MARCELO DUTRA ÓS

**A COMMUNITY CLOUD ARCHITECTURE FOR
REAL-TIME APPLICATIONS**

**UMA ARQUITETURA DE NUVEM EM COMUNIDADE
PARA APLICAÇÕES DE TEMPO REAL**

**SÃO PAULO
2016**

MARCELO DUTRA ÓS

**A COMMUNITY CLOUD ARCHITECTURE FOR
REAL-TIME APPLICATIONS**

**UMA ARQUITETURA DE NUVEM EM COMUNIDADE
PARA APLICAÇÕES DE TEMPO REAL**

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Ciências

Área de Concentração: Engenharia de Computação

Orientadora: Profa. Dra. Graça Bressan

**SÃO PAULO
2016**

A COMMUNITY CLOUD ARCHITECTURE FOR REAL-TIME APPLICATIONS

UMA ARQUITETURA DE NUVEM EM COMUNIDADE PARA APLICAÇÕES DE TEMPO REAL

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 30/11/2016. A versão original está disponível para consulta na Biblioteca da Eng. Elétrica da Escola Politécnica da USP.

Comissão Julgadora:

- Prof^a. Dr^a. Graça Bressan (orientadora) - Escola Politécnica-PCS-USP
- Prof^a. Dr^a Liria Matsumoto Sato - Escola Politécnica-PCS-USP
- Prof. Dr. Fernando Frota Redigolo - Autônomo
- Prof. Dr. Marcos Dias de Assunção - INRIA Institut, França
- Prof. Dr. Rodrigo Neves Calheiros - University of Melbourne, Austrália

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

Ös, Marcelo Dutra

Uma arquitetura de nuvem em comunidade para aplicações de tempo real
/ M. D. Ös -- versão corr. -- São Paulo, 2015.
155 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo.
Departamento de Engenharia de Computação e Sistemas Digitais.

1.Computação em nuvem 2.Tempo-real (Aplicações I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

DEDICATORY

To my beloved Ana, who is the center of my life and who shows me always the right and simplest way. For my beloved son, Igor, and my beloved daughter, Heloísa, both who make my life full of sense and who teaches me something new and wonderful everyday. Hope your life that is just beginning resemble all the good fruits your father and mother wish for you. To God, who taught man how to think, but, above all, to will and to believe.

To a little angel, who God carried close to Him.

ACKNOWLEDGEMENTS

I wish to thank my thesis supervisor Professor Graça Bressan for the greatest opportunities she has provided me during my academic studies. Her patience, guidance and words of encouragement were invaluable during the researching and writing of this doctoral thesis.

Also, I would like to thank several Professors at Escola Politécnica of University of São Paulo, who gave me the foundation an engineer needs to have. To name a few, Jorge Risco Becerra, Wilson Ruggiero, Jorge Amazonas, etc.

For the members of the committee formed to evaluate this thesis, I wish to thank for all their time, dedication, guidance and patience during the writing and reviewing of this thesis.

For the mates I have made during my academic studies, with whom I have spent incredible moments of relaxing and fun.

And of course, for my parents, who made countless sacrifices in order to raise me and my sister and gave all the educational and moral foundation we needed for life.

RESUMO

ÓS, M. D. **Uma arquitetura de nuvem em comunidade para aplicações de tempo real.** 2016. 122 f. Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo, Universidade de São Paulo, São Paulo, 2016.

A Computação em Nuvem é um paradigma de computação distribuída que vem sendo utilizado extensivamente em vários campos de interesse nos últimos anos, desde aplicações web comuns até a aplicações de alta-performance computacional. O modelo de pagamento pelo uso e a isonomia dos métodos de acesso transformaram o ambiente de Computação em Nuvem em uma alternativa extremamente popular e atrativa tanto para universidades como para empresas privadas. Entre os modelos de implantação adotados atualmente destaca-se o de nuvem em comunidade, onde várias entidades que possuem interesses em comum constroem, mantêm e compartilham a mesma infraestrutura de serviços em nuvem. O modelo computacional em nuvem também pode ser atrativo para aplicações que tenham como requisito o processamento em tempo real, principalmente pela capacidade de manipulação de grandes volumes de dados e pela propriedade de elasticidade, que é a inserção ou remoção de recursos computacionais dinamicamente de acordo com a demanda. Nesta tese, são identificados os requisitos para a construção de um ambiente em nuvem em comunidade para aplicações de tempo real. A partir destes requisitos e de uma revisão bibliográfica baseada em nuvem e sistemas distribuídos de tempo real, é desenvolvida a proposta de uma arquitetura de nuvem em comunidade de tempo real. Um estudo de caso de compra e venda de ações em bolsa de valores é apresentado como uma aplicação viável para este modelo, sendo que um algoritmo de escalonamento de tempo real para este ambiente é proposto. Por fim, é desenvolvido nesta tese um simulador cujo objetivo é demonstrar em termos quantitativos quais as melhorias de desempenho atingidas com esta arquitetura.

Palavras-chave: computação em nuvem, aplicações de tempo real, algoritmos de escalonamento, aplicações financeiras

ABSTRACT

Ós, M. D. **A community cloud architecture for real-time applications**. 2016. 122 f. Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo, Universidade de São Paulo, São Paulo, 2016.

Cloud Computing is a distributed computing paradigm which is being extensively applied to many fields of interest in the last few years, ranging from ordinary web applications to high-performance computing. The pay-per-use model and ubiquitous access methods have made Cloud Computing an interesting and popular alternative for both enterprises and universities. Among the deployment models adopted, one of the most prominent is the community cloud, where several entities who share similar interests build, maintain and use the same infrastructure of cloud services. The cloud computing paradigm can be attractive to applications whose requirements are the processing in real-time too, mainly because of its capacity of handling huge amounts of data as for the property of elasticity, which is the dynamic and automatic insertion or removal of computing resources on-demand. In this thesis, the requirements of a community cloud for real-time applications are identified. Based on these requirements and on a bibliographical review of the research fields of real-time distributed systems and real-time clouds, it is developed a proposal for a real-time community cloud architecture. A case study of a trading real-time application at a stock exchange is presented as a feasible application for this model. Also, a real-time scheduling algorithm is proposed for this environment. A simulator is built in order to demonstrate the quantitative improvements this architecture brings.

Keywords: cloud computing, real-time applications, scheduling algorithms, stock exchanges

Contents

List of Figures	xii
List of Tables	xiv
LIST OF ABBREVIATIONS AND ACRONYMS	xv
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Contributions	3
1.4 Method	4
1.5 Organization	5
2 CONCEPTS	7
2.1 Cloud Computing	7
2.2 Community Clouds	11
2.3 Real-time systems	12
2.4 Limitations of Cloud Environments for the Support of Real-Time Applications	13
2.5 Advantages of Adopting Cloud Environments for the Support of Real-Time Applications	13
2.6 Key questions	14
2.7 Concerns	15
3 BIBLIOGRAPHICAL REVIEW	17
3.1 Description of relevant real-time distributed systems	17
3.1.1 Real-time distributed operating systems	17
3.1.2 Real-time distributed middlewares	25
3.1.3 Common characteristics of real-time distributed systems	27
3.1.4 A taxonomy for real-time distributed systems	27
3.2 Description of relevant real-time cloud systems	31
3.2.1 RT-Xen	31
3.2.2 Hadoop-RT	32
3.2.3 Cloud service framework for real-time applications	33

3.2.4	RTF-RMS	34
3.2.5	RACE	35
3.2.6	Global-RT	35
3.2.7	Scheduling based on time utility function model	36
3.2.8	Discussion of the characteristics of real-time clouds research	36
3.2.9	A taxonomy for real-time cloud systems	37
4	REQUIREMENTS OF A REAL-TIME COMMUNITY CLOUD	40
4.1	Functional Requirements	42
4.2	Non-Functional Requirements	44
4.2.1	Timeliness	44
4.2.2	Capacity to handle peak loads	47
4.2.3	Predictability	47
4.2.4	High Availability	47
4.2.5	Modularity	47
4.2.6	Isolation among users	48
4.2.7	Elasticity	48
4.2.8	Scalability	48
4.2.9	Reliability	49
5	CASE STUDY: A REAL-TIME COMMUNITY CLOUD FOR A FINANCIAL TRADING APPLICATION	50
5.1	Goals	50
5.2	Scope	51
5.3	Real-time trading financial application	51
5.4	Typical parameters for trading	53
5.5	Typical infrastructure deployed at collocation facilities	55
5.6	Proposed real-time community cloud infrastructure	57
5.6.1	Hard real-time tasks	57
5.6.2	Soft real-time tasks	58
5.6.3	Regular tasks	58
5.6.4	Control of resources sharing	58
5.6.5	Collaboration among participants	59
5.7	Use Cases	60
5.7.1	Actors	60
5.7.2	List of use cases	61
6	ARCHITECTURE OF A REAL-TIME COMMUNITY CLOUD	63
6.1	Services	63
6.2	Community clouds	64

6.3	Tasks	65
6.4	Comparison to grids	66
6.5	Real-time orchestrator architecture	66
6.5.1	Task reception module architecture	67
6.5.2	Quality of service module architecture	69
6.5.3	Scheduling control module architecture	70
6.5.4	Run-time environment module architecture	73
6.5.5	Physical environment performance control module architecture	73
6.5.6	Virtual environment performance control module architecture	76
6.6	Communications architecture	76
6.7	Mechanisms	77
6.7.1	Hierarchical scheduling	77
6.7.2	An heuristic-based scheduler for a community cloud designed to support real-time trading applications	78
6.7.3	Adaptive Topology Mechanisms	82
6.7.4	Pre-provisioning of Virtual Environments	83
7	SIMULATIONS	84
7.1	Simulator	85
7.1.1	Data structure	89
7.1.2	Random number generator	90
7.2	Real-time capability	91
7.2.1	Baseline	91
7.2.2	Minimum virtualization overhead	94
7.2.3	Maximum oversubscription factor	96
7.2.4	CPUs with more capacity	98
7.2.5	More CPUs	100
7.2.6	More hard real-time tasks	102
7.2.7	More parallel tasks	104
7.2.8	Exponential distribution	105
7.3	Heuristic-based scheduler for a real-time trading application	107
8	CONCLUSIONS AND FUTURE WORKS	110
8.1	Conclusions	110
8.2	Future Works	111
8.2.1	Architecture	111
8.2.2	Scheduling	111
8.2.3	QoS	112
8.2.4	Simulations	112
8.2.5	Real-time communication	113

8.2.6	Dynamics of the behaviour of participants	114
8.2.7	Prototype	114
8.2.8	Benchmarks for real-time clouds	114
Bibliography		115

List of Figures

2.1	Infrastructure as a Service - IaaS	9
2.2	Platform as a Service - PaaS	9
2.3	Software as a Service - SaaS	10
5.1	Trading environment	52
5.2	Typical infrastructure at a stock exchange's collocated rental spaces for trading	56
5.3	Real-time community cloud infrastructure	57
6.1	Community Cloud Environment	65
6.2	Real-Time Orchestrator Architecture	68
6.3	Task reception module - Architecture	69
6.4	Quality of service module - Architecture	71
6.5	Scheduling control module - Architecture	72
6.6	Run-time environment module - Architecture	74
6.7	Physical environment performance control module - Architecture	75
6.8	Virtual environment performance control module - Architecture	76
6.9	Hierarchical Scheduling Mechanism	77
7.1	Simulator - Structure	86
7.2	Data structure - real-time capability	89
7.3	Data structure - additional structures	90
7.4	Results - Baseline - Number of CPU Cycles	92
7.5	Results - Baseline - Percentage of completed hard real-time tasks	93
7.6	Results - Baseline - Percentage of completed soft real-time tasks	94
7.7	Results - Minimum virtualization overhead - Number of CPU Cycles	95
7.8	Results - Minimum virtualization overhead - Percentage of completed hard real-time tasks	95
7.9	Results - Minimum virtualization overhead - Percentage of completed soft real- time tasks	96
7.10	Results - Maximum oversubscription factor - Number of CPU Cycles	96
7.11	Results - Maximum oversubscription factor - Percentage of completed hard real-time tasks	97

7.12 Results - Maximum oversubscription factor - Percentage of completed soft real-time tasks	98
7.13 Results - CPUs with more capacity - Number of CPU Cycles	98
7.14 Results - CPUs with more capacity - Percentage of completed hard real-time tasks	99
7.15 Results - CPUs with more capacity - Percentage of completed soft real-time tasks	99
7.16 Results - More CPUs - Number of CPU Cycles	100
7.17 Results - More CPUs - Percentage of completed hard real-time tasks	101
7.18 Results - More CPUs - Percentage of completed soft real-time tasks	101
7.19 Results - More hard real-time tasks - Number of CPU Cycles	102
7.20 Results - More hard real-time tasks - Percentage of completed hard real-time tasks	103
7.21 Results - More hard real-time tasks - Percentage of completed soft real-time tasks	103
7.22 Results - More parallel tasks - Number of CPU Cycles	104
7.23 Results - More parallel tasks - Percentage of completed hard real-time tasks . .	105
7.24 Results - More parallel tasks - Percentage of completed soft real-time tasks . .	105
7.25 Results -Exponential distribution - Number of CPU Cycles	106
7.26 Results - Exponential distribution - Percentage of completed hard real-time tasks	106
7.27 Results - Exponential distribution - Percentage of completed soft real-time tasks	107
7.28 Results - Heuristic-based scheduler - Number of CPU Cycles	108
7.29 Results - Heuristic-based scheduler - Percentage of completed hard real-time tasks	108
7.30 Results - Heuristic-based scheduler - Percentage of completed soft real-time tasks	109
7.31 Results - Heuristic-based scheduler - Profit	109

List of Tables

3.1	A taxonomy for real-time distributed systems - part 1	29
3.2	A taxonomy for real-time distributed systems - part 2	30
3.3	A taxonomy for real-time cloud systems	38
4.1	Functional requirements - system's	43
4.2	Functional requirements - user's	43
4.3	Functional requirements - scheduler's	44
4.4	Functional requirements - hardware control and visibility	45
4.5	Functional requirements - programming capabilities	45
4.6	Functional requirements - communication capabilities	45
4.7	Functional requirements - portability	46
4.8	Functional requirements - virtualization	46
4.9	Functional requirements - security	46
5.1	Typical values for stock exchanges	55
7.1	Simulator - range of arguments	91
7.2	Experiment Baseline: input variables	92
7.3	Experiment Baseline: calculated variables	92

LIST OF ABBREVIATIONS AND ACRONYMS

AWS	Amazon Web Services
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
EC2	Amazon's Elastic Cloud Computing
EDF	Earliest-Deadline First
FIFO	First-In-First-Out
GBS	Generic Benefit Scheduling
HPC	High-Performance Computing
HRT	Hard Real-Time
IaaS	Infrastructure-as-a-Service
IP	Internet Protocol
LAN	Local Area Network
OO	Object-Oriented
OS	Operating System
PaaS	Platform-as-a-Service
QFD	Quality Function Deployment
RM	Rate-Monotonic
RMS	Resource Management System
ROIA	Real-Time Online Interactive Applications
RT	Real-Time
RTF	Real-Time Framework
SaaS	Software-as-a-Service
SRT	Soft Real-Time
TDMA	Time-Division Multiplexing Access
TUF	Time Utility Function
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VMM	Virtual Machine Monitor
WAN	Wide Area Network
WCET	Worst-Case Execution Time
XPA	Extra-Performance Architecture

Chapter 1

INTRODUCTION

Cloud Computing is a distributed computing paradigm which is being extensively applied to many fields of interest in the last few years, ranging from commonly adopted web applications to high-performance computing (HPC) [ARMBRUST et al. 2009], [FOSTER et al. 2008], [BRYANT 2011]. The pay-per-use model and ubiquitous access methods have made Cloud Computing an interesting and popular alternative for both enterprises and universities. Among the deployment models adopted, one of the most prominent is the community cloud, where several entities who share similar interests build, maintain and use the same infrastructure of cloud services. The cloud computing paradigm can be attractive to applications whose requirements are the processing in real-time too, mainly because of its capacity of handling huge amounts of data as for the property of elasticity, which is the dynamic and automatic insertion or removal of computing resources on-demand. However, as it has been primarily designed to maximize throughput and the utilization of resources, and having technologies such as virtualization at its core, some limitations have to be overcome.

1.1 Motivation

Given the rapid deployment and extensive offerings of clouds, either public, private, community or hybrid ones, many applications are being migrated to these environments in order to accelerate deployment time, save costs and take the most of available resources. This has been successfully achieved in the

last few years for many types of applications. Conversely, classical real-time applications were consistently deployed in dedicated real-time distributed systems in order to maintain predictability and real-time performance goals. Natively, cloud environments cannot sustain the requirements of real-time applications as they were not primarily designed with these goals [PHAN et al. 2011], [ENDO et al. 2011], [BITTENCOURT, MADEIRA and FONSECA 2012], [LIU, QUAN and REN 2010], [GORLATCH et al. 2012].

The motivation behind this thesis lies in the fact that cloud computing systems can offer many advantages for the deployment of real-time applications, such as the capacity of handling huge amounts of data, ubiquitous access methods, portability through seamless execution of applications over different clouds and over heterogeneous environments, elasticity, multi-tenancy, which is the property of simultaneous sharing of processing capacity among several users, among others. The extensive research and literature regarding real-time distributed systems have addressed the main issues and design decisions to enable real-time applications to run in distributed systems. These achievements are taken into consideration in this effort to design real-time community clouds.

1.2 Objectives

The main objective of this thesis is to demonstrate that the cloud computing paradigm is suitable for the processing of real-time applications.

A novel architecture has to be proposed in order to achieve so. The mechanisms which this real-time cloud must support are identified based on the requirements of real-time applications. Among these mechanisms, one of utmost importance is the scheduling algorithm which should be applied in the modeled architecture in order to achieve the requirements of real-time applications. Real-time scheduling algorithms should adopt heuristic-based strategies, as the problem of calculating the optimal scheduling in clouds happens on-line and it

is NP-complete.

Another objective of this thesis is to identify a case study which can take advantage of adopting the cloud paradigm for real-time applications. An actual financial real-time application for trading at stock exchanges is depicted in order to provide a scenario suitable for it. It is based on trading systems which send orders of buy/sell to stock exchanges in real-time. Generally, dedicated hardware systems have been employed by the participants of the stock market to deploy its algorithms and send and receive orders to/from stock exchanges as fast and predictable as possible. With the advent of cloud computing, novel approaches for this environment have to be proposed. Another aim of this thesis is to develop a real-time scheduling algorithm suitable to solve this problem in an optimal manner while adopting the architecture described.

A final objective of this thesis is to develop a simulator in order to quantitatively measure the improvements that the architecture proposed and its mechanisms can provide.

