UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

IVÁN MARCELO CARRERA IZURIETA

Performance Modeling of MapReduce Applications for the Cloud

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Prof. Dr. Cláudio Fernando Resin Geyer
Advisor

Porto Alegre, 2014

CIP – CATALOGING-IN-PUBLICATION

Carrera Izurieta, Iván Marcelo

Performance Modeling of MapReduce Applications for the Cloud / Iván Marcelo Carrera Izurieta. – Porto Alegre: PPGC da UFRGS, 2014.

64 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Mestrado em Ciência da Computação, Porto Alegre, BR–

, 2014. Advisor: Cláudio Fernando Resin Geyer.

1. Performance Evaluation. 2. Cloud Computing. 3. MapReduce. 4. Capacity Planning. I. Resin Geyer, Cláudio Fernando. II. Título.

Dedicado a todos quienes acompañaron, aún a la distancia, mi vida en Brasil:

A mis padres, Manuel y Fabiola, por siempre haber incentivado la creatividad, la crítica y la curiosidad en mí, y por el apoyo constante e incondicional y porque me enseñaron a perseguir mis sueños hasta conseguirlos.

A Miguel Ángel, mi hermano mayor, por ser un ejemplo de amor al estudio, a la verdad, a la ciencia y a la familia.

A Paolita, mi hermanita menor, por sus constantes palabras de apoyo y por siempre estar pendiente de mi vida lejos de casa.

A Gloria, mi compañera de vida, por haberme llenado de amor y largas conversaciones académicas, por ser la mejor parte de mi vida en Brasil.

A mis amigos, por haber sido mi familia lejos de casa.

Este trabajo es la materialización de un sueño que empezó en 2009.
No dejaré de soñar, ahora lo haré más alto.

ACKNOWLEDGEMENTS

I want to thank God in first place, for He has given me the opportunities to be, do and live as I had to, in order to accomplish my path in this life.

I want to thank also my family and friends, for they have accompanied my journey in Brazil. And to the lady that inspired the idea of taking the concept of Impressionism to the Cloud. I could never have done none of this without their support.

I thank professor Cláudio Geyer for have given me the opportunity to work with him and thus being able to fulfil my dream of study grad school in Brazil.

Special thanks to professor Arnaud Legrand from *Centre national de la recherche scientifique* CNRS in France for his help explaining me the research problems in this work, professor Luciana Arantes from the University of Pierre et Marie Currie (Paris 6), professors Luciano Gaspary and Lucas Schnorr from *Universidade Federal do Rio Grande do Sul* UFRGS in Brazil, and to Ph.D. Pedro Velho for their help and reviews in several stages of my research.

Also special thanks to my colleague Julio Anjos for all his help during my time in UFRGS and for he has been a true friend. To my labmates Flavio Alles, Pierre Turin and Fabricio Scariot for their help running some of the experiments presented in this work, a true feeling of gratefulness.

The present work was made with the support of *Programa Estudantes-Convênio de Pós-Graduação – PEC-PG, of CAPES/CNPq - Brazil.*

Some of the experiments presented in this work were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (https://`www.grid5000.fr`).

"Divide each difficulty into as many parts as is feasible and necessary to resolve it."
RENÉ DESCARTES

ABSTRACT

In the last years, Cloud Computing has become a key technology that made possible running applications without needing to deploy a physical infrastructure with the advantage of lowering costs to the user by charging only for the computational resources used by the application. The challenge with deploying distributed applications in Cloud Computing environments is that the virtual machine infrastructure should be planned in a way that is time and cost-effective.

Also, in the last years we have seen how the amount of data produced by applications has grown bigger than ever. This data contains valuable information that has to be extracted using tools like MapReduce. MapReduce is an important framework to analyze large amounts of data since it was proposed by Google, and made open source by Apache with its Hadoop implementation.

The goal of this work is to show that the execution time of a distributed application, namely, a MapReduce application, in a Cloud computing environment, can be predicted using a mathematical model based on theoretical specifications. This prediction is made to help the users of the Cloud Computing environment to plan their deployments, i.e., quantify the number of virtual machines and its characteristics in order to have a lesser cost and/or time.

After measuring the application execution time and varying parameters stated in the mathematical model, and after that, using a linear regression technique, the goal is achieved finding a model of the execution time which was then applied to predict the execution time of MapReduce applications with satisfying results.

The experiments were conducted in several configurations: namely, private and public clusters, as well as commercial cloud infrastructures, running different MapReduce applications, and varying the number of nodes composing the cluster, as well as the amount of workload given to the application. Experiments showed a clear relation with the theoretical model, revealing that the model is in fact able to predict the execution time of MapReduce applications. The developed model is generic, meaning that it uses theoretical abstractions for the computing capacity of the environment and the computing cost of the MapReduce application.

Further work in extending this approach to fit other types of distributed applications is encouraged, as well as including this mathematical model into Cloud services offering MapReduce platforms, in order to aid users plan their deployments.

Keywords: Performance Evaluation, Cloud Computing, MapReduce, Capacity Planning.

Modelagem de Desempenho de Aplicações MapReduce para a Núvem

RESUMO

Nos últimos anos, Cloud Computing tem se tornado uma tecnologia importante que possibilitou executar aplicações sem a necessidade de implementar uma infraestrutura física com a vantagem de reduzir os custos ao usuário cobrando somente pelos recursos computacionais utilizados pela aplicação. O desafio com a implementação de aplicações distribuídas em ambientes de Cloud Computing é o planejamento da infraestrutura de máquinas virtuais visando otimizar o tempo de execução e o custo da implementação.

Assim mesmo, nos últimos anos temos visto como a quantidade de dados produzida pelas aplicações cresceu mais que nunca. Estes dados contêm informação valiosa que deve ser obtida utilizando ferramentas como MapReduce. MapReduce é um importante framework para análise de grandes quantidades de dados desde que foi proposto pela Google, e disponibilizado Open Source pela Apache com a sua implementação Hadoop.

O objetivo deste trabalho é apresentar que é possível predizer o tempo de execução de uma aplicação distribuída, a saber, uma aplicação MapReduce, na infraestrutura de Cloud Computing, utilizando um modelo matemático baseado em especificações teóricas.

Após medir o tempo levado na execução da aplicação e variando os parámetros indicados no modelo matemático, e, após utilizar uma técnica de regressão linear, o objetivo é atingido encontrando um modelo do tempo de execução que foi posteriormente aplicado para predizer o tempo de execução de aplicações MapReduce com resultados satisfatórios.

Os experimentos foram realizados em diferentes configurações: a saber, executando diferentes aplicações MapReduce em clusters privados e públicos, bem como em infraestruturas de Cloud comercial, e variando o número de nós que compõem o cluster, e o tamanho do workload dado à aplicação. Os experimentos mostraram uma clara relação com o modelo teórico, indicando que o modelo é, de fato, capaz de predizer o tempo de execução de aplicações MapReduce. O modelo desenvolvido é genérico, o que quer dizer que utiliza abstrações teóricas para a capacidade computacional do ambiente e o custo computacional da aplicação MapReduce.

