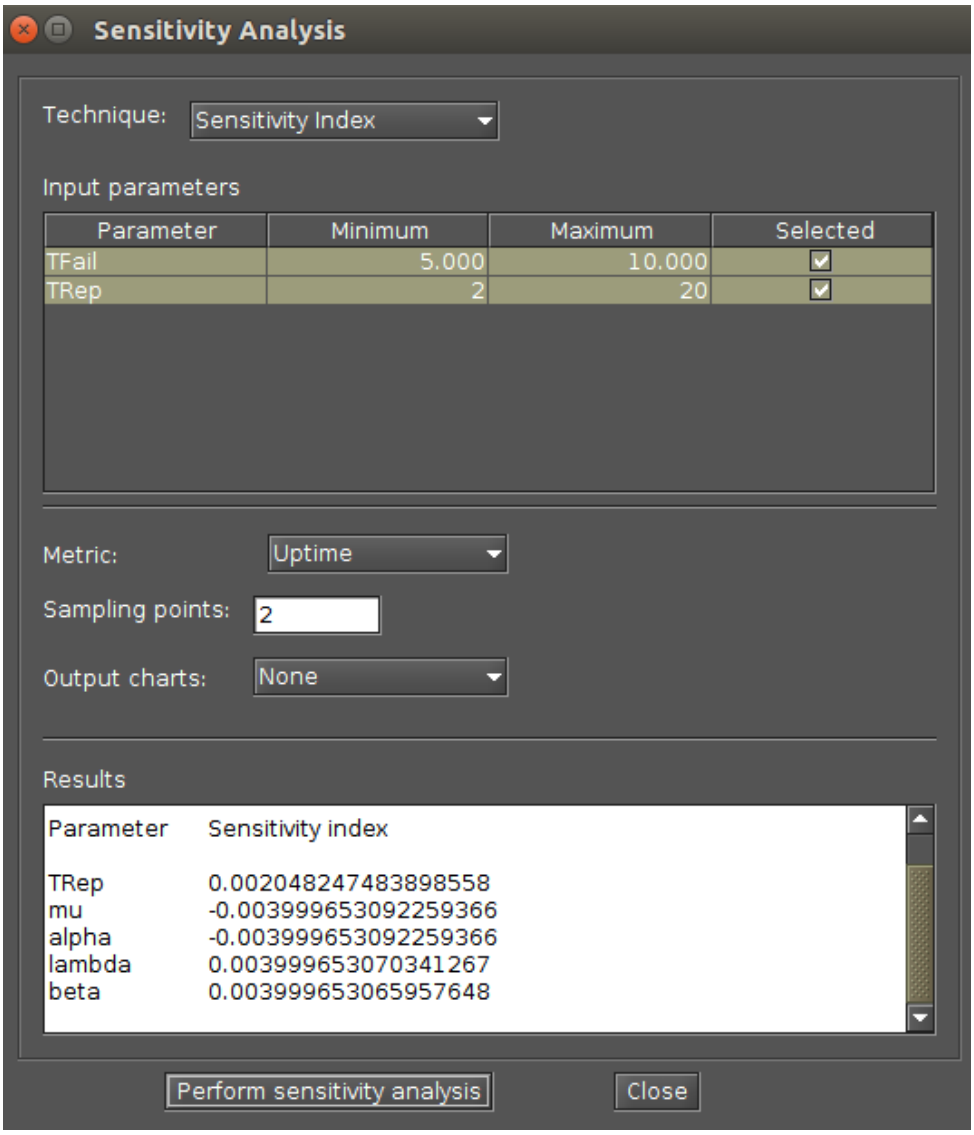


Figure A.4: Dialog window for SPN sensitivity analysis on Mercury

The Algorithm 2 provided the basis for development of sensitivity analysis when an SPN is the top-level model. Figure A.4 shows the dialog window for that feature. Notice that the user enters the minimum and maximum bounds for the range of each transition of the SPN top-level model: **TFail** and **TRep**. The number of sampling points will determine how many values will compose the range for each parameter. If the user enters only two sampling points, they will correspond to the maximum and minimum values specified. Moreover, the measure of interest is informed, since the SPN can be used to compute many kinds of metrics.