1.3 Contributions

The main contributions of this work are:

- A novel architecture for the support of real-time applications in community clouds
- A feasible and real case study based on trading at stock exchanges where this architecture could be applied
- A novel heuristic-based scheduling algorithm to be applied in this cloud environment which can perform better than other approaches such as First-In-First-Out (FIFO) and Earliest-Deadline First (EDF)
- A simulator where the concepts proposed in this thesis are quantitatively validated. This simulator supports many different scenarios and could be

expanded further

- A quantitative comparison of the improvements this real-time community cloud architecture provides for the processing of real-time applications
- The identification of how much each parameter of this architecture should be improved in order to obtain better quantitative results for the processing of real-time applications
- A quantitative comparison of the improvements the heuristic-based scheduler provides in comparison with the FIFO and EDF approaches for the case study based on trading at stock exchanges

1.4 Method

The method adopted for the development of this thesis is a systematic one where each step presents its scope and the deliverables are identified. However, each phase is not closed within itself as it could present feedback to another ones.

The following steps have been performed:

- A research regarding the state-of-the-art of cloud computing systems was performed, in particular the ones related to real-time applications and community clouds, in order to identify opportunities for novel developments. Concepts extensively applied to distributed real-time systems, where revised in order to provide ideas for the development of real-time cloud systems;
- A list of requirements to be supported by real-time cloud systems has been identified. These were separated in functional and non-functional requirements;
- A real case study was selected in order to demonstrate the applicability of the concepts presented. For this case study, a trading real-time application which send and receive orders of buy/sell securities to stock exchanges is

described. Another similar applications which share common requirements with this case study, such as air flight controls and military applications, could apply the same concepts;

- An architecture is proposed for the real-time cloud. Based on the case study presented, it adopts the concept of community clouds, where many entities share resources in a dynamic fashion in order to provide more computing capacity to the participants of the cloud. The main service provided by this cloud is Infrastructure-as-a-Service (IaaS) for real-time applications, but it is envisioned that it could evolve to a Platform-as-a-Service (PaaS) model;
- It is developed a real-time scheduling algorithm suitable to the case study described and which can be implemented in the architecture described. It is believed that this algorithm can be applied in other applications similar to the case study presented, but which have nothing to do with financial trading
- A simulation environment was built where different experiments can be performed in order to measure quantitatively the improvements this architecture and the proposed scheduling algorithm bring for real-time applications in general and for the case study presented;
- Finally, conclusions and future works were listed and identified. These are a compilation of all the eventual developments which were identified along the writing of this thesis.

1.5 Organization

This thesis is organized as follows:

In chapter two the concepts regarding community cloud computing and real-time systems are described. The advantages regarding the execution of real-time applications over cloud environments are presented along its limitations. The key questions and concerns to be addressed in this field of research are

presented. In chapter three an extensive review of real-time distributed systems and its main characteristics are presented. Thereafter the main works regarding the researching of real-time clouds are described. A comparison among the characteristics of real-time distributed systems and real-time clouds is performed in order to highlight the main similarities and differences among them. In the next chapter the requirements for real-time community clouds are identified. A case study of a real-time trading application where this architecture can be applied is presented in chapter five. In chapter six a novel community cloud architecture is proposed along with an heuristic-based scheduling algorithm which can be applied to the case study presented. Then, a simulator which was built for this thesis is presented along with the results of several simulations scenarios which provide a quantitative analysis of the improvements this real-time community cloud architecture and the heuristic-based scheduling algorithm brings. Finally, the main conclusions of this thesis are listed along with ideas for future works.

Chapter 2

CONCEPTS

In this chapter the concepts of cloud computing, community clouds, real-time distributed systems and real-time clouds are presented. The main limitations for the execution of real-time applications over cloud systems are presented along with its advantages. The key questions which need to be addressed in order to overcome these limitations are identified.

2.1 Cloud Computing

The Cloud Computing paradigm is derived from other computing fields such as Grid Computing [FOSTER et al. 2008], Distributed Systems [TANENBAUM and STEEN 2006], Networks and Operating Systems. Cloud Computing is defined by NIST [GRANCE and MELL 2011] as

"...a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources ... that can be rapidly provisioned and released with minimal management effort or service provider interaction."

These computing resources include storage, processing, memory, network bandwidth and virtual machines. Cloud environments present some common properties such as [ARMBRUST et al. 2009], [FOSTER et al. 2008], [VARIA 2008],

[GRANCE and MELL 2011]:

- Elasticity: the capability to shrink or expand its computational capacity on demand
- Multi-tenancy: the capability to create different environments for distinct customers in the same shared infrastructure
- Processing of huge amounts of data
- Loosely-coupled distributed processing
- Capability to survive hardware failures
- Virtualization
- Heterogeneity of both physical and virtual resources and applications
- On-demand self-service
- Per-usage metering and billing

Given these characteristics, some applications are more prone to be migrated to a cloud infrastructure, such as applications which processes large amounts of data which can be co-localized to the processing infrastructure, extension of compute-intensive mobile or desktop applications and highly parallel applications.

Typically, the service model offered by providers can be classified into the following categories [RIMAL and CHOI 2012], [LENK et al. 2009]: IaaS - *Infrastructure as a Service*, where low level resources such as CPU, memory, storage and network are offered for consumption. The IaaS cloud computing model is depicted in figure 2.1.

PaaS - *Platform as a Service*, where development platforms are available in the cloud for the creation of applications. The PaaS cloud computing model is depicted in figure 2.2.

Figure 2.1: Infrastructure as a Service - IaaS **Figure 2.2: Platform as a Service - PaaS**

SaaS - *Software as a Service*, where the software itself is offered through the cloud infrastructure. The SaaS cloud computing model is depicted in figure 2.3.

Figure 2.3: Software as a Service - SaaS

Also, there are four typical deployment models [GRANCE and MELL 2011]:

- Private cloud: the cloud infrastructure is owned by one company, which is both the provider and consumer of cloud services
- Public cloud: the cloud infrastructure is offered to anyone who has a connection to the cloud provider
- Community cloud: the cloud infrastructure is shared by organizations which have interests in common. It can be managed by one of the companies, a

group of them or a third-party company

- Hybrid cloud: the cloud infrastructure is composed of two or more of the models described above, which are presented to the user transparently as a single entity

Clearly, there are some obstacles to be overcome for cloud computing services to enjoy general acceptance. The most important of them is related to security and data confidentiality, as users will start to deploy applications, virtual machines and data not over its own premises and infrastructure but over the cloud provider's infrastructure. Some other obstacles to the general adoption of Cloud Computing are listed in [ARMBRUST et al. 2009] such as data lock-in, software licensing and interoperability. Of particular importance to this thesis is the obstacle of performance unpredictability which is a challenge that should be overcome by the devising of novel technologies. This performance unpredictability is related to this new paradigm the cloud brings where many resources are shared among several customers. It is demonstrated in [ARMBRUST et al. 2009] that I/O operations are more heavily affected than the sharing of CPUs, network bandwidth or main memory. Also, the scheduling of threads in scientific environments such as HPC can be affected by the adoption of virtual machines which provide a loosely control over the processor's clock and the hardware.

2.2 Community Clouds

The concept of community clouds was first described by Briscoe and Marino in [BRISCOE and MARINO 2009]. Community clouds derive from the concept of digital ecosystems [BRISCOE and WILDE 2006] and present properties of adaptation, self-organization, scalability and sustainability, inspired by natural ecosystems, which are robust and scalable too. In this type of architecture, the infrastructure is shared by different companies which present a common goal [RIMAL and CHOI 2012]. These companies can provide resources to each other

in order to increase processing capabilities during peak times, taking on the roles of consumer, producer and (or) coordinator [BRISCOE and MARINO 2009]. These resources might be servers which provide CPU cycles, network bandwidth and storage areas which are connected locally to each other in a certain physical topology, which could assume many logical forms. It is envisioned that partner companies might establish community clouds depending on their economic interests; conversely, public or private clouds do not provide this level of flexibility and adaptation as they possess a centralized controller or arbiter who is the owner of the scheduling decisions and the resources for the whole system.

The primary goal of such communities is to provide a better performance than the sum of standalone deployments. Also, another goal is to have a lower infrastructure cost versus performance ratio, therefore allowing its members to save money but still being able to achieve performance goals.

2.3 Real-time systems

A real-time system presents the following features as described by Giorgio C. Buttazzo in [BUTTAZZO 2005]: timeliness - which is the property of executing tasks correctly and within a required deadline, capacity to handle peak loads, predictability, fault-tolerance and modularity. It can be classified into two types [CHENG 2002]: hard - which presents the property that if one deadline fail a catastrophic condition will occur - and soft - where if one deadline is missed the system will not perform adequately but no damage will affect the surrounding environment. In order to meet deadline constraints, real-time systems employ a wide variety of real-time scheduling algorithms, such as EDF (Earliest Deadline First) [HORN 1974]. A real-time system is therefore constrained by timing deadlines and integrity checking. From an architectural perspective, A. Cheng explains in [CHENG 2002] that

"a real-time system has a decision component that interacts with the external environment (in which the decision component is embedded)

by taking sensor readings and computing control decisions based on sensor reading and stored state information."

Examples of real-time applications are as broad as: flight control systems, military systems, robotics, industrial automation and financial applications among others [CHENG 2002], [LIU 2000].

2.4 Limitations of Cloud Environments for the Support of Real-Time Applications

Cloud environments were primarily designed to maximize throughput and the utilization of resources. Currently, they present limitations to support real-time applications such as: scheduling algorithms which do not take time into account [PHAN et al. 2011], [LIU, QUAN and REN 2010], [GORLATCH et al. 2012], no support for a real-time clock as an internal time reference [BUTTAZZO 2005], delay to provision resources and virtual machines [WU et al. 2012], no predictability of the execution of tasks [ARMBRUST et al. 2009] and simple QoS (Quality of Service) mechanisms [PERROS and XIONG 2009]. Also, the suboptimal physical topology and connectivity commonly adopted in order to accommodate a large number of applications brings performance penalties in comparison with other distributed systems, such as grids, as described by R. Aversa in [AVERSA 2011].

2.5 Advantages of Adopting Cloud Environments for the Support of Real-Time Applications

Despite these limitations, cloud environments are indeed attractive to the deployment of real-time applications as they present features such as: elasticity, multi-tenancy, ability to survive hardware failures, virtualization support and a layer of abstraction where many different applications could execute its own (original) code with no knowledge of the underlying platform of software and

hardware, which provides flexibility and portability. Also, the opportunity to increase or decrease on-demand the number of computing resources at disposal is very interesting for the development and testing of new real-time applications.

2.6 Key questions

In order to enumerate the key questions that need to be answered for building real-time cloud systems, it is useful to get insights from paradigms adopted in classical real-time distributed systems. A high degree of parallelism must be allowed, as this will bring important improvements in achieving very strict deadlines. This raises the issue of how to schedule and coordinate the execution of parallel tasks in a virtualized cloud environment. Many works cite that the responsibility of managing resources efficiently should not be done at the expense of application programmers, as in [NORTHCUTT 1987]. Therefore, the cloud system itself should manage to allocate all the resources in real-time for all users, presenting mechanisms to solve contention problems during intervals of high utilization. As described by J. Duane Northcutt in [NORTHCUTT 1987]

“the aspect that is most involved in meeting the needs of real-time applications is the manner in which contention for system resources is solved.”

Predictability, which is the property that a system will always take the same time to execute a program given a certain input regardless of the system’s utilization [MOK 1983], is currently a challenge in cloud systems, mainly because the peak load conditions are not known beforehand. Ultimately, real-time cloud systems should perform with no variability in response time for a given input for a given application. It is useful here to bring a citation from B. Shneiderman in [SHNEIDERMAN 1979] that

“...increasing the variability of response time generates poorer perfor-

mance and lower user satisfaction. Users may prefer a system which always responds in 4.0 seconds to one which varies from 1.0 to 6.0 seconds, even though the average in the second case is 3.5."

Some assumptions should be made about the cloud system workload that it would allow it to react on-line and dynamically during peak load intervals.

Given that real-time cloud systems should hold dynamic properties and present on-line reaction to changes in the workload, a system which performs scheduling off-line, statically, would be of little use in these environments. Typically, scheduling problems in real-time dynamic systems have been solved through the development of novel adaptive scheduling algorithms, most of them based on heuristics specifically developed for a particular environment or application [STANKOVIC 1988]. Also, tasks to be run in real-time cloud systems should have its time bounds delimited, whereas timing constraints must be derived from the environment and the implementation.

Finally, given the capacity of processing huge amounts of data, real-time cloud systems must address problems related to data localization and distribution [PHAN et al. 2011], which should be taken in consideration regarding any scheduling strategy. In the case of MapReduce's cloud applications, the precedence relationships among tasks is too a very important property which scheduling strategies should support.

2.7 Concerns

Considering these key questions, it is possible to detail the concerns regarding the porting of real-time applications into cloud systems. First, the overhead required in order to set up virtual machines should not be overlooked. These are related to the hypervisor and virtualization layers, building of virtual networks in order for the virtual machines to communicate, among others [LLORENT 2011]. In particular, the cost and performance of communication between hosts (or

VMs) in a real-time distributed environment has been a concern for a long time.

Also, for hard real-time environments, monopolizing system resources should be deliberately forbidden [MOK 1983]. Additional issues based on classic real-time systems' theory might impact directly the development of real-time cloud systems. J. Duane Northcutt states that it is easily demonstrated that, in general, fixed and static priority assignments are incapable of meeting activity deadlines, even when the computing capacity is far greater than the needs of activities [NORTHCUTT 1987]. The complexity involved in the managing and scheduling of so many pieces of hardware (or virtual slices of hardware) only get worse as the number of resources increase, complicating even more the coordination among tasks and communications [STANKOVIC 1988]. Also, over-provisioning brings some other problems as increased cooling and operation and management costs [KIM and PARASHAR 2011].

In fact, there is no guarantee that all of an application's timeliness requirements will be always met and trying to do so will require the system to conform to certain unrealistically over-simplified assumptions regarding system behaviour [NORTHCUTT 1987]. In a cloud multi-tenant environment where many users and applications share the same resources, a realistic assumption is that only certain and selected hard deadlines will be met.

Chapter 3

BIBLIOGRAPHICAL REVIEW

In this chapter it is presented an extensive review of the main principles applied in the design of real-time distributed systems. The most important research systems developed and deployed are explained. Then, the advances and achievements regarding real-time cloud computing systems are described. This inquiry aims to compare real-time classic distributed systems against real-time cloud systems in order to reveal similarities and antagonisms between them.

3.1 Description of relevant real-time distributed systems

3.1.1 Real-time distributed operating systems

RNet

RNet is a distributed real-time kernel proposed by M. Coulas *et. al* in [COULAS, MACEWEN and MARQUIS 1987] in 1987. It provides message-passing communication facilities and a real-time scheduler which employs a modified Earliest Deadline With Preemption (m-E-D-W-P) algorithm [COULAS, MACEWEN and MARQUIS 1987]. Three types of processes can be scheduled in RNET: real-time processes, which can be periodic or sporadic, alarm processes and background processes, these last executed in a round-robin fashion. The message-passing model adopted in RNet is based on ports that have associated deadlines.

Designed to be a high-level programming system where distributed hard real-time programs can be built and executed, it provides a proprietary language in order to specify the structure and real-time properties of a distributed program. An interesting feature of RNet is the modeling of the physical communications network during the specification of a real-time distributed program in order to achieve better efficiency. Also, it is possible to test how a distributed program will perform over a particular hardware during the specification phase. If the efficiency is not good enough, the program can be modified and tested again before going live.

RNet was firstly executed over NS32000-based Unix systems connected via an Ethernet network, but the main objective of the developers was to support a set of heterogeneous processors and devices [COULAS, MACEWEN and MARQUIS 1987]. In order to do so, several versions of the kernel had to be developed for each set of hardware. Its main application was in the field of multisensory robots, whose environment is characterized by several updates and replacements in the supporting hardware.

RK/Timix

Developed at the University of Pennsylvania around 1988 by Insup Lee *et. al*, RK [LEE, PAUL and KING 1988] is a real-time kernel which provides predictable response for distributed systems. This is achieved by means of a scheduling process and communications based on timing constraints. The kernel provides primitives for applications whereby the specification of timing requirements can be guaranteed in advance by the scheduler. These timing requirements are as simple as: start time, maximum execution time, deadline, an unique id and a flag which points that the process is a hard or a soft real-time one. Also, the primitives provided by RK allow the direct control of devices by application processes.

Processes in RK are classified as regular, soft real-time or hard real-time.

Regular processes are scheduled based on their priority whereas real-time processes are scheduled based on their deadlines [LEE, PAUL and KING 1988]. Hard real-time processes are always scheduled in advance and, if accepted, there is a guarantee that they will be executed by the system. In RK, all real-time processes must be periodic [LEE, PAUL and KING 1988]. Regarding communication primitives, RK natively provides three basic methods to guarantee a predictable fast communication within certain timing deadlines between processes. The application domain for RK is robotics, which include several physically distributed components such as manipulator arms and sensors and, typically, is composed of periodic processes whose period is in the order of a few milliseconds. RK has also been used as the basis for the development of a similar system called Timix [LEE and KING 1988].

Chorus

Chorus, developed in France at INRIA by M. Rozier *et. al* and described in [ROZIER and OTHERS 1988], is a object-oriented distributed operating system which presents features suitable to the execution of distributed real-time applications. The primary idea behind Chorus was to bring distributed control techniques originated in the context of packet switching networks into completely decentralized distributed operating systems. Its kernel is called Nucleus, which provides communication functions based on message passing primitives to be used for unicast, multicast and broadcast communication. Also, it provides a priority-based preemptive real-time scheduler. Chorus is based on a modular architecture and it allows the creation of subsystems above its kernel which can offer services to applications in order to emulate another operating system. An Unix emulation subsystem was built upon its kernel in order to execute distributed real-time applications originally written to run over the standard Unix kernel. Chorus allows one user application to control directly an I/O component in order to handle hardware events in a timely manner.

A global identifier is provided for each resource in the system, therefore they will be found regardless of its physical location. A site is a concept of locality in Chorus where machines are interconnected by a communication network. A given site support many simultaneous resources and their virtual machines are protected as they allocate different address spaces. However, real-time events can break this protection scheme in order to maintain the execution of some applications bounded by a certain interval of time. This is a design decision made by Chorus in order to not delay context switching in the presence of real-time applications. However, resources or threads cannot migrate from one site to another.

Chorus had a commercial incarnation where it was used primarily for real-time applications in telecommunications systems.

MAFT

In [KIECKHAFFER et al. 1988], it is described MAFT, a distributed fault-tolerant system built to support real-time applications. A MAFT system consists of several semi-autonomous nodes connected by a broadcast bus network. These nodes are composed of two processors: the application processor (AP), which is responsible for running the application itself, and the operations controller (OC), which executes several functions such as scheduling and communications.