Motiva-se a desenvolver trabalhos futuros para estender esta abordagem para atingir outro tipo de aplicações distribuídas, e também incluir o modelo matemático deste trabalho dentro de serviços na núvem que ofereçam plataformas MapReduce, a fim de ajudar os usuários a planejar suas implementações.

Palavras-chave: Avaliação de Desempenho, Computação em Núvem, MapReduce, Dimensionamento de Servidores.

LIST OF FIGURES

Figure 3.1: MapReduce execution flow 32

Figure 5.1: Execution time vs. Workload for the `gradep` cluster 49
Figure 5.2: Execution time vs. Workload for the Griffon cluster running the Sorter application 51
Figure 5.3: Execution time vs. Workload for the Griffon cluster running the Word-count application 52
Figure 5.4: Execution time vs. Workload for the Amazon EMR Environment . . 54
Figure 5.5: Execution time vs. Workload for the Azure HDInsight Environment . 56

LIST OF TABLES

Table 3.1: Related work in Performance Evaluation of MapReduce applications 36
Table 3.2: Related work in Benchmarking for MapReduce applications 38

Table 4.1: Parameter Values 45

Table 5.1: Parameter values for `gradep` cluster 47
Table 5.2: Results for model assessing on the `gradep` cluster 49
Table 5.3: Parameter values for Grid'5000 50
Table 5.4: Parameter values for Amazon Elastic MapReduce environment 53
Table 5.5: Results for model assessing of the Amazon measures 54
Table 5.6: Parameter values for Windows Azure HDInsight 55
Table 5.7: Results for model assessing the measures on the Azure HDInsight Environment 56

LIST OF ABBREVIATIONS AND ACRONYMS

AWS	Amazon Web Services
CDN	Content Delivery Network
CUS	Component Under Study
DFS	Distributed File System
EC2	Elastic Compute Cloud
ECU	EC2 Compute Unit
EMR	Elastic MapReduce
HDFS	Hadoop Distributed FileSystem
IaaS	Infrastructure as a Service
MTC	Many Task Computing
MR	MapReduce
NFS	Network File System
SaaS	Software as a Service
S3	Simple Storage Service
SLA	Service Level Agreement
SPoF	Single Point of Failure
SUT	System Under Test
VM	Virtual Machine

CONTENTS

1 INTRODUCTION 19
1.1 Motivation 19
1.2 Objectives and Contributions 20
1.3 Relation with GPPD's research 21
1.4 Organization 21

2 PERFORMANCE EVALUATION 23
2.1 Introduction 23
2.2 Performance Modeling 23
2.3 Capacity Planning 25
2.4 Performance Evaluation in Cloud Environments 26
2.5 Concluding Remarks 29

3 RELATED WORK 31
3.1 MapReduce 31
3.2 Hadoop Overview 32
3.3 Performance Evaluation and Modeling over MapReduce 33
3.4 Benchmarking in MapReduce 37
3.5 Concluding Remarks 38

4 PROPOSED MODEL AND EXPERIMENTAL DESIGN 39
4.1 Proposed Hadoop Performance Model 39
4.2 Scenarios 42
4.2.1 Hardware description 42
4.2.2 Software description 43
4.3 Specification of Experiments 44
4.3.1 Evaluation Technique 44
4.3.2 Performance Metrics 44
4.3.3 Workload 44
4.3.4 Parameter Values 44
4.4 Concluding Remarks 45

5 RESULTS 47
5.1 Experiments on Physical Clusters 47
5.1.1 Private Cluster `gradep` 47
5.1.2 Grid'5000 50
5.2 Experiments on Cloud Infrastructure 52
5.2.1 Amazon Elastic MapReduce 52

5.2.2 Windows Azure HDInsight 55
5.3 Concluding Remarks 57

6 CONCLUSIONS 59
6.1 About the Results 59
6.2 Further Work 60

REFERENCES 61

1 INTRODUCTION

1.1 Motivation

In the last years, Cloud Computing has become a key technology that made possible to run applications without needing to deploy a physical infrastructure. Cloud Computing, as defined in (MELL; GRANCE, 2011) by the U.S. National Institute of Standards and Technology NIST, is a computing model enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Also, the U.S. Department of Energy, DoE, addresses in (YELICK et al., 2011) how the Cloud Computing model can bring to the general user a big gain in terms of scalability, usability and flexibility of a computing infrastructure, which is a very positive feature for a computing system, however, as a negative feature, it also says that a Cloud Computing model still lacks a good performance management.

One of the key features of Cloud Computing is the one of a way for understanding the Cloud infrastructure as a shared pool of configurable computing resources, this means that Cloud Computing users and system managers can define the values of the system characteristics of their virtual machines with the goal of meeting application requirements and user constraints. Research work like (BOUTABA; CHENG; ZHANG, 2012) shows Cloud Computing nowadays as a computational paradigm becoming more present in many environments and applications, as well as being said to be by far the most cost-effective technology for hosting Internet-scale services and applications.

Virtualization is an essential technology for Cloud Computing. It introduces an abstraction layer that logically separates each virtual machine, specifically their resources like RAM, CPU, storage, etc., as if they were physically isolated, allowing Cloud providers to offer services in models as Infrastructure as a Service *IaaS*. The capability provided in *IaaS* to users, as described in (MELL; GRANCE, 2011), is to provision features as processing, storage, networks, and other fundamental computing resources, separately or associated with virtual machine instances where the consumer is able to deploy and run arbitrary software defined by himself. Cloud Computing offers its users the advantage of being able to manage distributed systems as a collection of virtual machines forming a cluster that can be modified, and therefore optimized for a certain application. One of the constraints of this approach is that we have to keep in mind that it would require a Cloud-specific capacity planning in order to specify parameters of virtual machines, namely, number of cores of the processor, RAM, network bandwidth and some others, according to the requirements of the application. This customization of virtual machine clusters would have to be specific of the application, since not all applications behave the same way, in other words, there is no single way to tune a virtual machine (HERODOTOU; DONG; BABU, 2011). These clusters can be implemented on physical machines as a normal implementation or they can be implemented on virtual machines, as they are the basic component of a Cloud Computing environment to fully utilize the system resources, improve reliability and save power (IBRAHIM et al., 2009).