The results for the sensitivity analysis of the hierarchical model are presented in the text area on the bottom of the window. It shows the sensitivity indices for the transition TRep, and for parameters **mu**, **alpha**, **lambda**, and **beta**, which are parameters of a CTMC sub-model assigned to the TFail transition.

Before such an implementation, the proposed method had already been successfully tested without full automation. Section 5.3 presents a case study with results computed with

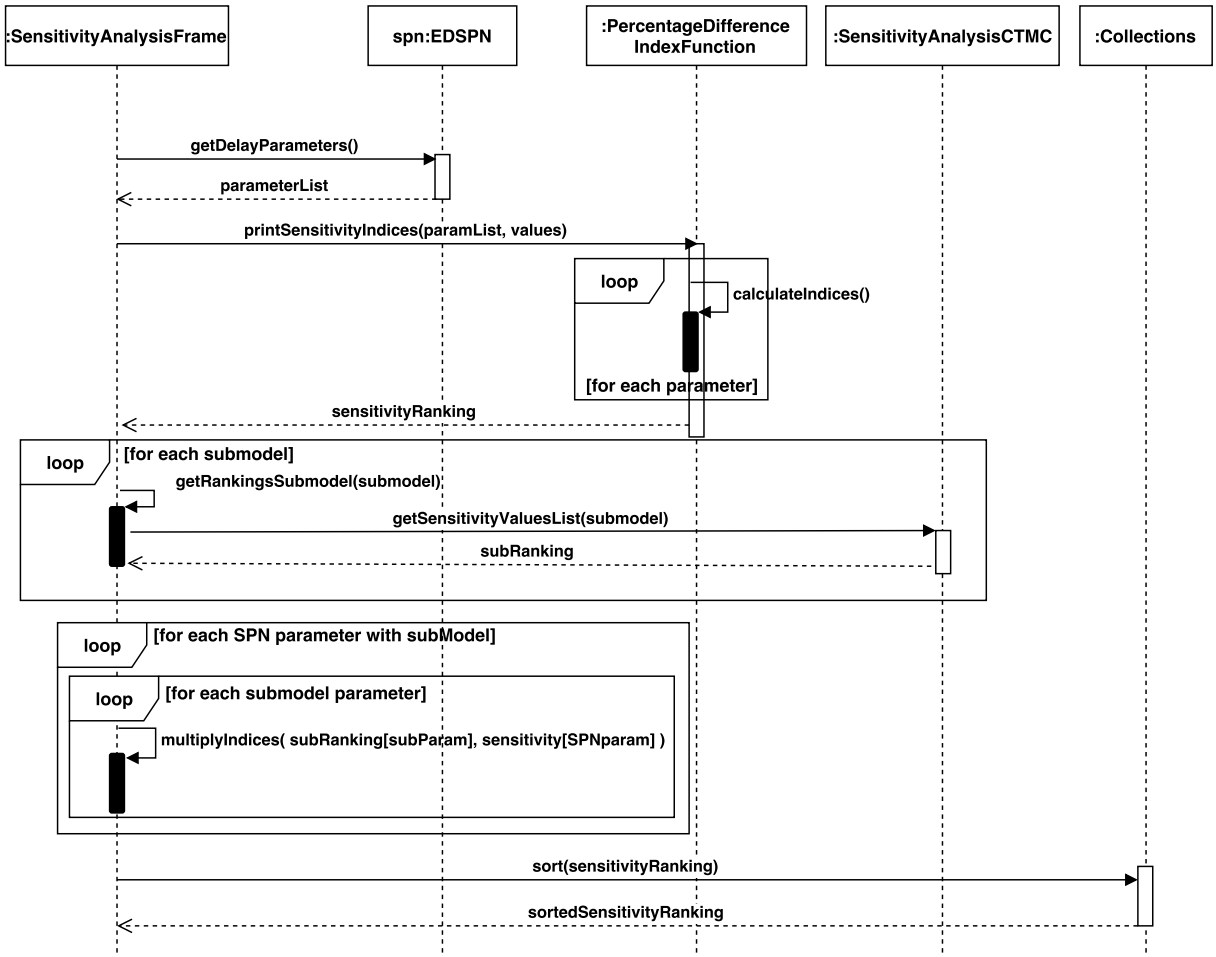


Figure A.5: UML sequence diagram for sensitivity computation with SPN main model and CTMC sub-models

this method.