All the tasks to be scheduled should be periodic and the frequency of the tasks must be constrained within a binary frequency distribution [KIECKHAFFER et al. 1988]. The scheduling strategy is based on a deterministic priority-list. In this approach, each task is assigned a priority number and when a node becomes available the list of tasks is scanned in order of decreasing priority. The assignment of tasks to nodes is static for any given operating set. The scheduling function is replicated so that the scheduler of each node might schedule tasks for other nodes in the system. As the scheduling decision are defined off-line, a node's OC only allocate tasks to

its AP and it monitors the others nodes for responses of tasks which could impose precedence constraints. Then, task dependencies could affect the overall system performance even if the scheduling is calculated off-line. There were mechanisms developed in order for each node to be synchronized with the scheduler of any other node. When there are changes in the system operating set, a task reconfiguration process is called in order to redistribute the application workload. The main application of MAFT is in flight-control systems.

ARTS

H. Tokuda and C. Mercer developed a distributed real-time kernel called ARTS at the Carnegie Mellon University in 1989 as described in [TOKUDA and MERCER 1989]. The ultimate goal of ARTS was to provide users with a predictable and reliable distributed real-time computing environment [TOKUDA and MERCER 1989]. It is based on objects, and the real-time properties of a object are encapsulated within time constraints such as period and worst case execution time. Objects cannot migrate from a node to another during runtime but they might be shut down and then re-initialized in another node later.

ARTS' scheduler is time-driven, prevents priority inversion among communication tasks through the use of a priority inheritance protocol [SHA, RAJKUMAR and LEHOCZKY 1987] and it separates policy and mechanism. Therefore, users are free to change the system's scheduling policy during the boot or even during runtime among a pre-defined set of scheduling policies. These are: round-robin, FIFO, least slack time, rate monotonic, EDF, deferrable server, among others. Real-time tasks are classified as hard or soft and they might be periodic or sporadic. Hard real-time tasks are discarded if their timing constraints cannot be met whereas soft real-time tasks might run even if their deadline failed. Therefore, hard real-time tasks are scheduled first and the unused processor capacity is calculated to be used by soft real-time tasks. As described

in [TOKUDA and MERCER 1989], the main goal of the scheduler is

“not fairness, but rather to reduce the response time of aperiodic activities while guaranteeing that the periodic tasks’ deadlines are met.”

The communications subsystem of ARTS treats messages in the same way as processes, classifying them as hard or soft.

In order to predict the scheduling of real-time activities, a real-time toolset was built for ARTS. Within this toolset, the scheduling of a given set of tasks can be verified before runtime under a wide range of different real-time schedulers. These tools are also used for monitoring and debugging purposes.

HARTOS

HARTOS is a distributed real-time operating system which was part of a larger research project developed at the University of Michigan called the Hexagonal Architecture for Real-Time Systems (HARTS) [KANDLUR, KISKIS and SHIN 1989]. An important aspect of this work is the attention given to network communications, whereas dedicated communication processors were employed while the distributed kernel were executed on dedicated application processors[KANDLUR, KISKIS and SHIN 1989]. Messages were delivered within a bounded time, with deadlines defined on both a hop-by-hop and end-to-end basis. The communications subsystem assumes an unreliable transport network, therefore a reliable sequenced transport was developed for HARTOS. Therefore, each transaction is identified by a sequence number.

The kernel provided a preemptive priority based scheduler, where the priority of processes could be changed dynamically. Also, during critical sections, processes had the alternative to set their mode to non-preemptive.

Mars

In [KOPETZ et al. 1989] it is described Mars, which stands for Maintainable Real-Time System and it was developed around 1989 by H. Kopetz *et al.* Mars is a classical real-time distributed system in the sense that calculated all the scheduling offline, before runtime. It presented predictable performance under a specified peak load for tasks which imposed hard deadlines. Mars was written primarily to address real-time requirements of periodic systems. The architectural principles which guided the design of Mars are the following [KOPETZ et al. 1989]: capacity to handle peak loads with the same performance, transaction orientation, network structure with clustering, global time for synchronization of all the events in the system, interprocessor communication, end-to-end protocols where the application software controls the communication traffic in order to maintain real-time guarantees, TDMA media-access strategy in order to keep an uniform delay time to access the bus, fault-tolerance by active redundancy, multiple transmissions of messages, maintainability and extensibility, which are achieved through modularity. All these design principles were implemented by means of a operating system developed for these purposes. It runs an identical copy of the OS in each node of the system.

As the scheduling in MARS is calculated off-line, the set of tasks and its properties must be given in advance. It handles hard and soft real-time tasks as well as regular system tasks. The off-line scheduler is also responsible for the synchronization of the sending and receiving rates of messages whereby a flow control mechanism is implemented. Therefore, the CPU schedule is synchronized with the bus schedule. Later, an off-line algorithm based on a heuristic search strategy which calculates a function related to task urgency was implemented in order to improve schedule generation. Another interesting property presented in Mars is the estimation of the WCET of a task from its source code [KOPETZ et al. 1989]. The granularity of the system time is 1 μ s.

Alpha

Presented by E. Douglas Jensen and J. Duane Northcutt in [JENSEN and NORTHCUTT 1990], Alpha is a real-time operating system which can be classified as the opposite of Mars as it is intended to adopt more dynamic approaches in order to maintain a predictable response. It was developed at the Carnegie-Mellon University around 1990 [JENSEN and NORTHCUTT 1990]. Different from its predecessors, its design was not based on static determinism, but on a dynamic environment.

Alpha employs a global resource management approach, instead of other systems which were based on nodal autonomy for scheduling decisions. The Alpha kernel presented a programming model which was network transparent and whose principle abstractions were objects, operation invocations and threads. Invocations masked the physical environment and there was a protocol which handled communication errors. The unit of scheduling in Alpha is a thread. They are fully preemptable, even those executing within the kernel [JENSEN and NORTHCUTT 1990].

Alpha's scheduler is based on dynamic priority and it provides features for the clients to specify the timeliness constraints (deadlines), processing requirements (expected computation time) and the relative importance (priority) of each activity. Global importance and urgency characteristics are propagated throughout the system along with threads. A separate hardware could be allocated to run only the scheduler code.

At the time it was developed, Alpha was in the public domain for US government use. It was portable, being available initially on Concurrent and Silicon Graphics MIPS-based multiprocessor nodes running over a FDDI network.

3.1.2 Real-time distributed middlewares

ISIS Toolkit

Developed between 1985 and 1990 by Kenneth P. Birman at Cornell University, ISIS was a real-time distributed system with a strong focus on the devising of new communication primitives [BIRMAN and JOSEPH 1987]. It considers an environment where failures do occur and the system should react transparently to those failures.

ISIS adopts a logical approach to failure handling. All the processes belonging to the same process group will run a protocol in order to reach an agreement on when a failure occurred. This brings situations as, whether there is no evidence to the contrary, it can be assumed by all participants that a message delivery took place after or before a failure. Also, it will not allow any process to communicate with another one that is in a inconsistent logical state, therefore not permitting the propagation of failures. Besides ordinary group broadcast and atomic broadcast primitives, which both provides logical ordering guarantees of delivery, ISIS introduced a new primitive, called the causal broadcast primitive [BIRMAN and JOSEPH 1987], in which the causal relationship between messages are enforced. The communication architecture in ISIS is layered, and it provides an unified view of the sites which belong to a particular process group at some point in time. Messages exchanged between processes are queued and dispatched accordingly to a higher level algorithm or to some other factors like, for instance, the load on the network. ISIS provided a set of procedural tools in order to enable the application itself to manage process group communications directly.

Its relevance is strictly related to the extensive list of practical applications which were developed based on the ISIS toolkit, which include components of the New York and Swiss stock exchanges, distributed financial databases, telecommunications applications involving mobility, applications in air traffic control and space data collection, among many others [BIRMAN 2005].

Delta-4 XPA

The Delta-4 Extra Performance Architecture (XPA) developed by Paulo Verissimo *et. al* in 1990 and described in [VERISSIMO L. RODRIGUES and SPEIRS 1990] was an approach developed in order to extend the existing Delta-4 fault-tolerant and object-oriented distributed system for the support of real-time requirements. Delta-4 was an open project in the sense that supported open protocols and standards as the OSI (Open Systems Interconnection) reference model. The architecture supported the real-time concepts of priorities and deadlines and reflected these within its communication protocols. In this system, messages were delivered in the order of 1 to 2 milliseconds.

One of the novelties presented by Delta-4 XPA was the development of a new fault-tolerance scheme which was capable of meeting real-time requirements, whereas active replication is complex and costly and in practice it leads to unacceptably large maximum preemption times and in passive replication there is a delay involved in the provision of a new active host which might lead to larger delay times. This was solved through the proposal of a leader/follower model of replication [VERISSIMO L. RODRIGUES and SPEIRS 1990], where all the hosts execute active replication but only the leader is the responsible for taking all decisions regarding processing and synchronism of messages.

Its scheduler supported both hard and soft real-time tasks as well as background activities and it is based on two levels of precedence. Preemption of objects were allowed under pre-defined preemption points. Its native communication primitives included support for services ranging from atomic multicast through reliable multicast to unreliable datagram.

3.1.3 Common characteristics of real-time distributed systems

Given this extensive description of real-time distributed systems, it is possible to infer some common characteristics they possess:

- Hardware visibility for the application level
- Direct hardware control provided to applications
- Hardware dedicated to specific functions
- Execution of programs with specification for timing constraints
- Classification of processes as hard, soft real-time, regular and background ones
- Sequencing of messages through the utilization of native communication primitives
- Real-time scheduler
- Tools built in order to predict performance before run-time
- Global synchronization and uniqueness of identification of tasks and resources over the system
- Novel redundancy schemas built in order to maintain the required real-time performance

3.1.4 A taxonomy for real-time distributed systems

In this section it is proposed a taxonomy for real-time distributed systems as presented in tables 3.1 and 3.2 . The main classification is based on how the scheduling of tasks is performed: in a dynamic or static basis. This parameter is important because dynamic scheduling is a requirement for real-time cloud systems as the workload they are subject to may have a large variation. All the systems analysed support the scheduling of hard, soft and regular processes, but

for some of them only periodic tasks are allowed. The portability of all these systems is very low, depending on the development of new versions for each type of hardware. The exception is Chorus, which allows the creation of subsystems above its kernel. This is an important characteristic, as cloud systems provide virtualization techniques in order to provide portability. Most of them provide visibility of lower layers, which might improve the performance for applications in some cases.

Table 3.1: A taxonomy for real-time distributed systems - part 1

System	Scheduling Discipline	Scheduling Algorithm	Type	Process Support	Frequency of real-time processes	Portability	Communication facilities	Low layers' visibility
Alpha	dynamic, on-line, global resource management	time-driven, preemptive, dynamic priority	distributed object-oriented real-time operating system	hard and soft real-time, regular processes	periodic, sporadic	different versions of the kernel for different hardware platforms	yes, reliable message protocols	none
Chorus	dynamic	priority-based preemptive real-time scheduler	distributed object-oriented real-time operating system	hard and soft real-time, regular processes	N/A	different versions of the kernel for different hardware platforms, allows the creation of subsystems above its kernel	unicast, multicast, broadcast Messages, port-groups	hardware control for an user application
RNet	N/A	m-E-D-W-P for real-time processes, round-robin for regular processes	distributed real-time kernel	hard and soft real-time, alarm and background	periodic, sporadic	different versions of the kernel for different hardware platforms	real-time message passing	static modeling of the physical network
RK/Timix	off-line, static, advanced guarantees	timing constraints for real-time processes, priorities for best-effort ones	distributed real-time kernel	hard and soft real-time, regular	periodic only in the order of few milliseconds	different versions of the kernel for different hardware platforms	yes, based on timing constraints	direct control of devices by user applications

Table 3.2: A taxonomy for real-time distributed systems - part 2

System	Scheduling Discipline	Scheduling Algorithm	Type	Process Support	Frequency of real-time processes	Portability	Communication facilities	Low layers' visibility
MAFT	off-line, static, distributed, synchronized	deterministic priority-list	distributed real-time system	hard and soft real-time, regular	all tasks treated as periodic with a binary frequency distribution	different versions of the kernel for different versions of hardware	yes, internode communications	distributed and synchronized scheduling decisions
ARTS	off-line, static	time-driven, selectable by the user	distributed object-oriented real-time kernel	hard and soft real-time	periodic, sporadic	N/A	yes, schedulable as hard or soft real-time	
HARTOS	N/A	preemptive dynamic priority-based scheduler	distributed real-time operating system	N/A	N/A	none	yes, optimized network topology	network visibility
Mars	off-line, static, global synchronization	time-based	distributed real-time system	hard and soft real-time, regular processes	periodic	none	yes, TDMA-media access based, scheduled off-line	bus visibility
ISIS	N/A	N/A	middleware	hard and soft real-time, regular processes	N/A	different platforms	yes, with new devised communication primitives	network visibility
Delta-4 XPA	N/A	time-driven, 2-levels of priority	distributed object-oriented real-time system	hard and soft real-time, background activities	N/A	N/A	yes, 1 or 2 ms for message delivery, unicast and multicast	

3.2 Description of relevant real-time cloud systems

In this section, it is proposed a survey on real-time cloud computing systems highlighting its achievements in the light of the review just presented.

3.2.1 RT-Xen

Described in [Xi et al. 2011], RT-Xen is a proposal to perform real-time scheduling in the Xen hypervisor. Because of the full virtualization support offered by cloud systems, it can be part of a eventual real-time cloud. Xen is a very popular open-source virtual machine monitor (VMM) [BARHAM et al. 2003] and it uses a default credit scheduler whose aim is to achieve fairness among applications. RT-Xen implements a hierarchical real-time scheduler with one level for the hypervisor and another level for the guest OS. It is designed to dispatch soft real-time tasks and it instantiates a suite of fixed-priority servers, namely Deferrable Server, Periodic Server, Polling Server and Sporadic Server [Xi et al. 2011].

The main scheduling abstraction in Xen is the virtual CPU (vCPU), as it appears as a ordinary CPU to the guest OS. In the case of Xen's default credit scheduler, each vCPU is scheduled in a round robin fashion with a total amount of CPU of 30ms (quantum) for each process. RT-Xen is currently under development and its latest version is 2.0 which was released in October, 2013.

There are some other works which propose changes to the default credit scheduler in Xen, such as the one described in [LEE et al. 2010], whose main novelty is the supporting of soft real-time tasks in the Xen hypervisor with only minimal modifications to the default credit scheduler.

3.2.2 Hadoop-RT

Presented by Insup Lee *et.al* in [PHAN et al. 2011], HadoopRT is a prototype which presents real-time scheduling policies for Hadoop. Hadoop is a parallel processing platform based on the MapReduce paradigm whereby slave nodes process pieces of independent tasks scheduled by a master node. Reduce tasks calculate results from the output data of the map tasks. Hadoop's default scheduler can be: FIFO, fair or the so-called capacity, which is a variant of the fairness algorithm sorted by organizations. In the work of [PHAN et al. 2011], the default Hadoop's fair-scheduler is replaced by a variant of EDF, where preemption of tasks is not allowed. This variant presents enhancements to improve locality-awareness in order to fetch the data to be processed quicker. Data locality-aware is of special interest as data transfer rates are high in cloud environments and a measurement of the tradeoff between executing a job locally or moving its data remotely must be made in order to evaluate the timeliness constraints of one particular job. This metric is done experimentally, and no network condition or congestion is considered. Also, a fixed network topology with a constant minimum transfer rate is the only alternative proposed. Overload handling is also considered in order to prevent anomalies under heavy-loads (e.g. when one task missing deadline causes a chain of subsequent deadline misses). It is targeted for soft-real time applications, where one real-time metric, such as miss rate, is minimized. [PHAN et al. 2011]. Scheduling is performed on-line and jobs are submitted with a deadline. It is demonstrated in [PHAN et al. 2011] that its scheduler clearly outperform Hadoop's default schedulers. However, they considered some restrictions on their work which limit the scope of the proposed adaptations:

- Only non-preemptable tasks are allowed
- Each task has the same deadline and release time
- There are no failures in the infrastructure and
- No pipelining/speculative execution is allowed

Also, in [PHAN et al. 2011] there is an analysis of the support of current cloud infrastructures for applications which have strong timing guarantees with strict deadlines. They cite online log processing, traffic simulation, personalized recommendations, advertisement placement, social network analysis, real-time web indexing and continuous web data analysis as examples of this kind of application. It is shown that Hadoop and Amazon EC2 (Elastic Cloud Computing) are ill-equipped for this kind of applications.

3.2.3 Cloud service framework for real-time applications

In [KIM, BELOGLAZOV and BUYYA 2011], R. Buyya *et. al* presents a cloud service framework whose purpose is the creation of a virtual platform for real-time applications [KIM, BELOGLAZOV and BUYYA 2011]. It consists of three main components: real-time service model, which is composed of several real-time applications or subtasks; real-time virtual machine model, which can be classified as hard or soft real-time; real-time service framework, which defines a workflow in order for a user to run a real-time service with the following steps: requesting of the virtual platform, creation of the RT-VM from real-time applications, requesting of the real-time VM, mapping of physical processors and, finally, the executing of the real-time applications.

In this work, the default Xen credit scheduler is modified in order to reduce energy consumption while real-time deadlines are still met. It claims that the proportional sharing scheduling can guarantee the execution of real-time services of even HRT-VMs (hard real-time virtual machines) [KIM, BELOGLAZOV and BUYYA 2011]. It relies on the infinite capacity paradigm of cloud environments in order to meet real-time requirements.

3.2.4 RTF-RMS

Presented by S. Gorlatch *et. al* in [GORLATCH et al. 2012], RTF-RMS (Real-Time Framework - Resource Management System) is a resource management system which is based on the Real-Time Framework (RTF), targeted to match the requirements of Real-Time Online Interactive Applications (ROIA). RTF is a platform which provides monitoring and distribution mechanisms [GORLATCH et al. 2012]. Also, it provides communication facilities to applications. Typically, ROIA applications require soft-real time guarantees and the number of users can scale up or down very fast. The ultimate goal of RMS is to distribute ROIA sessions among virtualized Cloud Computing resources during real-time, using the monitoring data provided by RTF as input.

RTF-RMS consists in three different software components: cloud controller, which implements the communication interface with the cloud system; distribution controller, which implements the load-balancing algorithm for allocating resources in the cloud; ROIA Server controller, which enacts the load-balancing actions performed by the distribution controller. Based on a pre-determined metric, such as the deadline for some tasks, each of the following actions take place in the system: users are migrated from overloaded to underutilized servers; new servers are added to the environment; a resource is replaced by a more powerful one. In order to circumvent the time it takes to set up the virtual machines, RTF-RMS pre-allocates virtual machines in a resource buffer, which can be used immediately, as soon as a new request arrives.