MapReduce is a programming model and an associated implementation for processing and generating large data-sets proposed by Google. MapReduce programs can be parallelized and scaled rapidly and automatically, and they are intended to be executed on a multi-computer environment (DEAN; GHEMAWAT, 2008), as a grid or a cluster. MapReduce performance depends, amongst other things, on the type of data that is going to be processed, so various types of workloads can have different characteristics. It is important to consider a number of different kinds of workloads. In (ABAD et al., 2012) authors describe the parameters that can characterize a Map Reduce workload and how to synthesize it. There are some variable parameters on the MapReduce application that can also be changed or *tuned* in order to change the performance, specially the execution time. Research in (BABU, 2010) establishes a number of parameters that have to be configured by the administrators of a Map Reduce application, in a way that is time-effective. As said previously and shown in research as (HERODOTOU; DONG; BABU, 2011) and (BOUTABA; CHENG; ZHANG, 2012), an optimal cluster size should save money by optimizing infrastructure Cloud resources for the MapReduce user and the Cloud provider. This optimization can let users to take a better advantage of the resources they are using in the Cloud.

The work presented in (BOUTABA; CHENG; ZHANG, 2012) shows a survey of cloud computational models and heterogeneity challenges. The heterogeneity problem of workloads like job scheduling, task and data placement, resource sharing and resource allocation, and the problems of machine capabilities like varied CPU, memory, I/O speed, and network bandwidth capacities are described as a real challenge for researchers in this field; and further research is encouraged.

This master's thesis is motivated by the need of being able to predict the behavior of a distributed application running in a Cloud Computing environment in terms of execution time that would allow users to perform capacity planning and so plan their virtual clusters in a cost and time-effective way. The main objective is to present a way of predicting the execution time of a distributed application. MapReduce has been chosen as the major application for its nowadays importance in the scientific media. This research's main approach is to develop a capacity planning model as was described and validated by the scientific community in (CARRERA; GEYER, 2013).

1.2 Objectives and Contributions

The main objective of this work is to present a way of predicting the execution time of a distributed application when run in different environments. The selected distributed application for this work is a MapReduce application, but further work with other kind of distributed applications is encouraged as well. The prediction will be done with a formula that computes the execution time t of the application as a function of four variables:

1. W, the amount of workload the application is intended to process;
2. p, the number of computers the application will be run on top of, since it is a distributed application;
3. A, the type of the MapReduce application, since Map Reduce is a very versatile framework and many different applications can be programmed with it, and A will

be expressed as a relation between the number of operations per GB the application has to perform in order to process the W amount of data in Map and Reduce phases; and,

4. *T*, the capacity of the p computers that compose the cluster on top of which the application will be run, T will be expressed in terms of the number of operations per second that every computer in the cluster is capable of perform.

Thus,

$$t = f(W, p, A, T) \tag{1.1}$$

How and why these parameters were chosen will be fully explained in the following chapters.

The contribution of this work is to help users to know *'a priori'* how long it will take for the application to run, so they can predict how much money they will spend when running in a commercial Cloud Computing environment, and find a configuration that can save them money.

The experimental results of this work are also an important contribution showing that it is possible to model the behavior of a MapReduce application, and how the performance of a MapReduce application in terms of execution time changes when running in different environments.

1.3 Relation with GPPD's research

In GPPD (*Grupo de pesquisa em Processamento Paralelo e Distribuído*), since 2010, there has been several studies and research about MapReduce. One of the most notables is MRSG (MapReduce over SimGrid) simulator published in (KOLBERG et al., 2013) developed to simulate the behavior of the MapReduce model under different scenarios. After that, a set of algorithms for improving MapReduce in heterogeneous environments were addressed in (ANJOS et al., 2012) with very successful results. And currently, the main focuses of in progress research are new algorithms for running Map Reduce on volatile environments and improve its behavior with fault tolerance techniques.

The present work is about modeling performance for distributed applications and the application of this models in Cloud environments, it is not a work about MapReduce performance or algorithms. It bases its approach on the MapReduce algorithms described and modeled in the cited research work; however, it does not pretend to test them nor improve them in any way. In this work we use MapReduce as an example application to test our approach about modeling the behavior of distributed applications in Cloud environments.

1.4 Organization

The remainder of the dissertation is organized as follows:

Chapter 2 talks about Performance Evaluation and how it is achieved, as well as Performance Modeling and Capacity Planning techniques and considerations, in physical infrastructures, and specially in Cloud Computing environments.

The Performance Modeling section will present the concepts of the *SUT*, the *CUS*, and some other topics, as well as the ways a mathematical model can be defined for a computer system.

In the Capacity Planning section it will be explained some considerations for capacity planning based on performance evaluation and a few research projects that aimed to perform capacity planning for MapReduce applications.

In the Performance Evaluation in Cloud Environments section, there will be the specific considerations for capacity planning in the cloud, and some research projects developed in the subject.

Chapter 3 talks about the related work concerning to modeling and assessing the MapReduce computing model.

The MapReduce section talks about how the MapReduce computing model works. After that, the Hadoop Overview section defines how Hadoop, one of the most known implementations of MapReduce functions, and the execution model time of a Hadoop application.

The Performance Evaluation and Modeling over MapReduce section, as well as the Benchmarking in MapReduce section talk about the benchmarks used to theorize MapReduce applications. There has been several benchmarks that deal with the different types of applications, some are theoretically designed and some others are obtained from actual production MapReduce job logs. It focuses in MRBS, a MapReduce Benchmarking Suite, that was used as an example of MapReduce applications to test the modeling approach in physical infrastructure.

Chapter 4 talks about the Proposed Model and the Experimental Design.

The Proposed Hadoop Performance Model section talks about the definitions used to model the performance, and specifically the execution time of MapReduce applications in the present work and in related work.

The Scenarios section talks about the characteristics of platforms where the experiments were run, as well as the applications used for the experiments. The Specification of Experiments section talks about the definition of System Under Test *SUT* and the Component Under Study *CUS* that will be used in the experiments of the present work, as well as how the values of the parameters from section 4.1 were specified and combined to form the experiments execution plan.

Chapter 5 talks about the results of the experiments, as well as how the model was assessed against experimental results. The results of the tests in every environment are explained, as well as it is commented on what were the simplifications to the model led from the experimental data.

It has two sections, the first section talks about how the formula was obtained and how it was assessed in physical environments and the second sections explains the formula for Cloud Computing environments.

And finally, Chapter 6 talks about the conclusions of this work, how the results show that a MapReduce application can actually be modeled in a mathematical formula with a general form for physical and cloud platforms and most kind of applications, and how a user can benefit from this research and what are the next steps to continue.

2 PERFORMANCE EVALUATION

This chapter talks about Performance Evaluation and how it is achieved, as well as Performance Modeling and Capacity Planning techniques and considerations in physical infrastructures, with a special section on Cloud Computing environments.

2.1 Introduction

Performance Evaluation of a system is an important matter in computer science, specially when trying to get out the maximum performance of a system. The process of evaluating the performance of a given system begins by defining its characteristics. As exposed in (JAIN, 1991), the System Under Test *SUT* is defined as the collection of parts, treated as a whole, with a specific input and output data that will be assessed; and similarly, the Component Under Study *CUS* defines the part of the system whose parameters are going to be manipulated in order to cause changes in the performance of the system. The specifications of the *SUT* and *CUS* for this research project will be presented further in detail in chapter 4.