In Mercury source code, five classes interact to accomplish the proposed sensitivity analysis steps in hierarchical composition of SPN and CTMCs. Figure A.5 shows a UML sequence diagram that describes the execution of methods for that activity. The user triggers the sensitivity analysis in **SensitivityAnalysisForm**, that is the class of the dialog window shown in Figure A.4. The **EDSPN** class is responsible for returning the list of parameters of the main model. Next, the class **PercentageDifferenceIndexFunction** computed and provides the sensitivity ranking for the main model, using the percentage difference method. The class **SensitivityAnalysisCTMC** is responsible for computing and returning the sensitivity indices for each CTMC sub-model. Those indices and then multiplied by the indices of respective parameters from the SPN ranking. The method *sort()* from the **Collections** class is called to produce a ordered ranking, finishing the procedure.

It is important to highlight that was necessary implementing sensitivity analysis features for single (i.e., non-hierarchical models) before developing features described in the current section. These initial features were released in version 4.2.1. The Mercury tool also did not support hierarchical modeling before the efforts of this research work. Version 4.5.0 included

that feature. Such an implementation was a requisite for proceeding forward in the automation of the proposed methodology. All features presented in this appendix are available since version 4.6.0 of Mercury.

B

Partial Derivatives for Case Study 1

The following equations (B.1, B.2, B.3, and B.4) show the partial derivatives for the availability of the redundant Cloud Manager (CLC) component from the case study in Section 5.1.

$$S_{\lambda_{CLC}}(A_{CLC}) = -\frac{(sa\alpha_2 + {\alpha_1}^2 + \lambda_{CLC}\lambda_{CLC_i})\mu_{CLC}({\alpha_1}^2 + \alpha_3\lambda_{CLC_i} + \lambda_{CLC}\lambda_{CLC_i} + sa(\alpha_1 + 2\lambda_{CLC}))}{(\alpha_3({\alpha_1}^2 + \lambda_{CLC}\lambda_{CLC_i}) + sa(\lambda_{CLC}^2 + \lambda_{CLC}\alpha_1 + \mu_{CLC}\alpha_1))^2} + \frac{\mu_{CLC}(sa + \lambda_{CLC_i})}{\alpha_3({\alpha_1}^2 + \lambda_{CLC}\lambda_{CLC_i}) + sa(\lambda_{CLC}^2 + \lambda_{CLC}\alpha_1 + \mu_{CLC}\alpha_1)}, \quad \text{(B.1)}$$

where

$$\begin{aligned}\alpha_1 &= \lambda_{CLC_i} + \mu_{CLC},\\ \alpha_2 &= \lambda_{CLC} + \lambda_{CLC_i} + \mu_{CLC}, \text{ and}\\ \alpha_3 &= \lambda_{CLC} + \mu_{CLC}.\end{aligned}$$

$$S_{\lambda_{CLC_i}}(A_{CLC}) = \frac{(\lambda_{CLC} + 2\lambda_{CLC_i} + sa + 2\mu_{CLC})\mu_{CLC}}{(\lambda_{CLC}^2 + \lambda_{CLC}\alpha_1 + \alpha_1\mu_{CLC})sa + (\lambda_{CLC}\lambda_{CLC_i} + {\alpha_1}^2)\alpha_3} - \frac{(\alpha_3(\lambda_{CLC} + 2\lambda_{CLC_i} + 2\mu_{CLC}) + \alpha_3 sa)(\lambda_{CLC}\lambda_{CLC_i} + {\alpha_1}^2 + \alpha_2 sa)\mu_{CLC}}{((\lambda_{CLC}^2 + \lambda_{CLC}\alpha_1 + \alpha_1\mu_{CLC})sa + (\lambda_{CLC}\lambda_{CLC_i} + {\alpha_1}^2)\alpha_3)^2}, \quad \text{(B.2)}$$