It can be said that RTF-RMS works on a brute-force scheme, as it tries to allocated the resources in the best way possible and it monitors the responsiveness of the applications in order to load-balance them. However, it provides no hard or soft real-time guarantees.

3.2.5 RACE

Presented by B. Chandramouli *et. al* in [CHANDRAMOULI et al. 2012], RACE is a framework whose purpose is the specification and executing of distributed real-time applications in a particular topology called Cloud-Edge, which is characterized by several smart edge devices connected via the cloud [CHANDRAMOULI et al. 2012]. There is no direct communication between edge devices whereas all of them happen through the cloud. Workloads are distributed for processing between edge devices and a cloud. The main goal is to minimize communication between devices in order to save energy. Each device participating in the cloud runs a local scheduler, therefore making the system fully distributed running very cheap greedy algorithms, which runs off-line before runtime. Reoptimization might occur during runtime based on the statistics collected. Because of its distributed nature and cheap scheduling algorithm, it can scale to thousands of users without degradation in performance [CHANDRAMOULI et al. 2012]. It has been developed for soft real-time applications, such as location-aware coupon services, mobile multiplayer games and data center real-time monitoring applications.

3.2.6 Global-RT

Presented by I. Lee *et. al* in [XI et al. 2013], it introduces a multi-core VM scheduler called Global-RT for RT-Xen. It globally schedules hard real-time virtual machines as periodic servers or soft real-time virtual machines as deferrable servers. By global scheduler it is meant one which does not use a run queue per physical CPU, instead it presents a global run queue that is shared by all physical cores. It adopts an EDF scheduler with dynamic priority assignment, where the vCPUs, and not the tasks, are scheduled to the global queue. It is planned to be deployed with a real-time communication architecture designed for RT-Xen, as described in [XI and OTHERS 2013].

3.2.7 Scheduling based on time utility function model

As by his own citation, S. Liu *et. al* [LIU, QUAN and REN 2010] introduces a

"novel utility accrual scheduling algorithm for real-time cloud computing services."

In this work, the real-time tasks in a cloud are scheduled on-line and non-preemptively. A profit time utility function (TUF) and a penalty TUF are applied together in order to not only reward early completions but also to penalize abortions or deadline misses. The TUF model was first proposed by E. D. Jensen, C. D. Locke and H. Tokuda [JENSEN, LOCKE and TOKUDA 1985] and describes the value or utility increased by a system at the time when a task is completed. The penalty TUF is calculated because a missed deadline in a task means wasted resources as network bandwidth, storage space and processing power. Then, an approach is proposed in order to discard a task which will have its deadline missed as soon as possible. It is shown that this proposal outperforms other traditional scheduling approaches such as EDF, the Generic Benefit Scheduling (GBS) [LI 2004] and the Profit/Penalty-aware non-preemptive scheduling proposed in [YU 2010] (which is based on heuristics to derive a risk factor to identify the importance of tasks). In fact, the profit function of this new algorithm is no better than the more traditional ones. The gain is on decreasing penalties as some tasks are selected for discarding before execution.

3.2.8 Discussion of the characteristics of real-time clouds research

From this review, it is clear that the development of real-time clouds is still evolving. However, it is possible to notice the following characteristics in the research of real-time clouds:

- There is no proposed architecture in order for clouds to support real-time applications
- There are interesting approaches for the embedding of real-time schedulers in hypervisors and virtualization layers, but these are still isolated approaches
- Novel real-time schedulers adapted to cloud environments have been devised
- The frameworks presented do not guarantee the execution of tasks within strict required deadlines

3.2.9 A taxonomy for real-time cloud systems

In this section it is proposed a taxonomy for real-time cloud systems as presented in table 3.3. With the exception of the framework presented in [KIM, BELOGLAZOV and BUYYA 2011] by R. Buyya *et. al*, there is no support for hard real-time processes. The scheduling algorithms are all dynamic and RACE [CHANDRAMOULI et al. 2012] is the only one which presents a fully distributed scheduling approach, as it is very hard to maintain synchronism among all resources in a cloud system. Therefore, most of systems adopt a centralized scheduling discipline, where an hierarchical discipline is applied in some cases. The scheduling algorithm are based on modified EDF strategies or greedy strategies. EDF algorithms have been extensively proven in real-time distributed systems. Also, each approach presents some specific features related to cloud environments.

Table 3.3: A taxonomy for real-time cloud systems

System	Scheduling Discipline	Scheduling Algorithm	Type	Process Support	Frequency of Real-Time Processes	Communication Facilities	Cloud Specific Features
RT-Xen	hierarchical, dynamic	EDF	real-time hypervisor	soft real-time	periodic	none	vCPU scheduling
Hadoop-RT	on-line dynamic scheduling with no preemption	EDF	real-time cloud processing platform	soft real-time	periodic	none	map-reduce like applications, data locality-awareness, overload handling
Cloud service framework for real-time applications	global dynamic scheduling	credit scheduler modified	framework	hard and soft real-time	no contention allowed	none	soft and hard real-time virtual machines
RTF-RMS	on-line dynamic	greedy	framework	soft real-time with no guarantees	no restrictions	none	virtual machine migration, pre-allocation of virtual machines
RACE	fully distributed and dynamic	greedy	framework	soft real-time	no restrictions	none	elasticity
Global-RT	global dynamic scheduling	EDF with dynamic priority assignment	multi-core real-time scheduler	soft real-time	no restrictions	specific real-time communications architecture	vCPU scheduling
TUF	real-time scheduler	on-line scheduling with no preemption allowed	profit and penalty TUF	hard and soft real-time	no restrictions	none	tasks which will not be executed by its deadline are discarded earlier saving cloud resources

In the next chapter the requirements regarding the designing of a real-time community cloud will be presented.

Chapter 4

REQUIREMENTS OF A REAL-TIME COMMUNITY CLOUD

Given the extensive background presented regarding the development of distributed real-time systems, the research review regarding real-time clouds and the concepts explained before, this chapter identifies what are the requirements needed for the design of a real-time community cloud system.

The method adopted in order to list these requirements is called QFD - Quality Function Deployment, which is commonly applied in the development of products. The main two final objectives of this method is the system's compliance with all customer needs and the offering of new features which would exceed the customer's expectation. The requirements are first identified and classified into groups, which comprise requirements that share similarities. Thereafter, the requirements are ordered taking into account the customer's expectations. In the QFD method, it can assume three values:

- **Implicit:** the customer does not realize that this requirement is available, but if it is not present the customer will be upset
- **Ordinary:** the customer sees this requirement and considers it as essential to the system
- **Amazing:** the customer does not know that this requirement should be available but, if it is, it generates a value above the initial expectations

The customer, which is the central piece in the QFD method, here is envisioned as one of the owners of the community cloud who acts on behalf of some institution. He might hold technical administrative or more strategic responsibilities, as the development of scheduling algorithms. For instance, based in the case study presented in the next chapter, it could be a technical administrator of one participant of the market, i.e., a financial institution, a brokerage house, a bank or even an individual investor.

Following that, the priorities for these requirements are classified into High, Medium or Low. Prioritizing is done accordingly to the scope of this research. The more close to the objectives of the research, the higher the priority for the implementation of the requirements. Finally, the total effort involved in the development of the features needed in order to comply with each requirement is estimated as High, Medium or Low. This is an informal measure of the estimated total number of hours needed in order to implement a requirement. Priority and effort act as guidelines when one is implementing a new product but might be influenced by other business or strategic goals.

The requirements here proposed, both functional and non-functional, and its properties - function, type, customer's expectation, priority and effort - were gathered based on the examples listed in the bibliographical review presented in the previous chapter and on the issues which need to be solved regarding real-time applications and clouds as described in the chapter 2. Concepts. Also, it was based on informal conversations that this researcher established with some participants of the financial market and for some extent on my personal experience too.

4.1 Functional Requirements

In the QFD method, the first step is to break down the functional requirements into groups whose components share similar roles. Gathering the main functionalities which a real-time cloud must provide, the groups proposed in order to classify the functional requirements are the following:

- F-S: general requirements of the real-time cloud (table 4.1)
- F-U: user's requirements (table 4.2)
- F-Sch: requirements related to scheduling (table 4.3)
- F-Hwr: requirements related to hardware control and visibility (table 4.4)
- F-Pro: requirements related to programming capabilities of the real-time cloud (table 4.5)
- F-Com: requirements related to communication facilities provided by the real-time cloud (table 4.6)
- F-Ptb: requirements related to portability of programs in the real-time cloud (table 4.7)
- F-Vtl: requirements related to virtualization capabilities of the real-time cloud (table 4.8)
- F-Sec: requirements related to security (table 4.9)

Having these groups defined, the requirements are listed and classified in each of them as showed in the following tables. which compiles all these functional requirements, its classification's group, customer's expectation, priority and effort.

Table 4.1: Functional requirements - system's

ID	Requirement	Type	Customer's expectation	Priority	Effort
1	Coupling/de-coupling of a hardware resource on-line and dynamically	F-S	Implicit	High	Medium
2	Support for soft real-time virtual machines	F-S	Ordinary	High	Medium
3	Pre-allocation of virtual machines	F-S	Amazing	Medium	Medium
4	Global control with uniqueness of identification of resources in the real-time cloud	F-S	Implicit	High	Low
5	Agnostic user's access method	F-S	Implicit	Low	Low
6	Automatic updating of firmware and underlying software	F-S	Implicit	Low	Medium
7	Support for overload handling	F-S	Ordinary	High	Medium
8	Real-time-infrastructure-as-a-service support	F-S	Ordinary	High	Medium
9	Real-time-platform-as-a-service support	F-S	Ordinary	High	Medium
10	Real-time-software-as-a-service support	F-S	Amazing	Low	High
11	Storage of historical performance results	F-S	Implicit	Low	Low
12	Performance feedback with historical results with eventual reconfiguration of the real-time cloud	F-S	Amazing	Medium	High

Table 4.2: Functional requirements - user's

ID	Requirement	Type	Customer's expectation	Priority	Effort
13	Dynamic sharing of physical resources among users	F-U	Implicit	High	Medium
14	Dynamic sharing of virtual resources among users	F-U	Implicit	High	Medium
15	Graphical control and visibility of a pool of user's resource	F-U	Ordinary	Low	Medium
16	Graphical control of the performance of a user's pool of tasks	F-U	Ordinary	Low	Medium

Table 4.3: Functional requirements - scheduler's

ID	Requirement	Type	Customer's expectation	Priority	Effort
17	Preemption of tasks	F-Sch	Implicit	High	Medium
18	Use of different real-time scheduling algorithms chosen by the user	F-Sch	Amazing	High	High
19	Adaptive real-time scheduling algorithms	F-Sch	Ordinary	Medium	High
20	Heuristic on-line real-time scheduling algorithm	F-Sch	Ordinary	Medium	High
21	Hierarchical real-time scheduling algorithm for the physical and virtual resources	F-Sch	Implicit	Medium	Medium
22	Advanced guarantees	F-Sch	Ordinary	High	Medium
23	Support for soft real-time processes	F-Sch	Ordinary	High	Low
24	Support for hard real-time processes	F-Sch	Amazing	High	Medium
25	Support for regular processes	F-Sch	Implicit	High	Low
26	Support for dynamic priorities for regular processes	F-Sch	Ordinary	Low	Medium
27	Support for periodic real-time processes	F-Sch	Implicit	High	Low
28	Support for sporadic real-time processes	F-Sch	Ordinary	High	Medium
29	Global scheduler synchronization	F-Sch	Implicit	High	Medium
30	Physical and virtual CPU allocation granularity	F-Sch	Ordinary	High	Medium
31	Migration of processes among resources	F-Sch	Ordinary	Medium	High
32	Multi-core scheduling support	F-Sch	Ordinary	High	High
33	Scheduler's awareness of the locality of data	F-Sch	Amazing	High	High

4.2 Non-Functional Requirements

In this section, we identify and describe the non-functional requirements needed for real-time community clouds:

4.2.1 Timeliness

This is a property required and exhibited by real-time systems which guarantees responses to certain events always within the range of some pre-determined interval. For real-time clouds, it means that the cloud should guarantee the exe-

Table 4.4: Functional requirements - hardware control and visibility

ID	Requirement	Type	Customer's expectation	Priority	Effort
34	Direct hardware control by the cloud orchestrator	F-Hwr	Implicit	High	Low
35	Cloud orchestrator's visibility of hardware changes	F-Hwr	Implicit	High	Low
36	Cloud orchestrator's visibility of virtual network changes	F-Hwr	Implicit	High	Medium
37	Cloud orchestrator's visibility of physical network changes	F-Hwr	Implicit	High	Medium
38	Hypervisor's hardware control (once the hardware has been allocated to the hypervisor)	F-Hwr	Implicit	High	Low
39	Allocation and protection of a hardware resource for an exclusive function	F-Hwr	Ordinary	High	Low

Table 4.5: Functional requirements - programming capabilities

ID	Requirement	Type	Customer's expectation	Priority	Effort
40	Language provided for the programming of the cloud	F-Pro	Ordinary	Low	Medium
41	Language provided for the writing of real-time programs to the cloud	F-Pro	Ordinary	Low	High
42	Emulation of the behaviour of the cloud with synthetic workloads	F-Pro	Amazing	Medium	High
43	Emulation considering the underlying hardware	F-Pro	Ordinary	Medium	High
44	Emulation considering the underlying virtual network topology	F-Pro	Ordinary	Medium	High
45	Emulation considering the underlying physical network topology	F-Pro	Ordinary	Medium	High

Table 4.6: Functional requirements - communication capabilities

ID	Requirement	Type	Customer's expectation	Priority	Effort
46	Real-time communication primitives provided by the cloud	F-Com	Amazing	Medium	High
47	Unicast, multicast and broadcast capabilities	F-Com	Implicit	Medium	Medium
48	Reliable message delivery	F-Com	Implicit	Medium	Medium
49	Ordinary IP and Ethernet support	F-Com	Implicit	Low	Low

Table 4.7: Functional requirements - portability

ID	Requirement	Type	Customer's expectation	Priority	Effort
50	Different real-time operating systems support	F-Ptb	Ordinary	High	Low
51	Different regular operating systems support	F-Ptb	Implicit	High	Low
52	Hypervisor with real-time capabilities support	F-Ptb	Amazing	High	Low
53	Different guest real-time operating systems support	F-Ptb	Amazing	High	Low
54	Different guest regular operating systems support	F-Ptb	Ordinary	High	Low

Table 4.8: Functional requirements - virtualization

ID	Requirement	Type	Customer's expectation	Priority	Effort
55	Virtualization support of CPU resources	F-Vtl	Implicit	Medium	Low
56	Virtualization support of I/O resources	F-Vtl	Implicit	Medium	Low
57	Virtualization support of network resources	F-Vtl	Implicit	Medium	Low
58	Virtualization support with real-time capabilities of CPU resources	F-Vtl	Amazing	High	Medium
59	Virtualization support with real-time capabilities of I/O resources	F-Vtl	Amazing	High	Medium
60	Virtualization support with real-time capabilities of network resources	F-Vtl	Amazing	High	Medium

Table 4.9: Functional requirements - security

ID	Requirement	Type	Customer's expectation	Priority	Effort
61	Authentication mechanisms for users	F-Sec	Implicit	Medium	Low
62	Authorization mechanism for users	F-Sec	Implicit	Medium	Low
63	Privacy of users' data	F-Sec	Implicit	High	Medium
64	Confidentiality of users' data	F-Sec	Implicit	High	Medium

cution of real-time processes within a pre-defined deadline. It should handle two kinds of real-time processes:

- **Hard real-time:** once accepted into the system for processing, there is a guarantee that its deadlines must always be met

- Soft real-time: once accepted into the system for processing, they will be processed in order to meet its deadline or some other target measure related to time, but this will not be guaranteed

4.2.2 Capacity to handle peak loads

In order to deliver capacity to handle peak loads, the real-time cloud should present on-line mechanisms to adjust the allocation of resources to the workload being presented to the system. In a real-time system, admission control mechanisms are required in order to discard a task that will not meet its deadline, therefore not allocating resources unnecessarily.

4.2.3 Predictability

As described by A. K. Mok in [MOK 1983], predictability is the property that a system will always take the same time to execute a program given a certain input regardless of the system's utilization. This property is required in real-time clouds for hard real-time processes.

4.2.4 High Availability

The real-time cloud system should present built-in mechanisms in order to auto recover from failures.

4.2.5 Modularity

Modularity is a key requirement in any cloud system and it relates to the capacity of the system to absorb or remove resources from its configuration dynamically. Also, it provides the capacity to enable new features in the system without affecting others. These two requirements should be presented in

real-time clouds.

4.2.6 Isolation among users

As the system resources will be shared by many users, it should provide mechanisms in order to protect both the resources and the execution of tasks among each other. A real-time cloud system must not allow that one single user allocates all the resources available in the system. However, in divergence with ordinary cloud systems, a real-time cloud system should not aim at fairness of utilization or maximizing of the throughput, but at the number of hard real-time tasks which can be processed into the system within its deadline limits along with the number of soft real-time tasks processed which could meet their deadline or other real-time target measure.

4.2.7 Elasticity

Elasticity is the property of a cloud whereby a user can have its pool of resources increased or decreased dynamically in a very short interval of time. In public clouds or in hybrid private-public clouds, the user has no knowledge of the physical location of where its workload is being processed. For real-time clouds, elasticity must be provided too, but the system should be designed in a way where the requirements of timeliness for hard and soft real-time processes can be met.

4.2.8 Scalability

Scalability is the property which a real-time cloud must present in order to gracefully increase its total processing capacity without having to change its architectural principles and design.

4.2.9 Reliability

A real-time cloud must be reliable meaning that the system delivers the same performance even under situations where peak workloads and failures may arise.

Chapter 5

CASE STUDY: A REAL-TIME COMMUNITY CLOUD FOR A FINANCIAL TRADING APPLICATION

In this chapter a case study for the deployment of a real-time community cloud is proposed. This case study is related to a financial trading application.