Performance Modeling refers to developing a mathematical model to express the performance of a system as a mathematical function that computes one or more output variables of the performance, based on relations among input parameters. The Performance Modeling section will present the concepts of the *SUT*, the *CUS*, and some other topics, as well as the ways a mathematical model can be defined for a computer system.

Capacity Planning refers to knowing what values for the parameters of the infrastructure of a system can achieve a desired performance (MENASCÉ et al., 2004). In the Capacity Planning section it will be explained some considerations for capacity planning based on performance evaluation and some research projects aiming to perform capacity planning for MapReduce applications.

In cloud environments, the capacity planning has to deal with specifying the virtual infrastructure that will ensure the best performance in terms of execution time and money expended when executing on a public/commercial cloud infrastructure. In the Performance Evaluation in Cloud Environments section, the specific considerations for capacity planning in the cloud, and some research projects developed in the subject will be described.

2.2 Performance Modeling

In Computer Science, Performance Modeling is a discipline that refers to develop a mathematical model that can compute one or more output variables dealing with the performance of a system as a mathematical function of one or more input variables referring to characteristics of: the system, the workload, the users behavior and/or other input conditions (ALMEIDA; MENASCÉ, 2002). Analytical performance models capture fundamental aspects and relate to each other by mathematical formulas and/or computational algorithms. When developing performance models it is important to know conditions like: system parameters, characteristics of a system that affect performance; resource parameters, features of the resources that affect performance; and workload parameters, intensity and service demand from the workload characterization.

In the methodology specified in (ALMEIDA; MENASCÉ, 2002) the first step is to specify the objectives of the system in terms of performance metrics, for example, defining the performance as the number of cache hits/misses, the amount of write/read operations, or the execution time of an application, amongst other metrics. One critical goal when analyzing and designing IT systems is guaranteeing the performance objectives are satisfied, and it is important to minimize the amount of guesswork (MENASCÉ et al., 2004); in this matter, IT systems users tend to be more concerned on Quality of Service *QoS* than technical issues, normally, they do not worry about numerical metrics but they do worry about their experience in terms of timing (response time, read/write time, execution time, etc.). So, performance goals should be stated in a simple manner and quantified with numbers in order to be measured and compared with the results.

Following the specification of the objectives, the system should be understood as a collection of components (subsystems) following a queue in which the n-*th* subsystem receives the output queue of the n-1-*th* subsystem and it provides its own output queue to the n+1-*th* component. The performance definitions established in the previous step should relate to the component of the system that is in charge of controlling that metric, or the system itself when the metric is related with the performance of the whole system. Also, in (HARBAOUI et al., 2009) performance prediction tools are based on a performance model of the system. The general approach is to decompose the system in black boxes, with each one studied in isolation and modeled with a queuing model.

After the characterization of the performance, then the workload has to be characterized as well, because the performance depends heavily on how much load it receives. In order to do this, a benchmark can be used as a workload model. A benchmark is a term for defining the workload used when comparing the performance of two or more systems by measurements (JAIN, 1991). Benchmarking is useful when the system has defined a workload. For example, for MapReduce applications a benchmark composed by typical applications can be used to test a computer system in which a MapReduce application is to be run.

A workload model is a representation of the real load given to the system that should be compact and representative of the actual workload in production environments. There are two kinds of workload models, the natural models which are constructed from components of the real workload, they are also called natural benchmarks; and the artificial models, theoretical models that aim to test the whole set of features a certain system can have (JAIN, 1991).

After the workload characteristics are specified, the system has to be measured in order to obtain the values of the workload characteristics. To characterize a workload, some specifications must be given, such as its quantity, the number of service requests, or specifying which are the resource demands. When choosing the parameters to characterize the workload, it is better to use the parameters which depend on the workload rather than on the system.

Following, with all the information obtained in previous steps, a performance model can be developed using quantitative techniques. Since it was settled previously that a performance model can compute a set of performance characteristics as function of a set of characteristics of the system, the workload, and/or other input conditions, the quantitative techniques should use the relations between those two sets and relate them using a mathematical model. This model is understood as the Performance Model of the system. The data that will be used to relate the two sets of input and output characteristics ought to be obtained via theoretical deduction or experimentation. In this work it will be used the experimental approach. The developed model has to be proven and validated. Depending of the nature of the model, also its proof and validation can be theoretical or experimental.

The validation of a performance model can be done predicting the system performance, which would be to determine how a system will react with a specific workload or when changes in load levels occur, and compare the prediction with actual measurements of the performance afterward. The results of the comparison will give us the idea of a reliable model under some conditions, called scenarios. The analysis of performance scenarios will establish conditions in which the performance will achieve desirable or undesirable performance levels in the future (MENASCÉ et al., 2004).

2.3 Capacity Planning

In the study of (ALMEIDA; MENASCÉ, 2002), the design of IT systems must be done with service levels in mind, a designer of IT systems must know their limits *'a priori'*. The term Capacity Planning refers to knowing what values for the parameters of the infrastructure of a system can achieve a desired performance.

Capacity Planning is described in (JAIN, 1991) as a process where computer system managers and engineers can ensure that adequate computer resources will be available to the users of the system to meet future workload demands in a cost-effective way, while meeting the performance objectives. Capacity Planning can be understood as determining the size of the components of a computer systems that will allow performing in a desired way with the minimum waste of computational and time resources. The main goals of Capacity Planning are described as improving performance availability, reliability, security and cost.

Also, in (MENASCÉ et al., 2004), authors define the process of Capacity Planning as to properly design and size a computer system for a given load condition, allowing computer system engineers to make a full use of the capacity of a system. They also describe a process called *Performance engineering* which can be understood as applying the concepts of Capacity Planning to the whole lifespan of a computer system. Performance engineering has two objectives: to predict the level of performance a system can have during its lifespan, and to provide recommendations to accomplish the optimal performance level.

Some of the problems concerning Capacity Planning are the lack of a standard definition of capacity, but it usually can be understood as maximum throughput of a system; also, there is no standard workload unit, because of the variability of environments, and this is something that has to be specified for every system.

The steps for Capacity Planning are in many ways similar as the ones for Performance Modeling, described in the previous section. For achieving capacity planning, in the same way as for performance modeling, it is required to fully understand the behavior, architecture and infrastructure of the system, breaking its complexity, and this can be achieved by analyzing its components; and afterward, analyzing the functionality of each component, evaluate its requirements, and design systems that will meet user's expectations.

The steps for Capacity Planning can be summarized as: to instrument in the first place the system with probes, using software or hardware components that can measure the performance of the system in some subsystems; these probes will monitor system usage under variations of the workload. The variation of the workload should be characterized as well as the workload itself using models. Often, queuing models are used for sizing because capacity planning models are general, coarse and system independent or less specific. Afterward, we should be able to predict performance under different alternatives. The final step in Capacity Planning is to select the lowest cost and/or the highest performance alternative (MENASCÉ et al., 2004). Capacity Planning relies in many ways on Performance Modeling in order to fully understand the system and its workload. There cannot be an appropriate Capacity Planning of a system without good performance and workload models.