where

$$\begin{aligned}\alpha_1 &= \lambda_{CLC_i} + \mu_{CLC},\\ \alpha_2 &= \lambda_{CLC} + \lambda_{CLC_i} + \mu_{CLC}, \text{ and}\\ \alpha_3 &= \lambda_{CLC} + \mu_{CLC}.\end{aligned}$$

$$
\begin{aligned}
S_{\mu_{CLC}}(A_{CLC}) = &- \frac{(\alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i} + 2\alpha_3\alpha_1 + (\alpha_4 + 2\mu_{CLC})sa)\mu_{CLC}(\alpha_2 sa + \alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i})}{((\lambda_{CLC}\alpha_1 + \mu_{CLC}\alpha_1 + \lambda_{CLC}^2)sa + \alpha_3(\alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i}))^2} \\
&+ \frac{\alpha_2 sa + \alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i}}{(\lambda_{CLC}\alpha_1 + \mu_{CLC}\alpha_1 + \lambda_{CLC}^2)sa + \alpha_3(\alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i})} \\
&+ \frac{(2\lambda_{CLC_i} + sa + 2\mu_{CLC})\mu_{CLC}}{(\lambda_{CLC}\alpha_1 + \mu_{CLC}\alpha_1 + \lambda_{CLC}^2)sa + \alpha_3(\alpha_1{}^2 + \lambda_{CLC}\lambda_{CLC_i})},
\end{aligned}
\tag{B.3}
$$

where

$$
\begin{aligned}
\alpha_1 &= \lambda_{CLC_i} + \mu_{CLC}, \\
\alpha_2 &= \lambda_{CLC} + \lambda_{CLC_i} + \mu_{CLC}, \text{ and} \\
\alpha_3 &= \lambda_{CLC} + \mu_{CLC} \\
\alpha_4 &= \lambda_{CLC} + \lambda_{CLC_i}.
\end{aligned}
$$

$$
\begin{aligned}
S_{sa}(A_{CLC}) = &\frac{\alpha_2\mu_{CLC}}{sa(\alpha_1\lambda_{CLC} + \lambda_{CLC}^2 + \alpha_1\mu_{CLC}) + (\lambda_{CLC_i}\lambda_{CLC} + \alpha_1{}^2)\alpha_3} \\
&- \frac{(\lambda_{CLC_i}\lambda_{CLC} + \alpha_2 sa + \alpha_1{}^2)\mu_{CLC}(\alpha_1\lambda_{CLC} + \lambda_{CLC}^2 + \alpha_1\mu_{CLC})}{(sa(\alpha_1\lambda_{CLC} + \lambda_{CLC}^2 + \alpha_1\mu_{CLC}) + (\lambda_{CLC_i}\lambda_{CLC} + \alpha_1{}^2)\alpha_3)^2},
\end{aligned}
\tag{B.4}
$$

where

$$
\begin{aligned}
\alpha_1 &= \lambda_{CLC_i} + \mu_{CLC}, \\
\alpha_2 &= \lambda_{CLC} + \lambda_{CLC_i} + \mu_{CLC}, \text{ and} \\
\alpha_3 &= \lambda_{CLC} + \mu_{CLC}.
\end{aligned}
$$

The partial derivatives equations for the availability of the Cluster Manager (CC) component are very similar to those just presented for the CLC component, since both use the same CTMC model for describing their warm-standby redundancy.

The following equations (B.5 and B.6) present the partial derivatives of a node availability with respect to its parameters μ_{NC} and λ_{NC}, respectively.

$$
S_{\mu_{NC}}(A_{NC}) = \frac{1}{\lambda_{NC} + \mu_{NC}} - \frac{\mu_{NC}}{(\lambda_{NC} + \mu_{NC})^2} \tag{B.5}
$$

$$S_{\lambda_{NC}}(A_{NC}) = -\frac{\mu_{NC}}{(\lambda_{NC}+\mu_{NC})^2} \tag{B.6}$$