5.1 Goals

The main goals which need to be achieved with the description of a case study in this research are the following:

- Demonstrate a feasible application suitable to the proposal of this thesis
- Enhance the understanding of the hypotheses described in this research, mainly the capability of handling real-time workloads by cloud systems
- Identify possible issues that may arise in the actual deployment of real-time cloud systems
- Specify and deploy a scheduling algorithm that demonstrates the real-time capabilities of the cloud system
- Specify and deploy a scheduling algorithm that demonstrates the performance gains for a given workload scenario typical to the application described

- Widen the visibility of this research bringing potential new ideas for the development of novel applications based on similarities with this case study

5.2 Scope

The scope of this case study has the following extent:

- An application is identified and described with the level of details necessary to demonstrate the goals listed in the previous section
- The typical manners of solving the problems addressed by this application are described. Conversely, the advantages brought by the real-time clouds are clarified
- The main parameters that need to be improved for this application are identified
- Typical workloads for these parameters are presented
- A real-time scheduling algorithm for this case study is developed
- This algorithm is applied to workloads by means of simulations

5.3 Real-time trading financial application

Consider a trading application, whereby financial institutions, brokerage houses - called participants - use to send orders of purchase or sale of securities (most commonly equities, futures and options) to stock exchanges on behalf of their customers. The decision to send orders to buy or sell securities is done based on a particular strategy whose inputs are, among others, the prices of the securities at an instant in time. These prices are received by the participants by means of a continuous flow of information called market data ¹, which is generated by the stock exchanges and advertises the absolute and relative prices of the securities being traded. Market data is comprised of two data channels:

¹<http://en.wikipedia.org/wiki/marketdata>

incremental channel, which provides the delta in prices for the securities since the last trading done and the snapshot channel, which advertises the absolute prices of all the securities. The incremental channel is aperiodic and updates the prices of all securities in real-time. The snapshot channel sends its information in a periodic rate and it adds some delay in comparison with the incremental channel.

At the stock exchange, the orders sent by the participants are stored in a table called the order book, which lists all bids (purchase) and asks (sale) for all the securities. A matching engine will verify if there is a match between these bids and asks in order to confirm or not a deal. This is illustrated in figure 5.1.

Figure 5.1: Trading environment

As the conditions of the market fluctuate, so does the number of orders traded. On busy days, the number of orders can increase to a very high rate. At the stock exchange side, every order should be processed in real-time. The same is true for the participant side, where the decision to generate a new order of buy or sell should be done in real-time.

5.4 Typical parameters for trading

Typically, tasks that need to run in real-time on the participants' side in order to perform trading to stock exchanges have the following characteristics:

- They are computing intensive as they perform many calculations for several different scenarios. Depending on the strategy of the participant, they might be data intensive too
- As most of the decisions are done based on the incremental channel information, which is aperiodic by nature, tasks are event-triggered and present arbitrary arrival times
- Their computation time may vary as their deadlines because these variables will be dependable on the strategy adopted by the participant
- Overload conditions may occur because tasks may arrive more frequently than expected or because their computation times may exceed their expected values. This situation happens when the market data rate is very high due to a particular condition of the market
- Commonly, trading is available during business hours, but there are variations between the opening and closing hours of the market for each product

Also, the following parameters are of interest for trading:

- Market data's bandwidth: typically in the order of Mbits/s
- Market data's number of messages' rate: this rate can reach thousands of messages per second and millions per day
- Time to process orders at the stock exchange side: typically in the order of microseconds
- Latency from participants to stock exchange venues: typically in the order of microseconds to milliseconds

- Jitter from participants to stock exchange venues: minimum, typically in the order of nanoseconds to microseconds
- Number of orders sent by a participant to a stock exchange: typically in the order of thousands per second. Limited by participant
- Numbers of orders that can be processed by a stock exchange system: typically in the order of hundreds of thousand per second and millions a day
- Fluctuations in the market: it can range from no order sent in one second to a burst of millions of orders in the next one. The stock exchange as well as the participant's system must have mechanisms in order to throttle this kind of behaviour when a certain rate limit is reached

As an example, table 5.1 shows some values measured at three different stock exchanges. BMFBovespa is the stock exchange which is responsible for the trading of equities, futures and options in Brazil. Its statistics can be found at its web site ² and the numbers in table 5.1 are related to equities traded at 01.07.2014. BATS Trading is the third largest US equities market (after NYSE and Nasdaq) and its statistics can be found at [BATS Global Markets, Inc 2012]. The numbers for BATS in table 5.1 are related to the BZX Exchange at 01.03.2012. CME (Chicago Mercantile Exchange) is the stock exchange in the US responsible for the trading of commodities, futures and options and its statistics can be found at [CME Group, Inc 2011]. Its numbers in table 5.1 are related to the Futures and Options market in June-2012.

There is a contrasting order of magnitude that each market deals with as the peak of messages per second for BATS is almost 45 times the peak for BMFBOVESPA. The peak of messages per second and peak of messages per millisecond (data available only for BATS) shows there are very high bursts in the rate of market data's messages, suggesting that this traffic may present a large variation over time.

²<http://www.bmfbovespa.com.br/market-data/estatistica.aspx?idioma=pt-br>

Table 5.1: Typical values for stock exchanges

Parameter / Stock Exchange	BMFBOVESPA	BATS	CME
Market Data - Total Number of Messages	10,779,788	N/A	N/A
Market Data - Peak of messages/s	8,045	361,739	37,000
Market Data - Peak of messages/ms	N/A	6,184,000	N/A
Market Data - Peak Bandwidth in Mb/s	5.3576	124	32
Market Data - Peak Bandwidth in Mb/ms	N/A	639	N/A

Customers pay different prices for different physical distances to the trading systems. That's why collocation spaces are the most valuable ones at stock exchanges' premises. Typically, each customer will deploy its own infrastructure at these collocation spaces, with the price for rental being proportional to the number of rack units allocated. As of today, NYSE (New York Stock Exchange) claims to deliver the lowest latency connection from collocated customers to its trading system - 75 microseconds round-trip time [NYSE Euronext 2011].

5.5 Typical infrastructure deployed at collocation facilities

Customers pay different prices for different physical distances to the trading systems. Facilities co-located to its premises are offered by stock exchanges to participants in order to provide the lowest feasible latency to the trading system³ and <http://www.eurexchange.com/exchange-en/technology/co-location-services>. Typically, a participant will deploy the following infrastructure at a collocation facility: a network connection which provides low latency, servers with multi-core CPUs, a relatively small storage system which will store all the orders sent and deals done and a number of backdoor connections to other venues which can use mixed information in order to have a global strategy of trading, which is called market arbitration [Johnson 2010]. The participant's system must work on real-time, on the order of microseconds to milliseconds and be as predictable as possible in order not to lose opportunities to buy/sell securities accordingly to his strategy. As each customer deploys its own infrastructure, the stock exchange venue has to provide enough physical space and power supplies. The same

³<http://www.nasdaqtrader.com/Trader.aspx?id=colo>

market data information must be received by every customer and this is done based on multicast techniques in order to save bandwidth.

Figure 5.2 depicts a typical infrastructure deployed by participants in a collocation rental space at a particular stock exchange's premises. Each participant deploys its own dedicated servers/CPU's, storage and backdoor connections. There is one (sometimes two for the sake of redundancy) network connection to the stock exchange by participant, whereby it will receive market data, send orders of purchase/sale and receive confirmations whether the deals were done.

Figure 5.2: Typical infrastructure at a stock exchange's collocated rental spaces for trading

5.6 Proposed real-time community cloud infrastructure

Given the scenario presented in this case study, it is proposed the deployment of a real-time community cloud infrastructure. The users of the system are the participants of the market, and each of them contributes with a set of hardware piece which is added to the real-time community cloud. This environment is depicted in figure 5.3.

Figure 5.3: Real-time community cloud infrastructure

5.6.1 Hard real-time tasks

In this environment, hard real-time tasks are characterized by the processing of a certain amount of calculations given a certain market situation which could potentially match one participants' strategy but it should be calculated very fast and with predictability before the market's picture changed. In a real-time

cloud community infrastructure, more resources are available than in standalone deployments. This brings the potential for participants to perform more complex calculations still in a predictable manner and with a reasonable cost. Also, ultimate hardware technologies could be deployed transparently and bring advantages to all participants.

5.6.2 Soft real-time tasks

In this environment, soft real-time tasks are characterized by the processing of a certain amount of calculations given a certain market situation which could potentially match one participants' strategy but, if the market's picture changed, these calculations should not be discarded as they still could bring some value to the participants' strategy, differing in this way from hard real-time tasks.

5.6.3 Regular tasks

In this environment, regular processes are characterized by tasks such as those which perform background calculations, copy data to a local storage, communicates with other venues via backdoor connections in order to feed arbitration strategies as explained before, among other functions.

5.6.4 Control of resources sharing

Mechanisms of fairness among users must be devised in order to protect the system from being allocated to just one participant through the Admission Control module. In particular, the module responsible for the executing of hard real-time tasks, which is constructed through the sharing of hardware among the participants, must employ these mechanisms.

5.6.5 Collaboration among participants

The primary goal of such communities is to provide a better performance for its members in comparison with a standalone deployment. Another goal is to have a lower infrastructure cost versus performance ratio, therefore allowing them to save money but still being able to achieve performance targets. Each of the companies involved in the community cloud present different strategies to make the highest profit in financial environments. If they share similar strategies, they might be competing for the same resources during the same interval of time. These strategies are comprised of one or several algorithms which intend to understand the market situation and send orders to stock exchange's trading systems. The picture of the market varies overtime, and the difference of values for any security X in instant A and instant B is what is used to make profit. Consider a particular security X which is traded by a stock exchange, whose price might be increasing during, for instance, the first hour of a trading day. Conversely, in the following hour, it might decrease its price very fast. Company A developed a strategy where it will make money if the price of this security is increasing. On the other hand, Company B developed a strategy where it will make money if the price of this security is decreasing. They do not know about each other's strategy and do not need to reveal it to each other. Some other companies which are participating in this same market have the same strategy as Company A and some other companies have the same strategy as Company B. Both Company A and B want to be the fastest to send orders to the trading system if the price of the security X is increasing or decreasing, respectively, so they can realize their strategies. As their strategies are not conflicting with each other, they will borrow computational resources to each other depending on the market situation. The final expected result is such as both will have a better performance acting together than in a standalone fashion.

It is envisioned that these agreements between participants would follow the same dynamism of the stock exchange market, where cooperation and trust levels could be very high during a certain time interval but fall to very low

values during particular market hours. Also, one group of participants in the collocation facility could arrange itself in a way that it could be more efficient than other participants that operate in a standalone fashion and are not part of the community cloud.

An interesting question that arises is how to promote the adherence to the community cloud. The main advantages that the participants might have in comparison to a standalone deployment should be advertised, mainly the possibility of having more performance with less investment on infrastructure which derives in lower operational expenditures. There could be a challenge if the participants do not believe that the community cloud provides total isolation of data among users. Also, participants which share similar strategies might not perceive the expected performance gains as they will be competing for resources.

5.7 Use Cases

In this section the main use cases envisioned for this environment are described. However, they are not meant to be an exhaustive list.

5.7.1 Actors

The main actors who interact with this real-time community cloud system are:

- Participants of the market, which are the parts interested in joining the community cloud
- Stock Exchange, which is the part responsible for the facilities at collocation
- Mediator, which is responsible for the adhering or removal of participants and the solving of conflicts among them. It might be represented by an external entity in order to maintain neutrality
- Administrator, who is responsible for the administration of the community cloud. It might be represented by an external entity in order to maintain

neutrality

- Participants' application, which implements the strategy of each participant

5.7.2 List of use cases

The main use cases envisioned for this environment are:

- Discovery/mapping of the physical topology of the environment
 - Actors: Participants, administrator, participants' application
- Execution of a hard real-time task
 - Actors: Participants, participants' application
- Execution of a soft real-time task
 - Actors: Participants, participants' application
- Execution of a regular task
 - Actors: Participants, participants' application
- Provisioning of a new participant
 - Actors: Participants, stock exchange, mediator, administrator
- Insert/withdraw of physical resources in the community cloud to/by a participant
 - Actors: Participants, stock exchange, mediator, administrator
- Insert/withdraw of virtual resources in the community cloud to a participant
 - Actors: Participants
- Definition of restrictions of the utilization of a resource by a participant
 - Actors: Participants, mediator

- Emulation of the performance of an application given a certain hardware configuration of the real-time cloud
 - Actors: Participants, administrator, participants' application
- Allocation of resources for a virtual machine
 - Actors: Participants
- System's configuration
 - Actors: Administrator
- System's accounting
 - Actors: Administrator
- System's failure handling and control
 - Actors: Administrator
- System's configuration
 - Actors: Administrator
- System's performance analysis
 - Actors: Administrator, mediator, participants, participants' application
- System's authentication and authorization
 - Actors: Administrator, mediator, participants

Chapter 6

ARCHITECTURE OF A REAL-TIME COMMUNITY CLOUD

In this chapter an architecture for a real-time cloud is proposed within the boundaries of the following constraints:

- The service provided by this real-time cloud is Infrastructure-as-a-Service (IaaS)
- The deployment model adopted is the community cloud

Therefore, the real-time cloud orchestrator architecture and its embedded mechanisms are described.

6.1 Services

The service provided by this community cloud is infrastructure as a service for real time applications - which we call, for short, IaaS-RT. As an IaaS platform, it should provide: a computing infrastructure which can schedule and execute real-time applications in a timely manner, a network infrastructure which can deliver packets as predictable as possible and a storage infrastructure which can store relevant data. It might be deployed through the use of virtual machines or physical machines that could be allocated depending on the performance expected. IaaS services provide higher flexibility, faster service delivery, better

performance and dynamic scaling [RIMAL and CHOI 2012].

Eventually, it could evolve to a Platform-as-a-Service (PaaS) real-time cloud, where the interested companies would share the same platform in order to develop its own algorithms and perform trading, without having to care about the underlying hardware and operating system details.

6.2 Community clouds

The requirements identified in the previous chapters will be realized through a cloud architecture which adopts the concept of community clouds as presented in the chapter 2, Concepts.

We adopt probability values, just as described by Briscoe and Wilde in [BRISCOE and WILDE 2006], as the chance that company A might offer resources to company B as demanded. This probability values may vary over time. Users will make agreements with other trusted users in a dynamic fashion and the probability values chosen will reflect these agreements. So, company A will borrow resources as demanded by company B based on these probability values. These values should be taken into consideration when a scheduling decision is being done in the system.

Also, probability value is a function of another one, called trust, which measures the level of confidence over time which two companies might exchange resources in a peer-to-peer fashion. It makes sense to offer more resources in the community cloud to trusted companies, as long as they do not influence negatively the result of company A. On the other hand, it is the interest of company A to make agreements with other companies, as during a particular point in time, it would undoubtedly let company A with a far competitive advantage over its competitors without having to invest more money in hardware and software in order to achieve its targets.

In this way, as depicted in figure 6.1, the users will form a logical community cloud in order to mutually cooperate and use resources from each other. In figure

6.1, P_{xy} is the probability level that X will offer resources to Y. This value is unidirectional meaning that not necessarily P_{xy} equals P_{yx} . Also, the sum of P_{xi} , where i varies from 0 to $n-1$, equals 1.

It is envisioned that these agreements among users could present a high variation in time, as cooperation and trust levels could be very high during a time interval but fall to very low values during other hours.

Figure 6.1: Community Cloud Environment

6.3 Tasks

The tasks processed by this real-time community cloud are computing intensive. For the case study presented in the previous chapter, they will perform simulation of several mathematical scenarios using the prices and amount of the securities listed in the order book of a stock exchange as input. These scenarios implement the strategy algorithm of each participant in the market. As its output, these tasks provide an offer with the type of order (buy, sell) for each type of security with its respective amount to be sent to the stock exchange. When sent to the stock exchange, this offer might be converted into a deal. Another scenarios of real-time applications may share similar patterns for real-time tasks.

6.4 Comparison to grids

Despite some similarities, the community cloud proposed is different from grid architectures in the following aspects [FOSTER et al. 2008]:

- It provides only partial visibility of the underlying hardware to the users
- It can be scaled up or down on demand, depending on the users' willingness to provide or remove hardware from the cloud
- The resources of the cloud are shared among all the participants
- Jobs from different participants execute in parallel at the same time. Therefore the job of one participant does not have to wait until all the resources are freed only to him
- Virtualization is more often applied than in grids, partially because the resources must be shared among users

6.5 Real-time orchestrator architecture

It is proposed an architecture for an orchestrator (or VIM - Virtual Infrastructure Manager) which will realize the features that comply with real-time requirements. In a cloud system the orchestrator is the central module as it is the one responsible for the allocation of resources.

The orchestrator proposed is composed of the following modules, as indicated in figure 6.2:

- Task reception module: it is responsible for task reception in the system and for the maintenance of historical data regarding the workloads which the system has been subject to. This historical data will be used to select the best real-time scheduling algorithm for the cloud environment at a particular point in time. Also, it will send Quality of Service (QoS) parameters to the QoS module described in the next item

- **Quality of service module:** it evaluates the QoS parameters contained in the metadata of each task. It is composed of the following submodules: admission control, prioritization mechanisms, guarantee of execution mechanisms (with thresholds for maximum utilization) and signaling for the scheduling module of the prioritization of tasks. It receives feedback from the performance modules in order to dynamically adjust its policies
- **Scheduling control module:** it is responsible for the scheduling decisions. It will take as inputs the metadata of each task and the policies defined by the QoS module. With the information provided in the historical data of the task receiver module, it adapts the scheduling algorithm policy. It can discard tasks if necessary. It might take communication requirements among tasks as input too in order to minimize the communication costs
- **Run-time environment module:** given a set of tasks scheduled, this module will provision and execute them in the selected resources
- **Physical environment performance control module:** it is responsible for the reading, compiling and interpretation of the performance of the physical infrastructure resources. The outputs of this module - general system health, performance metrics - will be taken as inputs by the run-time environment module and the QoS module.
- **Virtual environment performance control module:** it is responsible for the reading, compiling and interpretation of the virtual environments created. The outputs of this module - general virtual environment system health, performance metrics - will be taken as inputs by the run-time environment module and the QoS module.