2.4 Performance Evaluation in Cloud Environments

In cloud environments, Capacity Planning refers to specify the virtual infrastructure that will ensure a desired performance in terms of execution time and money expended, when a public cloud infrastructure is chosen. In this section, the specific considerations for capacity planning in Cloud Computing environments will be explained, as well as some research projects in the subject.

According to the Magellan Report, a document presented by the U.S. Department of Energy DoE in 2011 (YELICK et al., 2011), a Cloud computing model offers to the users a big advantage in terms of scalability and usability of a computing infrastructure. The goal of the Magellan Report was to research the potential role of Cloud computing in dealing with the computing needs for the DoE, related to serving the needs of future data-intensive workloads. The main findings of the Magellan Report concerning to the present project can be listed as:

- Cloud computing has won attention from industry and academic perspectives in research, as it has shown its capacity to address a broad array of computing needs.

- Developing an application in a cloud computing environment can require significant initial effort and skills in order to port them to these new models.

- A cloud computing model still lacks a good management of security and performance.

- A cloud performance management model should guide the cloud user in the capacity planning process of its virtual machine cluster in order to have a time and cost-effective performance for its application.

- The key economic benefit of clouds comes from consolidating physical resources across a broad community, which results in higher utilization rates of infrastructures.

- Cloud computing is ultimately a business model, but cloud models often provide additional capabilities and flexibility, helpful to certain workloads.

In a private cloud environment, as it is defined in (MELL; GRANCE, 2011), parameters of the virtual machine images, which are specifications of the virtual hardware that will be provided for the virtual machines, such as: number of cores of the processor, amount of main memory, bandwidth capacity and some other features can be freely managed according to the settings defined by the system administrator; on the other hand, in a public cloud environment, this ability is restricted by the available instance types defined by the provider. A Capacity Planning approach in a public cloud environment will have to be restrained to choose between the possibilities made available by the provider.

Research projects like (HERODOTOU; DONG; BABU, 2011), as well as (BOUTABA; CHENG; ZHANG, 2012) state that an optimal cluster size should save money by optimizing infrastructure cloud resources for the cloud user and provider. Also, in (MIETZNER; LEYMANN, 2008) the vision of a Software as a Service that requires Infrastructure as a Service level management is described as very important research field and an interesting trend in cloud computing. This vision relates to the resource provisioning based on the application performance. Therefore, this optimization can let users to take a better advantage of the resources they are using in the cloud.

In particular, (HERODOTOU; DONG; BABU, 2011) uses a training workload, to asses the performance of the cluster, and suggests computing the performance as a function of input data, resources and the application configuration.

Other study, exposed in (IBRAHIM et al., 2009), shows virtual machines that are tuned for data-intensive applications, and the master node is shown as a single point of failure (SPoF) in a MapReduce application. In that case, the master node is recommended to be instantiated on a physical machine rather than a virtual machine. Said work does not consider a cloud computing middleware, as it works directly with XEN based virtual machines. In a cloud computing environment this way of thinking is valid, but will not necessarily present the same results, and likely the experiments will not be the same either.

In (MIETZNER; LEYMANN, 2008), the approach of understanding an application offered in a *SaaS* model, can benefit from management at infrastructure level, or *IaaS* management. Provisioning infrastructure is described as very important research field for *SaaS* users and providers and a trend in Cloud Computing. This vision relates to automatic resource provisioning based on the performance of the application. Authors propose a whole architecture for provisioning infrastructure, which is based on Web services and workflow technology, so it can be possible to allow *SaaS* application providers to specify generic installation and maintenance flows to be independent from the underlying provisioning engines.

In the study of (WANG; HUANG; VARELA, 2010), the impact of VM granularity on workload performance in cloud computing environments shows that the number of VMs per physical machine has an actual significant impact on the performance of the virtual machines. Authors use HPL, which is a portable implementation of the high-performance Linpack benchmark for distributed-memory computers, and a web server as representative workloads. Their results show the optimal number of VMs per physical machine in a cluster. The results suggest that there can be scenarios, like when having tightly coupled computational workloads, where is best to change dynamically VM granularity to improve performance, and other scenarios, where VM granularity would have to be static for energy savings, like when having loosely coupled network intensive workloads.

In (IBRAHIM et al., 2009), the evaluation of virtual machine capacity planning for running MapReduce applications is shown, and authors state that it is feasible to demonstrate the applicability of MapReduce on a virtualized data center. A virtualized data center is a way of seeing the virtual machine instances of applications running in a Cloud environment. The study shows that there is a tradeoff between poor performance and resource utilization. Virtual machines can be used to fully utilize the system resources, which is one of the most important advantages in Cloud computing environments, along with easing the management of such systems, improving the reliability, and saving power. However, this full resource utilization can compromise the performance of a given application when run in a virtualized environment because of the different processes competing over the shared resources. The experiments using MapReduce applications as a workload model. The authors relate a number of issues when running data intensive applications using Hadoop in virtual cluster. Those issues include:

- Due to VMs being highly prone to error, it could be useful to separate the permanent data storage from the virtual storage associated with VM. Similar to the Elastic MapReduce service from Amazon, separating the data storage related to MapReduce by using the Amazon S3 service for that purpose.
- Using VM as execution unit will allow to define using VM migration as a replacement of the existing fault tolerance mechanism represented as speculative tasks.
- It is feasible to use VM in data intensive computing systems to fully utilize the physical node resources, using VM only as a computation unit for the data located on its physical node.
- VMs within the same physical node are competing for the node I/O, causing poor performance.
- As the master node is a single point of failure for the MapReduce infrastructure, and a VM is being highly prone to failure, it would be recommended to allocate the master node in a physical machine, or use VM checkpointing to implement a more reliable master.

In (ANDRZEJAK; KONDO; ANDERSON, 2010), PCs are considered the processing tier of a web service. In a Cloud Computing environment the physical infrastructure can be understood as a tier of aggregated CPUs that exclusively perform computation. This idea of concieving the PCs as a processing tier, isolated from other tiers is very interesting and makes the performance modeling easier. Also, it is said that the mechanisms that could deal with assessing cost of using the system and the system's reliability could be very valuable for users seeking to lessen their costs while keeping reliability at a high level. In this schema, checkpointing can be used to minimize the cost and volatility of resource provisioning.

In (ANDRZEJAK; KONDO; YI, 2010), it is shown that the mathematical model for the execution time of a distributed application can be built. Said work addresses the challenge of determining bid prices of AWS Spot Instances that could minimize costs for a user meeting Service Level Agreement (SLA) restrictions by proposing a probabilistic model for optimizing costs, performance and reliability. The probabilistic model would rely on user and application requirements and dynamic conditions computed from real instance price traces and workload models and would guide users how to optimally bid on Spot Instances.

The work (IOSUP et al., 2011) aims to answer the question of that if the performance of cloud computing environments can be sufficient for scientific computing based on

Many Task Computing (MTC). The compute performance of cloud environment is tested and compared with other scientific computing environments such as grids and parallel production infrastructures. Authors find that, even if current cloud computing services can be considered as insufficient for scientific computing at large, they might be a fair solution for scientific applications needing resources instantly and/or temporarily. The concept of MTC is explained as loosely coupled applications comprising many tasks. In another work, In (IOSUP; YIGITBASI; EPEMA, 2011) authors analyze the dependability of cloud services, establishing that finding out the performance variability is significant, saying that there is, indeed, performance variability in commercial clouds and it can be an important factor when deciding between cloud providers, and that there is no accepted comprising model for commercial clouds.

Similarly, in (FOSTER et al., 2008), authors explain the main differences between a Grid and a Cloud Computing environment, presenting that the vision in both sides about reducing costs for computing problems are equally important, the clusters are expensive to maintain for a provider, and that is why the clouds are more accessible for the scientific and industrial world. The cloud brings new problems like security issues. But in a general view the problems are similar.

In (ARMBRUST et al., 2010), the Top10 obstacles and opportunities for Cloud computing is shown. Authors explain that multiple virtual machines VMs can share CPUs and main memory surprisingly well in cloud computing, but that network and disk I/O sharing is more problematic, with the resulting performance variations in I/O more frequently than in main memory performance. This demonstrates the problem of I/O interference between virtual machines. Authors also establish the opportunity to improve architectures and operating systems to efficiently virtualize I/O channels.

In (DEELMAN et al., 2008), it is said that the cost of running an application on commercial clouds depends on the consumed resources regarding computation, storage and communication. Different execution plans, using different mixes of infrastructure and resources with the same application may result in significantly different performance values, and consequently different costs. Authors modeled the use of AWS cloud structure by a real-life data-intensive scientific application, and simulated the performance trade-offs of different execution and resource provisioning plans. Said work shows that by provisioning the right amount of storage and compute resources, cost can be significantly reduced with no significant impact on application performance. Similarly, in (SHI; TAIFI; KHREISHAH, 2011), Timing Models are specified, which are formulas for Sequential and Parallel Time of distributed applications. Authors prove simple models using AWS EC2 nodes.

2.5 Concluding Remarks

In this chapter we show the relevance for the particular issue of determining *'a-priori'* the size of a virtual machine cluster for distributed applications to be run in the cloud, with the objective of having a time- and/or cost-effective performance. The approach intended in this work is a way to present a methodology that can guide users to find a capacity planning model for cloud computing applications.

3 RELATED WORK

This chapter talks about the computing model of MapReduce, the current research work on Performance Evaluation and Modeling of MapReduce, as well as research work on benchmarking for MapReduce environments.

The MapReduce and the Hadoop Overview sections talk about the MapReduce model, as well as Hadoop, one of the most known implementations of MapReduce, and how it works, The Performance Evaluation and Modeling over MapReduce section talks about research projects with the objective of the performance prediction of MapReduce applications, as well as the benchmarks used to theorize and tests MapReduce environments.

3.1 MapReduce

MapReduce is a largely used programming model for processing and generating extensive data-sets proposed by Google in (DEAN; GHEMAWAT, 2008). MapReduce offers a mechanism inspired in the Map and Reduce primitives from high-level languages like *Lisp* and *Haskell*. The MapReduce model isolates the programmers from the complexity of the parallelization and management of data. Also, MapReduce is mostly used in large clusters, with low-latency networks and low-cost local storage devices (DEAN; GHEMAWAT, 2008).

In MapReduce, applications must be written in the form of two functions: a Map function and a Reduce function. The Map function transforms the input data into tuples <*key, value*> in what is called the Map phase, and the Reduce function takes the resulting tuples and summarizes them according to the key value during the Reduce phase.

MapReduce works with a master/worker architecture. In this architecture, the master node splits the input data in a distributed file system among the worker nodes, thus spreading the entire job into smaller tasks running in parallel, which are assigned to the workers.

Each worker node performs the Map function to its corresponding data, which were assigned when splitting the input. These data are grouped in small parts called *chunks*. After the Map function is executed over each chunk, the resulting data is again spread across the network. A *key* set is assigned to each node, so it processes only the tuples containing its corresponding *key* value. This process is called Shuffle. After the Shuffle phase is finished, each node executes the Reduce function over its corresponding data.

The execution model creates a computational barrier, which allows to synchronize the tasks execution between the Map and the Reduce tasks. A Reduce task does not begin its processing as long all the Map tasks are not finished.

In MapReduce it is preferred the execution of local taks, using local data, avoiding the need to transfer large volumes of data in processing time (DEAN; GHEMAWAT, 2008).

3.2 Hadoop Overview

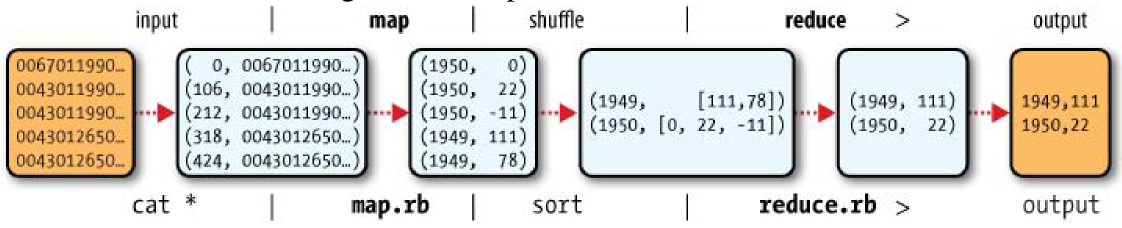
Hadoop (HADOOP, 2013) is a Java implementation of MapReduce proposed by the Apache Foundation. It is the most used MapReduce implementation (WHITE, 2012). Hadoop's API (Application Programming Interface) allows the user to program the Map and the Reduce functions.

Hadoop was originally thought to be run as a distributed application on top of a cluster of homogeneous machines, sharing a Distributed FileSystem, though several research projects have been developed to cover some of its issues in matters of fault tolerance, and execution on top of heterogeneous and voluntary computing environments.

Hadoop also uses a Distributed FileSystem, called the Hadoop Distributed FileSystem HDFS, which is likewise an open implementation of the Google FileSystem GFS described in (DEAN; GHEMAWAT, 2008). HDFS performs similarly to Google File System (SHVACHKO et al., 2010). HDFS and GFS are based on NFS working as a shared partition, accessible from all machines, in every hard disk of the nodes composing the MapReduce cluster. However, Hadoop can work with other distributed file systems.