6.5.1 Task reception module architecture

The task reception module, depicted in figure 6.3, is composed of the following components:

Figure 6.2: Real-Time Orchestrator Architecture

- Task reception/compilation: responsible for task reception. It does an initial verification of data and metadata integrity. It stores a task to further send it to the scheduling control module
- QoS parameters compilation: it reads the QoS requirements for each task and, if necessary, translate it to a common format
- Workload historical data: it stores, for a certain time interval, a history of the workload that the system has been subject to. This data may be used to decide which scheduling algorithm to adopt. An open question here is how often this verification will be done. If the interval is too large, the system may react slowly to changes and the performance might be suboptimal; if it is too short, more resources will be consumed and the system may not reach a steady state

Figure 6.3: Task reception module - Architecture

6.5.2 Quality of service module architecture

The quality of service module, depicted in figure 6.4, is composed of the following components:

- QoS parameters reception: it receives and stores in a buffer the QoS requirements for each task which are described in the task's metadata
- QoS policy: it defines the QoS policy to be used by the orchestrator. For real-time applications, the following mechanisms are required: guarantee

of task execution in a certain deadline, prioritization and discarding of tasks. As this module will receive feedback from the performance control modules, it can adjust itself and modify its policy dynamically

- Admission control: it does an analysis to accept or not a new task into the system according to the QoS policy defined above
- Resource reservation control: it does a dynamic reservation of resources according to the QoS policy. As this module will receive feedback from the performance control modules, it can adjust itself and modify its policy dynamically. During peak times, it can either reserve no resource at all (so leaving the system working in a best-effort manner) or can reserve a minimal number of resources to only a pre-determined number of tasks
- Prioritization control: it defines the level of prioritization and how each task will be allocated to it, according to the QoS policy
- Maximum and minimum resource allocation: it controls the maximum number of resources a task can allocate. Also, it guarantees a minimum number of resources which a task can allocate to itself
- QoS directives: contains all the QoS definitions
- QoS signaling: it translates all the definitions above in a message which will be sent to the Scheduling Control module

6.5.3 Scheduling control module architecture

The scheduling control module, depicted in figure 6.5, is composed of the following components:

- Reception: it receives the data and metadata of the tasks which have been in the Task Reception module and store them in a buffer until the analysis to select the scheduling algorithm (next step) is performed

Figure 6.4: Quality of service module - Architecture

- Scheduling algorithm's selection: using as input both the workload historical data and a list of pre-defined algorithms with different parameters, it does an analysis to select the best scheduling algorithm. It can be the same adopted for the execution of the previous task or it might be modified depending on the workload. In this way, the system is dynamic and can adapt itself to different workloads at different intervals in time
- Real-time scheduling algorithm with prioritization: with the data and metadata of the task, the scheduling algorithm to be used and the signaling of the level of prioritization from the QoS module, it will schedule the next task to be executed by the system. It is proposed an hierarchical scheduling mechanism in order to guarantee the execution of hard real-time tasks. This mechanism is explained in 6.7. Also, the scheduling algorithm should take into account the communication requirements among tasks as described in the tasks metadata. The probability values for the borrowing of resources from one participant to another, as described in the chapter Case Study, should be taken as another input of the scheduling algorithm

- Optimization of network topology: depending on the output of the scheduling algorithm and the actual network configuration, this module provides the capability to optimize the network topology - either physical or virtual - according to the communication requirements of tasks. This might be accomplished using the OpenFlow protocol [BALAKRISHNAN et al. 2011]
- Queue manager: manages the high-priority queue, which is used to execute hard real-time tasks, and delivers the other tasks to the users' queue respectively, where they might be re-queued depending on the scheduling algorithm selected by the user

Figure 6.5: Scheduling control module - Architecture

A dedicated hardware of the real-time community cloud might be exclusively allocated to the scheduling control module.

6.5.4 Run-time environment module architecture

The run-time environment module, depicted in figure 6.6, is composed of the following components:

- Task reception: receives the next task ready to be executed and a pointer to the resources where it should be provisioned
- Provisioning: it does the provision of physical or virtual resources/machines as required
- Execution: it initiates the execution of tasks and monitor them until they are finished

6.5.5 Physical environment performance control module architecture

There are two performance modules: physical and virtual. For both of them the objective is to monitor meaningful performance parameters concerning the requirements of real-time applications. It is composed of the following components as depicted in figure 6.7:

- Performance data: it is the module responsible for the reception of performance data extracted from the real-time infrastructure
- Performance data compilation: it does the translation and formatting of performance data to further analysis and processing
- Analysis of performance data: component responsible for the analysis of performance data in real-time. After this analysis, these data are sent to feedback and potential reconfiguration of other modules
- Performance policy: it is a policy that might be defined by a user or the administrator which will contain all the performance parameters and thresholds that must be analysed

Figure 6.6: Run-time environment module - Architecture

Figure 6.7: Physical environment performance control module - Architecture

6.5.6 Virtual environment performance control module architecture

Depicted in figure 6.8, it is similar to the physical environment performance control module as it is composed of the same components. However, it might be instantiated for each customer and have a performance policy defined for each of them.

Figure 6.8: Virtual environment performance control module - Architecture

6.6 Communications architecture

The communication primitives provided by the real-time cloud should offer both unicast and multicast messages as well as broadcast. It should offer reliable services which could guarantee ordered delivering of messages or just unreliable services which trust in the underlying communication infrastructure.

6.7 Mechanisms

In this section some mechanisms which should be deployed by this architecture are presented. They are key to deliver the requirements listed in the preceding chapter.

6.7.1 Hierarchical scheduling

In order to match the requirements of real-time applications and guarantee the execution of hard real-time tasks, an hierarchical scheduling model is proposed, as depicted in figure 6.9. In this model, the Global Scheduler will select

Figure 6.9: Hierarchical Scheduling Mechanism

hard real-time tasks for immediate execution on plain hardware, without any virtualization layer. Soft real-time tasks and regular tasks should be forwarded to each user's queues where another scheduling decision should take place. Each user may have virtual or physical resources available for the allocation of tasks and it has the freedom to select his own scheduling algorithm. In addition to that, each user should be able to replicate the modules defined for the global system architecture within his own scope: task reception, quality of service, scheduling control, run-time environment, physical and virtual performance control modules, creating in this way a recursive system which could offer the maximum flexibility and modularity guaranteeing the best real-time performance for the cloud. An hierarchical scheduling mechanism could be deployed by the user, where, for instance, soft real-time tasks would be allocated to physical hardware

and regular tasks to virtual hardware.

Regarding the scheduling algorithms which could be selected by the Global Scheduler, clearly a real-time scheduling algorithm which takes into account the deadline of tasks is required.

6.7.2 An heuristic-based scheduler for a community cloud designed to support real-time trading applications

In this section a scheduling algorithm is proposed for the case study described in the previous chapter.

Presumptions

Most cloud environments present scheduling mechanisms which are not suitable to schedule real-time applications, as their main target is to increase the total throughput of tasks executed during a certain time interval [PHAN et al. 2011], [ENDO et al. 2011], [BITTENCOURT, MADEIRA and FONSECA 2012], [WOLF et al. 2010], [ZAHARIA et al. 2010], [KIM, BELOGLAZOV and BUYYA 2011], [LIU, QUAN and REN 2010], [GORLATCH et al. 2012].

In order to match the requirements of real-time applications, scheduling algorithms which take as input the deadline of tasks have been proposed. However, as described in [BUTTAZZO 2005], the problem of scheduling real-time tasks is NP-hard, therefore the calculation for finding a feasible schedule is too expensive. To solve this kind of problem, algorithms based on heuristic functions have been devised. In [STANKOVIC and RAMAMRITHAM 1987] and [STANKOVIC and RAMAMRITHAM 1991], John A. Stankovic proposed an algorithm based on a heuristic function H which was adopted in the real-time kernel called Spring. In the Spring kernel, the function H can assume many different values such as: arrival time (as in first come first served), computation time (as in shortest job first), deadline (as in EDF), among others. In this research,

we will propose a scheduling algorithm which resembles the one devised for the Spring kernel but which adopts an heuristic function H appropriated to the scenario described in the case study.

Data Structure

As described in [STANKOVIC and RAMAMRITHAM 1991], the ordering of a set of tasks in order to find a feasible schedule is a search problem; the data structure most used is composed by a search tree. A leaf of this search tree is a complete schedule, while a vertex between the root and a leaf represents a partial schedule. An heuristic approach is adopted in order to avoid performing an exhaustive search which would consider every path in the tree.

Therefore, two data structures are maintained:

- Partial schedule (search tree)
- Set of available tasks, which is sorted by tasks' deadline. This structure is comprised of tasks yet to be scheduled

Algorithm

In the proposed algorithm, the tasks to be scheduled are characterized by:

- Task arrival time
- Task deadline
- Task worst case execution time (WCET)
- Task resource requirements
- Task earliest start time, which indicates when the task can begin execution taking into consideration the results of the scheduling and the available resources

- An heuristic value H derived from a particular heuristic function

A detailed step-by-step of this search algorithm is described:

1. *Start with an empty schedule*
2. *If the set of available tasks is empty, return with no result*
3. *Choose the next task in the set of available tasks with the lowest heuristic value H . Insert this task in the end of the schedule*
4. *Check if the partial schedule is a strong feasible schedule. If it is, go to step 5. If it is not, withdraw the last task from the end of the schedule and go back to step 2 (backtrack)*
5. *Withdraw the task selected in step 3 from the set of available tasks. If the set of available tasks is not empty, go back to step 2. Otherwise, return with the full feasible schedule*

Heuristic function

The heuristic function H to be adopted in the algorithm just described can assume the following simple forms:

- Minimum deadline first, as in EDF
- Minimum task arrival time, as in FIFO
- Minimum worst case execution time, maximizing throughput
- Minimum earliest start time

However, we propose in this research that an heuristic function based on the dynamics of the case study presented should be devised accordingly to the participant's strategy for the trading of securities in real-time at stock exchanges. As an example, consider that securities A, B and C are being traded by participant 1 and securities D, E and F are being traded by participant 2, which are part of the community cloud. A trading strategy is defined such as the maximum gain being proportional to the maximum normalized difference between the last and actual value for a particular security. If this normalized gain is greater or equal than 1, the maximum number of orders should be sent to the trading system as fast as possible. Then it is possible to derive the following:

1. Heuristic function H is characterized by a value N , being N the normalized value measured for a particular security X at an instant in time

2. The scheduling algorithm should order the tasks for processing based on this value N

Some possibilities arise from these considerations. The real-time cloud system could adopt the following:

- If N is greater than 1, the task is a hard real-time one. Otherwise, it is a soft real-time one
- One task is necessary for the processing of each security
- The WCET of each task is linear for all the tasks, independently of the value of N
- The deadline of the tasks is inversely proportional to the value N of each security, but does not follow a linear function

Also, this simple heuristic function H could be composed of other values, such as $N \cdot \text{WCET}$ or $N \cdot \text{deadline}$.

The final objective of this scheduling algorithm is to maximize the value of H for all the calculations made. From a participant's perspective, it means that it should achieve, for his strategy, higher values of H in comparison with a standalone deployment. Therefore the sum of all the values of H for the whole community cloud should be higher than the sum of the value of H achieved with the deployment of standalone infrastructures for each participant.

Complexity analysis

When a new task arrives, the insertion of this task in the set of available tasks is $O(N)$, where N is the task set's size.

In the worst case, the algorithm can backtrack as many times as are the number of tasks which remain to be scheduled, which brings a worst case complexity of order $O(N)^{(2)}$.

6.7.3 Adaptive Topology Mechanisms

In HPC environments, the physical network topology is usually designed taking the application's requirements as the input. So, the physical network usually does not present performance problems to the applications. Conversely, it provides no flexibility as the hardware is tightly coupled with the software. In a cloud environment, flexibility is the rule. Then, the physical network might become a bottleneck for the deployment of real-time applications. R. Aversa [AVERSA 2011] suggests that the main performance issue of cloud environments is not the virtualization overhead, but the poor network connectivity between virtual machines which might bring latency and jitter issues.

In order to solve performance issues, most cloud environments propose the migration of virtual machines from one host to another (sometimes in different sites), as proposed in [DEVINE, BUGNION and ROSENBLUM 2002], [BARHAM et al. 2003]. However, this mechanism could bring more performance problems as virtual machine image files are large, therefore requiring a

long time interval where the application would not be available and the building of a parallel network only to perform migrations, making the system more expensive.

In order to avoid this, it should be developed a mechanism where the virtual and eventually even the physical network should be adapted in order to provide the best connectivity topology for the environment and its applications. It is envisioned that the concept of SDN (Software-Defined Networking) could be used, where a central controller would verify the performance of the applications according to the topology of the network and rearrange it as necessary. Protocols such as OpenFlow [BALAKRISHNAN et al. 2011] could be used to control both the physical and virtual topology.

6.7.4 Pre-provisioning of Virtual Environments

The delay required in order to provision virtual machines in cloud environments is described by S. Gorlatch in [GORLATCH et al. 2012]. This remains a challenge for environments which require performance, mainly the ones requiring real-time performance. For real-time applications, the time required to provision virtual machines is not acceptable. Then, mechanisms which can guarantee the pre-provisioning of virtual environments and its virtual machines should be considered for real-time clouds.

Chapter 7

SIMULATIONS

In this chapter scenarios for simulations are defined and executed in order to demonstrate two main points of this research:

- The capability of the real-time cloud to meet the requirements of real-time applications
- The feasibility and efficiency of the scheduling algorithm based on heuristics described in the previous chapter

According to the architecture proposed in the previous chapter, the scope of the simulations refer to the following items:

- Task reception module: the modules task reception/compilation and QoS parameters compilation are implemented. The module workload historical data is not implemented
- Quality of service module: the modules QoS parameters reception, QoS policy, Admission control, Maximum/minimum resource allocations, Prioritization control, QoS directives and QoS signaling are implemented. The module Resource reservation control is not implemented. Also, there is no feedback received from the performance control modules as these are not implemented

- Scheduling control module: all the modules are implemented but the features related to the optimization of the communication flows among tasks which could lead to changes in the network topology. The scheduling algorithm is manually selected for each simulation and not automatically adapted as the workload historical data module is not implemented. Also, the probability values that a participant may borrow resources to other participant as described in the chapter Case Study is always set to one, i.e., there is no control over which participant is using what resource
- Run-time environment module: all the modules are implemented
- Physical environment performance control module: not implemented
- Virtual environment performance control module: not implemented

7.1 Simulator

A simulator, written in the C language, was developed during this research in order to demonstrate the two capabilities above. The advantage of a program such as this is that it can simulate the behaviour of the system under many different workloads in a timely manner. Therefore, conclusions about the system conception and construction can be drawn earlier in the development process. Also, several scenarios of interest in this research can be created. However, the use of a simulator does not preclude the task of running exhaustive tests over a prototype or over a real environment, as all the environment variables can affect the performance for real-time tasks.

The simulator presents a structure whereby tasks are offered to be processed by the CPU resources over a timeline. These tasks arrive into the system according to a statistical distribution and are scheduled to be executed by the algorithm selected by the user. Parallel tasks are allowed; they are broken into smaller tasks for simultaneous processing. The simulator has an internal discrete counter which emulates a clock. A single queue is used in the system where tasks from all users are received, scheduled and then dispatched. Several properties of cloud systems

are presented as variables to the simulations, as virtualization overhead and the oversubscription factor. It is assumed that there is no delay in the provisioning of the virtual machines for the processing of the workloads. The figure 7.1 details the structure of the simulator.

Figure 7.1: Simulator - Structure

The tasks to be processed hold the following properties:

- They are classified in hard real-time, soft real-time or regular ones
- Hard and soft real-time tasks have a deadline by when they should be completed
- A task has a pre-defined worst-case execution time (WCET), which is the time necessary for the execution of a task
- A task has an arrival time, which is calculated accordingly to a statistical distribution
- Tasks can be split to be executed simultaneously on two or four CPUs. Conversely, some tasks can not be split and should be processed by a single CPU only

These tasks can be executed by one of the following algorithms:

- FIFO. First-in-First-out, where tasks are executed one after another accordingly to the order of entry in the system
- FIFO with hard real-time prioritization, where the FIFO discipline is maintained but hard real-time tasks are allocated only to physical CPUs
- FIFO with pCPUs allocated only to hard real-time tasks, where the FIFO discipline is maintained and physical CPUs are reserved for the solely execution of hard real-time tasks. Therefore, the physical CPUs are idle if there is no hard real-time task to process
- FIFO with Admission Control, where the FIFO discipline is maintained and hard real-time tasks are processed only if there are available resources in order to complete their deadline. Otherwise, they are discarded without processing
- EDF with Admission Control, where the Earliest-Deadline-First (EDF) algorithm is adopted with the mechanism of admission control for hard real-time tasks
- EDF with Admission Control with hard real-time prioritization, where the same algorithm in the preceding step is adopted with the difference that hard real-time tasks are executed only in physical CPUs

The simulator receives the following arguments:

- Number of tasks to be processed
- Percentage of hard real-time tasks from the total number of tasks
- Percentage of soft real-time tasks from the total number of tasks. From this value and the percentage of hard real-time tasks the percentage of regular tasks is deduced

- Percentage of mono-CPU tasks
- Percentage of dual-CPU tasks. From this value and the percentage of mono-CPU tasks the percentage of quad-CPU tasks is derived. Quad-CPU tasks might be broken to be executed in four different CPUs simultaneously
- Number of physical CPUs available
- CPU over-subscription factor. From this variable the number of virtual CPUs to be created will be derived. They will share the capacity of the physical CPUs
- Minimum physical CPU allocation. It guarantees that a percentage of physical CPUs will not be virtualized. This property is derived from the architecture proposed in this research and it will be used by some of the scheduling algorithms described below
- Capacity of the CPUs, which is homogeneous
- Virtualization overhead. This value quantifies the overhead added to the completion of a task being executed over a virtual CPU in comparison with a physical CPU
- Scheduling algorithm
- Statistical distribution for the inter-arrival of tasks. It can be a totally random distribution or a exponential distribution where the value for *theta* is entered

The performance variables which are measured are:

- Percentage of hard real-time tasks executed within their required deadline
- Percentage of soft real-time tasks executed within their required deadline
- Number of CPU cycles. This is a absolute number which measures how many CPU cycles were needed in order to process a workload. One CPU

cycle equals one cycle of processing of every CPU available in the cloud, being it physical or virtual. The lower this number, it means that the processing was more efficient

- **Profit.** For the scenario where the heuristic-based scheduler is applied, the profit is the sum of the value of each individual profit of each task processed within its deadline limits. Each task holds a profit value between 0 and 1. Our aim is to maximize this value

7.1.1 Data structure

In the simulation for the real-time capability, the data structures showed in figure 7.2 are used.

Figure 7.2: Data structure - real-time capability

All the vectors *arrival*, *deadline*, *WCET*, *Laxity*, *task_class* and *task_resource* holds the properties of the tasks defined in the workload. The CPU resources and their capacity are defined in the vector CPU. Also, there is a discrete time counter defined in *current_time* which is increased by the value defined in *UNITY_TIME* by each CPU cycle.

For each algorithm, the next task available in the vectors of tasks is selected

and the next CPU available in the vector CPU is selected in order to process that task. When the processing of the task is finished, the next task in the queue is picked and the next CPU available too. in this way, the parallelism of the execution of tasks at the same time in the cloud by all the CPUs is emulated.

For the scenarios which simulated the case study presented, 3 additional data structures are defined as shown in figure 7.3. They define the possible profit of each task to be executed, for which participant they belong and for which security value they belong to.

Figure 7.3: Data structure - additional structures

7.1.2 Random number generator

The simulator uses random numbers in several places in order to generate meaningful workloads for each scenario. The random number generator adopted is the function *rand()* available in the standard C language. Also, the function *srand(time(NULL));* is used in order to guarantee that the time of the machine where the simulator is executing will be taken as a seed to generate the random numbers, therefore spreading the range of values created as much and uniform as possible.