The Hadoop execution algorithm is based on the MapReduce proposed by Google. According to the Hadoop execution algorithm (HADOOP, 2013), represented in figure 3.1 (WHITE, 2012), it has four defined phases:

Figure 3.1: MapReduce execution flow



(WHITE, 2012)

1. **Data distribution.** The first process is to copy the data for the MapReduce execution to the HDFS. The node in charge of this phase is the **master** node. The time length of this step depends on three factors: the amount of data that has to be copied through the network to the MapReduce cluster nodes, the number of said nodes, and the speed of the network that connects the nodes. Normally, all nodes would receive approximately the same amount of data workload. It can be expected that all nodes would receive approximately the same amount of data workload, because the approach used is write-once/read-many.

2. **Map phase.** The second process can be described as mapping the data, taking the input data, and, according to the programming of the Map phase, generating an output composed of <*key, value*> pairs. Each node has locally stored the data that is in charge of processing. Since the Map phase is run by all the nodes in parallel, the time length of this phase would depend on the amount of workload a single node has to process, the speed of said node to process said individual workload, and the number of operations that involve the processing of the individual workload.

3. **Shuffle.** Once the Map phase has reached a 5% of its progress, map outputs are sorted to ease the processing in the next phase. Output data from the Map phase

is transmitted over the network to one or other data node according to its content. The time lenght of this phase depends on the amount of data is transmitted and the speed of the network; however, since this phase runs in parallel with the Map phase, it only takes into account the shuffling of the last map outputs, corresponding to the last chunks.

4. **Reduce phase.** Finally, the reduce phase takes the output of the shuffle phase and process it according to its programming. The time length of this phase depends on the amount of data that is going to be processed, the number of operations that would be required to process the Reduce inputs, again, the speed of the node.

3.3 Performance Evaluation and Modeling over MapReduce

As explained in chapter 2, in order to develop a performance model of a system, it is necessary to understand and evaluate it against experimentation. In this section some research projects in the field of applying Performance Evaluation and Modeling techniques in MapReduce environments will be exposed, focusing on presenting metrics and parameters that have made possible previous performance models of MapReduce:

- Very extensive and comprehensive models of each phase of MapReduce are presented in (HERODOTOU, 2011), and discussed in (HERODOTOU; BABU, 2011). In said work, the MapReduce algorithm has two sets of tasks: Map, run first, and Reduce, run last. The Map task execution is divided into five phases: Read, Map, Collect, Spill and Merge; and in the same way, the Reduce task execution is divided into four phases: Shuffle, Merge, Reduce and Write.

  The performance is calculated as a function of three parameters: *d*, data properties, *r*, cluster resource properties, and *c*, configuration parameter settings. It actually shows a number of formulas that are useful to determine the execution time of a MapReduce job. The formulas calculate the time spent in every stage of the Map and the Reduce phases. They also introduce the concept of *'cost'* for the stages, understood as what determines the amount of performed operations on the CPU or the hard drive to process data or write/read data from/to the hard drive.

  Also, in (HERODOTOU; DONG; BABU, 2011), authors develop a way for perform Capacity Planning in Cloud computing environments for MapReduce applications. Authors deal with the problem of determining the virtual machine cluster resources and the MapReduce configurations to achieve user-defined requirements on execution time and economic cost for a given analytic workload.

  A system called the *Elastisizer* abstracts the complexities of MapReduce applications by profiling them and considering the discrete space that is the resource choices offered by *IaaS* cloud providers in their instance types.

- Another interesting work in the topic of MapReduce and Performance Evaluation is (TIAN; CHEN, 2011). In said work, authors present three cost functions that show a relationship between some characteristics of the MapReduce application and the time that takes for the application to execute. These three cost functions for MapReduce differ from each other by taking into account more or less complexity of the MapReduce application; for example, in the most complex cost function, authors consider a more complex Map function, including Copy and WriteBack sub-functions inside the Map phase, as well as the MergeSort sub-function inside the Reduce phase. A simpler function considers the complexity of the Reduce phase linear to the size of input data. And the simplest approach assumes that the amount of input data is fixed, so it is not necessary to take into account in the cost function.

The authors of (TIAN; CHEN, 2011) assess their performance model taking into account very specific parameters of MapReduce applications like the number Map and Reduce slots, the number Map and Reduce rounds, the complexity of Map and Reduce phases and the cost of scheduling all said processes. The model is assessed with some MapReduce applications, like WordCount, TeraSort, PageRank and Join in an in-house 16-node Hadoop cluster.

Research presented in (TIAN; CHEN, 2011) is quite exhaustive and comprehensive of the MapReduce applications; however, it can be mentioned as a limitation that it is very specific to the MapReduce framework, and the shown approach could not be used for other types of applications.

- Other work showing performance models of MapReduce is (KARLOFF; SURI; VASSILVITSKII, 2010), where MapReduce is expressed in three separated phases: Map, Shuffle and Reduce. Also, said work takes into account three parameters: *memory*, understood as the capacity of a cluster node to save information in its hard disk, *machine*, understood as the total number of nodes composing the cluster and *time*, as the running time available for the execution of the MapReduce application.

  Authors describe also the execution of a MapReduce job subject to a probability function of correctness, meaning that the job will be executed with no error in only some cases, defined by the probability function.

- Authors in (JIANG et al., 2010) determine 5 design factors utilized to run MapReduce application that affect the performance, namely:

  1. **I/O mode.** Which is the way the MapReduce application gets its input data: *direct* mode, when reading directly from the hard drive or *streaming* mode, when streaming by a communication scheme as TCP/IP or JDBC,
  2. **Indexing.** Even if MapReduce is typically used for unsorted data, when using sorted files, or database indexed tables, performance seems to improve,
  3. **Data parsing.** If there is any decoding procedures inside the Map phase that can be fixed or variable along the execution,
  4. **Grouping schemes** of input data, and
  5. **Block-level scheduling.** Showing that the scheduler performs faster when using larger blocks.

  Authors also investigated alternative implementation strategies for each factor, and how they affect the general performance of MapReduce applications. They have evaluated the performance of MapReduce with representative combinations of these five factors using a benchmark consisting of seven tasks. Amongst their findings is that, when using block-level scheduling, MapReduce can improve its performance when increasing the number of nodes in execution time.

- In (VERMA; CHERKASOVA; CAMPBELL, 2011), authors aim for designing a MapReduce performance model based on job profile and performance bounds of completion time for different job phases. This means they target to calculate the minimum number of maps and reduce tasks for a given time constraint and also that before running a job, they analyze the data and formulate a job profile that describes the typical behavior of said data. Without a time constraint, their approach is to compute job completion time as a function of the characteristics of input data set and allocated resources with performance models based on service level objectives

- In (VIANNA et al., 2013), authors express the need for specific analytical models of MapReduce that capture the sources of delays of processing jobs. The approach of (VIANNA et al., 2013) is to model the execution of MapReduce considering a series of parameters, related to the architecture of the cluster and the MapReduce application workload. Namely, the architecture parameters that are taken into account are the number of nodes and how many CPUs and disks each node has. Similarly, the considered MapReduce workload parameters are the number of Map and Reduce tasks, the number of threads per task and a service demand matrix

  The MapReduce model of (VIANNA et al., 2013) considers a queue of phases with a complex pipeline relating several stages of the MapReduce execution, and it uses a task precedence tree. The model is interesting and it is validated against experimental data and a simulator.