7.2 Real-time capability

In order to demonstrate the real-time capability of the architecture proposed, a series of experiments are conducted through the use of the simulator above described. One hundred distinct random workloads and one hundred distinct exponential-distributed workloads are generated and persisted beforehand. These same workloads are deliver to each of the experiments in order to compare the performance of each one. For each experiment, the six scheduling algorithms defined are executed. Then, the median and standard deviation values are calculated for: number of cpu cycles, percentage of completed hard real-time tasks, percentage of completed soft real-time tasks and profit.

The range for each of the values that the simulator accepts as arguments is defined in table 7.1:

Table 7.1: Simulator - range of arguments

Variable	Minimum value	Maximum value
Number of tasks	1	25000
Number of CPUs	1	1000
Capacity of the CPUs	0.1	0.9
CPU over-subscription factor	1	100
Virtualization overhead	0.1	0.9
Minimum physical allocation	0.1	0.9
Percentage of hard real-time tasks	0.1	0.9
Percentage of soft real-time tasks	0.1	0.9
Percentage of mono-cpu tasks	0.1	0.9
Percentage of dual-cpu tasks	0.1	0.9

7.2.1 Baseline

The baseline scenario will serve as a parameter to the other experiments. The input variables are defined in table 7.2:

Which yields the calculated variables presented in table 7.3:

The information presented in table 7.3 is omitted in the next experiments for the sake of organization.

Table 7.2: Experiment Baseline: input variables

Number of tasks	25000
Number of CPUs	2
Capacity of the CPUs	0.5
CPU oversubscription factor	5
Virtualization overhead	0.2
Minimum physical allocation	0.2
Percentage of hard real-time tasks	0.1
Percentage of soft real-time tasks	0.15
Percentage of mono-cpu tasks	0.8
Percentage of dual-cpu tasks	0.15
Inter-arrival distribution	RANDOM

Table 7.3: Experiment Baseline: calculated variables

Number of pCPUs	1
Number of vCPUs	5
Percentage of regular tasks	0.75
Percentage of quad-CPU tasks	0.05
Approximated number of hard real-time tasks	250
Approximated number of soft real-time tasks	375

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.4:

Figure 7.4: Results - Baseline - Number of CPU Cycles

With the mechanism of Admission Control (CAC) enabled, there is a 13% gain in the number of CPU cycles. That happens because unnecessary processing is not wasted if a deadline of a hard real-time task can not be completed and this is checked in advance. Depending on the workload presented to the cloud, the number of tasks discarded can have a large variation; that is why the standard deviation for the experiments with CAC are around 9%, while for the other experiments are around 1%. Whether the scheduling discipline is based on FIFO

or EDF, it does not affect the total time of computation, as EDF is a discipline which targets to maximize the tasks executed within its deadline and not the total number of CPU cycles. Another interesting point is that the discipline FIFO with pCPUs which allocates physical CPUs only to hard real-time tasks present an increase of 7% in the total number of CPU cycles as these physical CPUs remain idle for some cycles. There is the same effect with the EDF with CAC with hard RT prioritization, which might mean that Admission Control is more effective when more CPUs (physical or virtual) are available.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.5:

Figure 7.5: Results - Baseline - Percentage of completed hard real-time tasks

In these experiments, it can be noticed why the FIFO discipline is not suitable to real-time clouds as it only achieved 49% of completeness for hard real-time tasks. However, the allocation of hard real-time tasks only to physical CPUs while maintaining the FIFO discipline brings an improvement of 39% in this value. The Admission Control mechanism, solely, yields an improvement of 18%. Conversely, EDF with Admission Control reaches the best value for completed hard real-time tasks (72%), which is 24% better than the FIFO discipline with Admission Control and 47% better than the FIFO discipline. Also, the standard deviation for EDF with Admission Control is the best among all experiments, around 27%, which, however, can still be considered high. What is interesting too is that EDF with Admission Control mechanism provides a better performance than EDF with Admission Control with hard real-time tasks allocated only to physical CPUs, which might mean again that CAC is more effective with more

CPUs available.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.6:

Figure 7.6: Results - Baseline - Percentage of completed soft real-time tasks

The value of completed soft real-time tasks reaches 50% with the FIFO discipline. When applying Admission Control, there is an increase of 20% over this value. EDF with Admission Control reaches a very similar value, 57% of completed soft real-time tasks. It is not better than FIFO is that EDF was designed only to prioritize hard real-time tasks in the simulator. What is interesting is that the discipline which allocated physical CPUs only to hard real-time tasks and therefore execute soft real-time tasks with the FIFO discipline only on virtual CPUs presents a very low number of completed tasks - 22% - with a very high standard deviation - 47% -, meaning that clouds that present only virtual CPUs to user and do not accept other algorithms than FIFO are not suitable to real-time workloads.

7.2.2 Minimum virtualization overhead

In this scenario the impact of the virtualization overhead of the cloud is analyzed. The input variables are the same as defined in the baseline scenario but the virtualization overhead which is 0.05, four times lower than in the baseline.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.7:

Figure 7.7: Results - Minimum virtualization overhead - Number of CPU Cycles

The decrease in the virtualization overhead from 0.2 to 0.05 yields a 14% improvement in the total number of CPU cycles. This number is consistent along all disciplines. This means that this overhead is important to be decreased in order to reach a better throughput performance in the cloud. With the FIFO discipline with Admission Control the best number for CPU cycles is reached, around 19377.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.8:

Figure 7.8: Results - Minimum virtualization overhead - Percentage of completed hard real-time tasks

However, this same decrease in the virtualization overhead from 0.2 to 0.05 yields very small changes in the total number of completed hard real-time tasks. For the disciplines which process hard real-time tasks only in physical CPUs, it does not bring any improvement.

The results for the percentage of completed soft real-time tasks for one

hundred attempts for each scheduling algorithm is presented in figure 7.9:

Figure 7.9: Results - Minimum virtualization overhead - Percentage of completed soft real-time tasks

7.2.3 Maximum oversubscription factor

In this scenario the impact of the oversubscription factor applied to the CPUs of the cloud is analyzed. The input variables are the same as defined in the baseline scenario but the oversubscription factor which is 20. This oversubscription factor yields 20 virtual CPUs out of 1 physical CPU.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.10:

Figure 7.10: Results - Maximum oversubscription factor - Number of CPU Cycles

Having a higher number of virtual CPUs with less capacity each one than in the baseline scenario does not bring an improvement in the number of CPU cycles in the FIFO discipline, as the number of CPU cycles increased by 3%. An hypothesis can be made that, if more parallel tasks are allowed into the system,

this number would improve. However, the FIFO discipline with Admission Control presents an 6% improvement in the total number of CPU cycles. The disciplines that allocated physical CPUs only to hard real-time tasks present an improvement of 8% on the number of CPU cycles, as the effect of having idle CPUs waiting for hard real-time tasks is diminished.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.11:

Figure 7.11: Results - Maximum oversubscription factor - Percentage of completed hard real-time tasks

These number are interesting as demonstrate the damage that the oversubscription factor can do hard real-time tasks when the FIFO discipline is applied. The number reached in the experiments are 6%. When FIFO is applied with mechanisms as allocation of hard real-time tasks only to physical CPUs the number is very similar to the baseline experiments, as when Admission Control is enabled. EDF with Admission Control presents the best number so far for completeness of hard real-time tasks (83% of median and 15% of standard deviation), proving the hypothesis that CAC improves when more CPUs area available. This means an 69% improvement of completeness over the baseline scenario with FIFO.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.12:

With the exception where Admission Control is applied, the number of completed soft real-time tasks falls below 6% in all experiments. When is

Figure 7.12: Results - Maximum oversubscription factor - Percentage of completed soft real-time tasks

applied, it can a number close to the baseline in the FIFO discipline but 32% with EDF. As EDF only prioritize hard real-time tasks, it is harmful to soft real-time tasks; much more when a lot of virtual CPUs with less capacity are available and the algorithm does not make a distinction among regular and soft real-time tasks.

7.2.4 CPUs with more capacity

In this scenario CPUs with more capacity are available in the cloud. The input variables are the same as defined in the baseline scenario but the CPU capacity which is 0.8.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.13:

Figure 7.13: Results - CPUs with more capacity - Number of CPU Cycles

An increase of 60% in the CPU capacity yields 33% less number of CPU

cycles consistently across all scheduling disciplines.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.14:

Figure 7.14: Results - CPUs with more capacity - Percentage of completed hard real-time tasks

An increase of 60% in the CPU capacity yields an 18% higher percentage for completed hard real-time tasks in the FIFO discipline and an 14% increase in the FIFO discipline with Admission Control. For all the other disciplines, there are improvements close to 3%, which shows that the adding of more powerful CPUs will not solve the problem for real-time clouds.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.15:

Figure 7.15: Results - CPUs with more capacity - Percentage of completed soft real-time tasks

The increase in the percentage of soft real-time tasks is around 11% and 16% across all disciplines. It has to be investigated why the percentage of FIFO

with pCPUs to hard RT is very low, 4%.

7.2.5 More CPUs

In this scenario more CPUs are available in the cloud. The input variables are the same as defined in the baseline scenario except that there are 20 physical CPUs available.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.16:

Figure 7.16: Results - More CPUs - Number of CPU Cycles

With an tenfold increase in the number of CPUs available in the cloud, the number of CPU cycles is decrease by more than 10 times, around 93% and 94% across all disciplines. This shows that the throughput of the cloud is directly proportional to the number of CPUs available, regardless of the scheduling discipline adopted.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.17:

For the percentage of completed hard real-time tasks, a great improvement is obtained depending in the scheduling discipline and whether the Admission Control mechanism is enabled. For EDF with Admission Control, the best result is achieved for all the experiments, with the percentage of completed hard real-time tasks reaching 95% with a standard deviation of 4%. The number for the FIFO discipline is slight lower, reaching 92% of completeness with the same

Figure 7.17: Results - More CPUs - Percentage of completed hard real-time tasks

value for standard deviation. Interesting enough, the percentage of completeness for the FIFO discipline is lower than even in the baseline scenario. This happens because more virtual CPUs available and more hard real-time tasks were sent for processing in this virtual CPUs. The disciplines which send hard real-time tasks only to physical CPUs presents similar numbers to the baseline experiments. The hypothesis is that, for these disciplines, the number of physical CPUs in the baseline experiments seemed to be enough, while if a workload with a higher number of tasks was offered to the cloud, the beneficial effect of having more CPUs in the cloud could be noticed.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.18:

Figure 7.18: Results - More CPUs - Percentage of completed soft real-time tasks

Regarding soft real-time tasks, the best number in all the experiments is reached but now within the FIFO discipline with Admission Control, which reaches 93% with a standard deviation of 4%. As EDF prioritize only hard real-

time tasks, it reached 83% of completeness with a 9% in the standard deviation. For the other disciplines, there is no improvement over the baseline scenario.

7.2.6 More hard real-time tasks

In this scenario the cloud has the same configuration as defined in the baseline but the number of hard real-time tasks in the workload is increased to a value of 50 %. The percentage of soft real-time tasks is the same - 15% - and the percentage of regular tasks is 35%.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.19:

Figure 7.19: Results - More hard real-time tasks - Number of CPU Cycles

With a 33% higher number over the baseline for hard real-time tasks presented in the workload, the FIFO discipline decreases the number of CPU cycles by slightly more than 50%. For EDF, the number is decreased by 40%. However, the standard deviation number is much higher, around 50 and 60% of the median. The more hard real-time tasks presented to the cloud, the more difficult it is to predict its behaviour. For the disciplines which allocates hard real-time tasks to physical CPUs, the improvement is around 25%. It has to be investigated why the FIFO discipline is improved by 37% and it is not similar to the baseline scenario.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.20:

Figure 7.20: Results - More hard real-time tasks - Percentage of completed hard real-time tasks

For the EDF discipline with Admission Control, 80% of completeness for the hard real-time tasks are reached, against 72% in the baseline. The standard deviation is lower too, around 12%. There is an improvement in the FIFO discipline with Admission Control too. Interesting in the number for completed hard real-time tasks for all other disciplines, which are all equal reaching 46%. This suggest there is a upper limit for the performance of completed hard real-time tasks, around 46%.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.21:

Figure 7.21: Results - More hard real-time tasks - Percentage of completed soft real-time tasks

Presenting more hard real-time tasks to the cloud can decrease very much the number of completed soft real-time tasks. The exception is for the disciplines which apply Admission Control that do not waste resources for tasks that can not be completed under their deadlines. For the disciplines which allocated hard real-

time tasks only to physical CPUs, the results falls under 25% of completeness.

7.2.7 More parallel tasks

In this scenario the cloud has the same configuration as defined in the baseline but the number of parallel tasks in the workload is increased. The percentage of mono-CPU tasks is defined as 10%, dual-CPU tasks as 50% and quad-CPU tasks as 40%.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.22:

Figure 7.22: Results - More parallel tasks - Number of CPU Cycles

This new distribution of the workload with much more parallel tasks brings the number of CPU cycles 50% than in the baseline. This shows that the more parallel tasks presented to the cloud, the best the throughput of it.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.23:

The number of completed hard real-time tasks is not improved when more parallel tasks are deliver to the cloud. That happens because is very hard to synchronize the processing of all the parallel tasks in all the CPUs. A better algorithm could be developed for this purpose though. Even worse, in the case of disciplines that allocate hard real-time tasks only to physical CPUs, there is not enough processing capacity to distribute the workload across all these physical CPUs.

Figure 7.23: Results - More parallel tasks - Percentage of completed hard real-time tasks

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.24:

Figure 7.24: Results - More parallel tasks - Percentage of completed soft real-time tasks

As in the case with hard real-time tasks, a similar behaviour of a slight decrease in the number of completed soft real-time tasks is noticed.

7.2.8 Exponential distribution

In this scenario the cloud has the same configuration as defined in the baseline but the interarrival distribution for the tasks is an exponential distribution with the value of θ defined as 0.5.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.25:

With this new interarrival distribution for the tasks in the workload, the EDF

Figure 7.25: Results -Exponential distribution - Number of CPU Cycles

and FIFO disciplines with Admission Control shows an improvement of almost 20% in its total throughput. It is a higher number than with the totally random distribution within the baseline scenario which reached 13%. However, for all the other disciplines the numbers are similar to the baseline scenario. It has to be investigated why the EDF with CAC with hard RT prioritization presented the same number as in the baseline.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.26:

Figure 7.26: Results - Exponential distribution - Percentage of completed hard real-time tasks

In comparison with the baseline scenario, the percentage of completed hard real-time tasks is lower in all the disciplines, showing that this exponential distribution presented to the cloud is more aggressive than the random distribution presented. However, the improvement of the EDF with Admission control discipline over the baseline scenario is improved, around 88% better.

The results for the percentage of completed soft real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.27:

Figure 7.27: Results - Exponential distribution - Percentage of completed soft real-time tasks

Similar to hard real-time tasks, the number of completed soft real-time tasks is lower too when an exponential distribution with θ equals to 3.5 is presented to the cloud in comparison with a total random distribution. However, the improvement of FIFO with Admission Control is 60% over the standard FIFO discipline, better than in the baseline scenario. In this experiments, it is interesting that the percentage for the FIFO with pCPUs to hard RT reached almost 0, around 0,06%.

7.3 Heuristic-based scheduler for a real-time trading application

In this section, it will be simulated a scenario which resembles the case study described with the utilization of the heuristic-based scheduler presented in the previous chapter. The resources available in the cloud are the same as in the baseline scenario presented. There are 2 participants in the cloud who will share its resources and each one possess a workload of 7500 tasks, where each tasks holds a value called profit which is a value that must be maximized. The value of profit is valid only if the task is executed within its deadline.

The results for the number of CPU cycles for one hundred attempts for each scheduling algorithm is presented in figure 7.28:

Figure 7.28: Results - Heuristic-based scheduler - Number of CPU Cycles

The number of CPU cycles is very similar with the 3 disciplines presented., being 6% better in the best case of EDF with CAC. The standard deviation is a little bit higher when the heuristic-based scheduler proposed is applied, but is still under 2% of the median.

The results for the percentage of completed hard real-time tasks for one hundred attempts for each scheduling algorithm is presented in figure 7.29:

Figure 7.29: Results - Heuristic-based scheduler - Percentage of completed hard real-time tasks

The best number is reached for the EDF with Admission control discipline, 59%, with the heuristic-based scheduler reaching 48%, which is still 41% over the number of FIFO with Admission Control. The prime objective of EDF is to maximize the number of completed hard real-time tasks. Conversely, the heuristic-based scheduler tries to find the best value of profit while still maintaining a high number of completed hard real-time tasks.

The results for the percentage of completed soft real-time tasks for one

hundred attempts for each scheduling algorithm is presented in figure 7.30:

Figure 7.30: Results - Heuristic-based scheduler - Percentage of completed soft real-time tasks

As EDF and the proposed scheduler treats soft real-time tasks as regular tasks with no prioritization, the best number reached in these experiments are within FIFO with Admission Control discipline, around 33%.

Finally, the results for the profit calculated for one hundred attempts for each scheduling algorithm is presented in figure 7.31:

Figure 7.31: Results - Heuristic-based scheduler - Profit

For the profit value, EDF reaches an 14% better performance over FIFO. However, the heuristic-based scheduler an 83% improvement over EDF and more than double the performance in comparison with the FIFO discipline. As showed, while maintaining the number of completed hard real-time tasks within a high range, it reached, by far, the best numbers for the profit value.

Chapter 8

CONCLUSIONS AND FUTURE WORKS

8.1 Conclusions

The main motivation of this research is to demonstrate that cloud systems are suitable for real-time applications. The architecture proposed presented properties which are different from ordinary cloud environments and, *per se*, it is an important contribution to the theme. As developed in the research, this architecture is based on the research of several real-time distributed systems. As it is modular and flexible, this architecture accepts the coupling of any kind of scheduling algorithm and new extensions could be developed over it.

The simulator written for this research in the C language provided the results which corroborate and tune the hypotheses proposed. It is an important tool in the sense that it provided a lot of conclusions about the real-time cloud environment over a short range or time. The idea to have developed my own simulator revealed to be a good one, as other alternative simulators can not give the same flexibility. It is envisioned that the code of this simulator could be used and extended by other researchers as well.

The case study presented, a real-time community cloud suitable to be adopted in collocation spaces at stock exchanges is a model that could be extended to other infrastructures which require real-time services. Also, it is envisioned that this environment possess many similarities with other ones,

such as flight control systems, military systems, robotics, industrial automation, among others. For some systems, very strict safety requirements should be added. The architecture proposed can support this safety requirements as well, being the role of the scheduling algorithm to allocate the resources in order to achieve this.

8.2 Future Works

There are some interesting opportunities to further develop this research which are listed here.

8.2.1 Architecture

The architecture developed in this research could be extended in order to provide some other interesting properties to the participants of a community cloud. For instance, it should be researched whether it is possible and the effects of an eventual allocation of dedicated hardware to some participants during a particular range of time in order to improve its performance.

Another interesting study is to consider the coupling of this real-time cloud with ordinary private cloud and even public clouds in order to verify compatibility issues and performance penalties which this strategy could eventually bring.

8.2.2 Scheduling

Besides the scheduling algorithm proposed in this research, a number of other advances regarding scheduling which are of relevance on this environment might be developed such as to consider the computation of tasks with precedence constraints. Solving this particular restriction and allowing the real-time cloud to take this into consideration would widen the scope of this research.

Another future work is to consider in the scheduling algorithm the arrival of not only aperiodic tasks, but also periodic tasks. This could bring more performance to the cloud system in the sense that the treatment of periodic tasks are easier than aperiodic ones.

As already stated, both the architecture and the simulator proposed bring the possibility to extend the concepts described to other environments. For these environments, particular heuristic functions could be inserted into scheduling algorithms, in order to take advantage of the behaviour of the workload that the system is subjected to it.

Machine learning algorithms could be employed in order to better understand the behaviour of a workload in a particular environment in order to adjust the parameters of the scheduling algorithm, improving the overall system performance.

8.2.3 QoS

Another advance is the possibility to better elaborate over Quality of Services parameters. The admission control mechanism, which proved to be very efficient, could adopt different policies depending on parameters such as hour of the day or occupation of resources. Also, it should be compared the effects of having a uniform QoS policy for all participants against a model where each participant adopts its own unique policy.

8.2.4 Simulations

The simulator developed for this research could be extended to bring:

- **Workload characterization:** new mathematical distributions could be added to the library of the simulator, in order to demonstrate the impact of each of them in the overall performance of the cloud

- Scheduling algorithm: new scheduling strategies could be included in the simulator in order to verify its efficiency against the workload the cloud is subjected to it
- Heterogeneous hardware: in the simulator developed, all the CPUs need to have the same capacity. An interesting feature would be to check the effects of the simulation with CPUs with different capacities
- Real-time communication: the delay involved in the communication between different virtual machines instantiated in the cloud could be emulated. That would bring the interesting possibility of the researching of novel real-time communication methods for cloud environments
- Sets of community clouds: another evolution of the simulator would be to consider the instantiation of multiple sets of community clouds, whereby it could be tested how they would interact with one another and how one could send tasks to another and vice-versa

8.2.5 Real-time communication

As explained in the bibliographical review, real-time communications for real-time clouds is based on ordinary Ethernet and IP platforms. In the history of real-time distributed systems, a whole area of research was fully dedicated to this subject. Regarding cloud-computing environments, one first proposal was introduced in [XI and OTHERS 2013]. So this is an area which should be developed given that networks, both physical (underlay) and virtual (overlay), are a crucial part of a real-time cloud system and have direct impact over its performance. In addition to it, an overlay architecture which could self-adapt to the demands of workload and quality of communications would be highly desirable.

8.2.6 Dynamics of the behaviour of participants

An interesting study would be to simulate the behaviour of participants over the private community real-time cloud proposed in the scenario described in the case study. Strategies for competition and cooperation could be applied, depending on the market conditions and the strategy of each participant. These strategies should be analysed in the light of how they would influence the whole performance of the cloud and of the other participants.

8.2.7 Prototype

Finally, an interesting challenge would be to build a prototype for the system developed, whereby the hypotheses demonstrated in this research could be proved in a real scenario. This prototype would help to fine tune the aspects related to the building of a real-time community cloud and it would advance this research in the sense that new aspects could be improved. Also, more precise quantitative measures of performance would be taken with this prototype.

8.2.8 Benchmarks for real-time clouds

Given all the parameters and metrics of efficiency presented in this research, a benchmarking for real-time clouds could be developed and it would be very useful in order to compare the real-time capabilities of public and ordinary private clouds. Today there are some benchmarks that measure the performance of clouds, but no one is related to real-time clouds.

Bibliography

- [ARMBRUST et al. 2009]ARMBRUST, M. et al. Above the clouds: a berkeley view of cloud computing. In: DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, UNIVERSITY OF CALIFORNIA, BERKELEY. *Technical Report No. UCB/EECS-2009-28*. [S.l.], 2009. 1, 7, 11, 13
- [AVERSA 2011]AVERSA, R. Performance prediction for hpc on clouds. In: *Cloud Computing, Principles and Paradigms*. [S.l.: s.n.], 2011. 13, 82
- [BALAKRISHNAN et al. 2011]BALAKRISHNAN, H. et al. Openflow: Enabling innovation in campus networks. *IEEE Network*, p. 6–10, July-August 2011. 72, 83
- [BARHAM et al. 2003]BARHAM, P. et al. Xen and the art of virtualization. *University of Cambridge Computer Laboratory*, 2003. 31
- [BARHAM et al. 2003]BARHAM, P. et al. Xen and the art of virtualization. *University of Cambridge Computer Laboratory*, 2003. 82
- [BATS Global Markets, Inc 2012]BATS GLOBAL MARKETS, INC. *BATS - US Equity Options Connectivity Manual*. 6.0.2. ed. [S.l.], 2012. 54
- [BIRMAN 2005]BIRMAN, K. P. *Reliable Distributed Systems - Technologies, Web Services and Applications*. First. [S.l.]: Springer, 2005. 25
- [BIRMAN and JOSEPH 1987]BIRMAN, K. P.; JOSEPH, T. A. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, v. 5, n. 1, p. 47–76, February 1987. 25

- [BITTENCOURT, MADEIRA and FONSECA 2012]BITTENCOURT, L.; MADEIRA, E.; FONSECA, N. Scheduling in hybrid clouds. *IEEE Communications Magazine*, p. 42–47, September 2012. 2
- [BITTENCOURT, MADEIRA and FONSECA 2012]BITTENCOURT, L.; MADEIRA, E.; FONSECA, N. Scheduling in hybrid clouds. *IEEE Communications Magazine*, p. 42–47, September 2012. 78
- [BRISCOE and MARINO 2009]BRISCOE, G.; MARINO, A. Digital ecosystems in the clouds: Towards community cloud computing. In: INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). *IEEE, (corp. ed.) 2009 3rd IEEE International Conference on Digital Ecosystems and Technologies (Dest 2009)*. [S.l.], 2009. p. 103–108. 11, 12
- [BRISCOE and WILDE 2006]BRISCOE, G.; WILDE, P. D. Digital ecosystems: Evolving service-oriented architectures. In: PRESS, I. (Ed.). *Conference on Bio Inspired Models of Network, Information and Computing Systems*. [s.n.], 2006. Disponível em: <<http://arxiv.org/abs/0712.4102>>. 11, 64
- [BRYANT 2011]BRYANT, R. Data-intensive scalable computing for scientific applications. *IEEE Computing in Science and Engineering*, p. 25–33, November-December 2011. 1
- [BUTTAZZO 2005]BUTTAZZO, G. C. *Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications*. Second. [S.l.]: Springer, 2005. 12, 13, 78
- [CHANDRAMOULI et al. 2012]CHANDRAMOULI, B. et al. Race: real-time applications over cloud-edge. In: ACM. *SIGMOD Conference*. [S.l.], 2012. p. 625–628. 35, 37
- [CHENG 2002]CHENG, A. M. K. *Real-Time Systems - Scheduling, Analysis and Verification*. First. [S.l.]: Wiley-Interscience, 2002. 12, 13
- [CME Group, Inc 2011]CME GROUP, INC. *CME Group Client-Managed Router Guidance*. [S.l.], 2011. 54

- [COULAS, MACEWEN and MARQUIS 1987]COULAS, M. F.; MACEWEN, G. H.; MARQUIS, G. Rnet: A hard real-time distributed programming system. *IEEE Transactions on Computers*, C-36, n. 8, p. 917–932, August 1987. 17, 18
- [DEVINE, BUGNION and ROSENBLUM 2002]DEVINE, S.; BUGNION, E.; ROSENBLUM, M. *Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture*. [S.l.], 2002. 82
- [ENDO et al. 2011]ENDO, P. et al. Resource allocation for distributed cloud: Concepts and research challenges. *IEEE Network*, p. 42–46, July-August 2011. 2
- [ENDO et al. 2011]ENDO, P. et al. Resource allocation for distributed cloud: Concepts and research challenges. *IEEE Network*, p. 42–46, July-August 2011. 78
- [FOSTER et al. 2008]FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: GCE08. *Proc. Grid Computing Environments*. [S.l.], 2008. p. 1–10. 1, 7
- [FOSTER et al. 2008]FOSTER, I. et al. Cloud computing and grid computing 360-degree compared. In: GCE08. *Proc. Grid Computing Environments*. [S.l.], 2008. p. 1–10. 66
- [GORLATCH et al. 2012]GORLATCH, S. et al. Towards bringing real-time on-line applications on clouds. In: IEEE. *International Conference on Computing, Networking and Communications, Cloud Computing and Networking Symposium*. [S.l.], 2012. p. 57–61. 2, 13, 78, 83
- [GORLATCH et al. 2012]GORLATCH, S. et al. Towards bringing real-time on-line applications on clouds. In: IEEE. *International Conference on Computing, Networking and Communications, Cloud Computing and Networking Symposium*. [S.l.], 2012. p. 57–61. 34
- [GRANCE and MELL 2011]GRANCE, T.; MELL, P. The nist definition of cloud computing. In: NATIONAL INSTITUTE OF STANDARDS AND

- TECHNOLOGY - U.S. DEPARTMENT OF COMMERCE. *Special Publication 800-145 (Draft)*. [S.l.], 2011. 7, 8, 10
- [HORN 1974]HORN, W. A. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, v. 21, n. 1, p. 177–185, March 1974. 12
- [JENSEN, LOCKE and TOKUDA 1985]JENSEN, E. D.; LOCKE, C. D.; TOKUDA, H. A time-driven scheduling model for real-time systems. In: *IEEE Real-Time Systems Symposium*. [S.l.: s.n.], 1985. 36
- [JENSEN and NORTHCUTT 1990]JENSEN, E. D.; NORTHCUTT, J. D. Alpha: a non-proprietary os for large, complex, distributed real-time systems. In: *Procs. of the IEEE Workshop on Experimental Distributed Systems*. [S.l.: s.n.], 1990. p. 35–41. 24
- [Johnson 2010]JOHNSON, B. *Algorithmic trading DMA : an introduction to direct access trading strategies*. First. [S.l.]: 4Myeloma Press, 2010. 55
- [KANDLUR, KISKIS and SHIN 1989]KANDLUR, D. D.; KISKIS, D. L.; SHIN, K. G. *HARTOS: A Distributed Real-Time Operating Sytem*. Ann Arbor, MI, 48109-2122, 1989. 22
- [KIECKHAFFER et al. 1988]KIECKHAFFER, R. M. et al. The maft architecture for distributed fault tolerance. *IEEE Transactions on Computers*, v. 37, n. 4, p. 4, April 1988. 20
- [KIM and PARASHAR 2011]KIM, H.; PARASHAR, M. Cloud computing: Principles and paradigms. In: _____. [S.l.]: John Wil, 2011. cap. 10 - Comet Cloud: an Autonomic Cloud Engine, p. 275–297. 16
- [KIM, BELOGLAZOV and BUYYA 2011]KIM, K. H.; BELOGLAZOV, A.; BUYYA, R. Power-aware provisioning of virtual machines for real-time cloud services. *Concurrency and Computation: Practice and Experience*, v. 23, p. 1491–1505, March 2011. 33, 37
- [KIM, BELOGLAZOV and BUYYA 2011]KIM, K. H.; BELOGLAZOV, A.; BUYYA, R. Power-aware provisioning of virtual machines for real-time cloud

- services. *Concurrency and Computation: Practice and Experience*, v. 23, p. 1491–1505, March 2011. 78
- [KOPETZ et al. 1989]KOPETZ, H. et al. Distributed fault-tolerant real-time systems: The mars approach. *IEEE Micro*, v. 9, n. 1, p. 25–40, February 1989. 23
- [LEE and KING 1988]LEE, I.; KING, R. *Timix: a Distributed Real-Time Kernel for Multi-Sensor Robots*. Philadelphia, PA, 19104-6389, 1988. 19
- [LEE, PAUL and KING 1988]LEE, I.; PAUL, R.; KING, R. *RK: A Real-Time Kernel for a Distributed System with Predictable Response*. Philadelphia, PA 19104-6389, 1988. 18, 19
- [LEE et al. 2010]LEE, M. et al. Supporting soft real-time tasks in the xen hypervisor. In: ACM. *VEE '10 Proceedings of the 6th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. New York, NY, USA, 2010. p. 97–108. 31
- [LENK et al. 2009]LENK, A. et al. What's inside the cloud? an architectural map of the cloud landscape. In: IEEE. *ICSE'09 Workshop*. [S.l.], 2009. p. 23–31. 8
- [LI 2004]LI, P. *Utility accrual real-time scheduling: Models and algorithms*. Tese (Doutorado) — Virginia Polytechnic Institute and State University, 2004. 36
- [LIU 2000]LIU, J. W. S. *Real-Time Systems*. First. [S.l.]: Prentice Hall, 2000. 610 p. 13
- [LIU, QUAN and REN 2010]LIU, S.; QUAN, G.; REN, S. On-line scheduling of real-time services for cloud computing. In: SOCIETY, I. C. (Ed.). *2010 IEEE 6th World Congress on Services*. [S.l.], 2010. 2
- [LIU, QUAN and REN 2010]LIU, S.; QUAN, G.; REN, S. On-line scheduling of real-time services for cloud computing. In: SOCIETY, I. C. (Ed.). *2010 IEEE 6th World Congress on Services*. [S.l.], 2010. p. 459–464. 13, 36, 78

- [LLORENT 2011]LLORENT, I. Cloud computing: Principles and paradigms. In: _____. [S.l.]: John Wile, 2011. cap. 6 - On the Management of Virtual Machines for Cloud Infrastructures, p. 157–191. 15
- [MOK 1983]MOK, A. K.-L. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. Tese (Ph.D.) — Massachusetts Institute of Technology, May 1983. 14, 16, 47
- [NORTHCUTT 1987]NORTHCUTT, J. D. *Mechanisms for Reliable Distributed Real-Time Operating Systems - The Alpha Kernel*. [S.l.]: Academic Press, Inc., 1987. 14, 16
- [NYSE Euronext 2011]NYSE EURONEXT. *US Liquidity Center Products and Services Guide - Colocation and Networks*. 4. ed. [S.l.], 2011. 55
- [PERROS and XIONG 2009]PERROS, H.; XIONG, K. Service performance and analysis in cloud computing. In: IEEE. *2009 Congress on Services - I*. [S.l.]: IEEE Computer Society, 2009. p. 693–700. 13
- [PHAN et al. 2011]PHAN, L. et al. An empirical analysis of scheduling techniques for real-time cloud-based data processing. In: SOCIETY, I. C. (Ed.). *SOCA'11 Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*. [S.l.], 2011. p. 1–8. 2, 15, 32
- [PHAN et al. 2011]PHAN, L. et al. An empirical analysis of scheduling techniques for real-time cloud-based data processing. In: SOCIETY, I. C. (Ed.). *SOCA'11 Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications*. [S.l.], 2011. p. 1–8. 13, 33, 78
- [RIMAL and CHOI 2012]RIMAL, B.; CHOI, E. A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing. *International Journal of Communication Systems*, v. 25, p. 796–819, 2012. 8, 11, 64
- [ROZIER and OTHERS 1988]ROZIER; OTHERS. Chorus distributed operating systems. *Computing Systems*, v. 1, p. 305–379, October 1988. 19

- [SHA, RAJKUMAR and LEHOCZKY 1987]SHA, L.; RAJKUMAR, R.; LEHOCZKY, J. P. *Priority Inheritance Protocol: an Approach to Real-Time Synchronization*. [S.l.], 1987. 21
- [SHNEIDERMAN 1979]SHNEIDERMAN, B. Human factors experiments in designing interactive systems. *IEEE Computer*, v. 12, n. 12, p. 9–19, December 1979. 14
- [STANKOVIC 1988]STANKOVIC, J. A. Misconceptions about real-time computing - a serious problem for next-generation systems. *IEEE Computer*, v. 21, n. 10, p. 10–19, October 1988. 15, 16
- [STANKOVIC and RAMAMRITHAM 1987]STANKOVIC, J. A.; RAMAMRITHAM, K. The design of the spring kernel. In: *Proceedings of the IEEE Real-Time Systems Symposium*. [S.l.: s.n.], 1987. 78
- [STANKOVIC and RAMAMRITHAM 1991]STANKOVIC, J. A.; RAMAMRITHAM, K. The spring kernel: a new paradigm for real-time systems. *IEEE Software*, May 1991. 78, 79
- [TANENBAUM and STEEN 2006]TANENBAUM, A.; STEEN, M. V. *Distributed Systems: Principles and Paradigms (Second Edition)*. [S.l.]: Prentice Hall, 2006. 7
- [TOKUDA and MERCER 1989]TOKUDA, H.; MERCER, C. W. Arts: A distributed real-time kernel. *Operating System Review*, v. 23, n. 3, p. 29–53, July 1989. 21, 22
- [VARIA 2008]VARIA, J. *Cloud Architectures*. 2008. [Http://aws.amazon.com](http://aws.amazon.com). 7
- [VERISSIMO L. RODRIGUES and SPEIRS 1990]VERISSIMO L. RODRIGUES, P. B. A. H. P. B. D. S. P.; SPEIRS, N. The delta-4 extra performance architecture (xpa). In: IEEE. *In Proceedings of the 20th Int. Symp. on Fault-Tolerant Computing Systems (FTCS-20)*. [S.l.], 1990. p. 481–488. 26
- [WOLF et al. 2010]WOLF, J. et al. Flex: A slot allocation scheduling optimizer for mapreduce workloads. *Middleware*, n. LNCS 6452, p. 1–20, 2010. 78

- [WU et al. 2012]WU, X. et al. Jump-start cloud: Efficient deployment framework for large-scale cloud applications. *Concurrency and Computation: Practice and Experience*, v. 24, p. 2120–2137, 2012. 13
- [XI et al. 2013]XI, S. et al. *Global Real-Time Multi-Core Virtual Machine Scheduling in Xen*. [S.l.], 2013. 35
- [XI and OTHERS 2013]XI, S.; OTHERS. Prioritizing local inter-domain communication in xen. In: *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*. [S.l.]: IEEE, 2013. p. 1–10. 35, 113
- [Xi et al. 2011]XI, S. et al. Rt-xen: Towards real-time hypervisor scheduling in xen. In: *Proc. of ACM International Conference on Embedded Software*. [S.l.: s.n.], 2011. 31
- [YU 2010]YU, Y. Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time. In: *RTS'2010. SAC2010 - Track on Real-Time System*. [S.l.], 2010. 36
- [ZAHARIA et al. 2010]ZAHARIA, M. et al. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In: *ACM (Ed.). EuroSys' 2010*. Paris, France, 2010. p. 265–278. 78