- Authors of (ZHANG; CHERKASOVA; LOO, 2013) address the problem of heterogeneity between the nodes of a Cloud Computing cluster, which is described as an important issue when dealing with virtualized infrastructure running distributed applications.

  In (ZHANG; CHERKASOVA; LOO, 2013), authors model MapReduce performance as a function of the number of tasks and nodes. The execution time of a MapReduce task is considered as bounded between a maximum and a minimum time. The described tasks comprehend three phases of the MapReduce execution: Map, Shuffle and Reduce; and there are upper and lower bound for the completion time of each phase. Authors indicate that the main factors impacting the MapReduce performance in Cloud environments are due to the heterogeneity of the cluster, which manifests in having different completion time of each one of the phases of the MapReduce execution.

- Finally, in (KONDO; ANDRZEJAK; ANDERSON, 2008) when modeling MapReduce applications, authors express CPU speed in FPOPS Floating Point Operations Per Second, which is a very handy way of generalizing the CPU speed, disregarding the amount of cores in it and understanding the CPU as a system which handles computing operations. Also, taking the phases of MapReduce as simple operations requiring an amount of time to be computed, regardless the type of data that is being processed is an interesting and simple way of understanding MapReduce and data processing.

  **This concept is going to be useful when modeling MapReduce applications execution in following sections.*

Table 3.1: Related work in Performance Evaluation of MapReduce applications

Author/Work	MapReduce modeling	Performance modeling
(HERODOTOU, 2011), (HERODOTOU; DONG; BABU, 2011), (HERODOTOU; BABU, 2011)	Map divided in 5 phases, Reduce divided in 4 phases	The performance is a function of: data properties, cluster resource properties, and configuration parameter settings.
(TIAN; CHEN, 2011)	MapReduce application parameters	Three cost functions relating execution time and MapReduce parameters.
(KARLOFF; SURI; VASSILVITSKII, 2010)	MapReduce in 3 phases: Map, Shuffle and Reduce. MapReduce is subject to errors based on probability functions.	Performance in 3 parameters: *memory*, *machines*, and *time*.
(JIANG et al., 2010)	MapReduce function of 5 applications parameters.	MapReduce application parameters affect performance.
(VERMA; CHERKASOVA; CAMPBELL, 2011)	MapReduce model based on job profiles.	Completion time bounds for each MapReduce phase.
(VIANNA et al., 2013)	MapReduce model based on precedence tree.	Performance as a function of architecture and workload parameters
(ZHANG; CHERKASOVA; LOO, 2013)	MapReduce affected by heterogeneity. MapReduce in 3 phases: Map, Shuffle and Reduce.	Performance modeled with upper and lower bounds due to heterogeneity.
(KONDO; ANDRZEJAK; ANDERSON, 2008)	Phases of MapReduce as simple operations requiring an amount of time to be computed.	CPU speed in FLOPS Floating Point Operations Per Second.

Table 3.1 summarizes the approaches of the authors cited in this research work. It can be seen a difference between the considerations to model the MapReduce execution; including or excluding execution phases. It is important to divide the MapReduce execution into several phases in the model because the algorithm of MapReduce describes it a queue where each process depends on its predecessor. A conservative approach not considering all the phases of MapReduce can be prone to errors and it would overlook some of the specific behavior of MapReduce applications; however, it can also be more general and comprehensive when modeling the performance for other types of applications, and not necessarily MapReduce.

In a general way, most authors take into account characteristics from MapReduce applications, which depend on the workload characteristics as well, and from the underlying cluster that executes the MapReduce program.

3.4 Benchmarking in MapReduce

A lot of research has been made in the field of characterizing and synthesizing the workload for MapReduce applications, some of the most relevant are cited and explained following:

- *MRBS*, specified in (SANGROYA; SERRANO; BOUCHENAK, 2012) as a MapReduce Benchmark Suite, is a collection of MapReduce benchmarks, that includes benchmarks covering five application domains and a wide range of execution scenarios like data-intensive and compute-intensive applications, batch and online interactive applications. *MRBS* can also simulate various types of faults at different rates, and considers different application workloads and data loads, and produces extensive reliability, availability and performance statistics. It can be executed over an on-premises private cluster, or in a third-party environment.

- In (CHEN et al., 2011), authors propose a framework for synthesizing workload for MapReduce applications. Its approach is an interesting way to work with MapReduce and, in the case of a very specific application to generate a workload for running tests with. In said work, authors analyze two production MapReduce traces to develop a vocabulary for describing MR workloads, and after that, they develop a framework that synthesizes workload with performance results closer to real workload. The analyzed data contains a log file of job submission and completion times, data sizes for the input, shuffle and output stages, and the running time in task-seconds of map and reduce functions.

- In the research exposed in (CHEN et al., 2010a) and (MISHRA et al., 2010), authors talk about 3 main characteristics for the MapReduce jobs: duration, CPU usage and Memory usage. They establish a few values for each characteristic in order to make their benchmark comprehensive of the majority of possible MapReduce applications that can be found in production environments. Namely, authors talk about small/large duration, small/medium/large CPU usage, and small/medium/large Memory usage. With the combination of the values, authors talk about 18 possible sizes of the jobs.

  In (CHEN et al., 2010b), the workload from a production environment is specified as a collection of vectors which differ from each other in inter-job arrival time, input size, shuffle-input ratio, and output shuffle ratio. Those metrics are very specific for MapReduce jobs. In their production environment traces they identify different performance gains when varying the underlying virtual infrastructure in a cloud environment, indicating how some combinations of sizes of virtual machine instances are equivalent in terms of performance.

  And in (CHEN et al., 2011), authors define the workload of MapReduce application with several characteristics like: *computation semantics*, which can be understood as a metric of how the Map and the Reduce functions will be programmed, *data characteristics*, which would depend on the type of data analyzed by the MapReduce application and *patterns* of arrival when the workload is real-time jobs. In said work, also it is been said that the disk I/O in the nodes is a bottleneck for the execution of MapReduce.

- Other work, like (BABU, 2010) and (HERODOTOU et al., 2011) present a way for self-tuning of Map Reduce applications. In (HERODOTOU et al., 2011) authors present a system that can automatically change some parameters of the MapReduce application in execution time depending of the running performance. Said work is based on (HERODOTOU; DONG; BABU, 2011) where the *Elastisizer* is used to help the on-time cluster tuning. Conversely, in (BABU, 2010) some techniques for tuning automatically the settings of Map Reduce programs are presented; in contrast, authors also say that it is possible that no single approach is good enough to set all high-impact job configuration parameters.

Table 3.2: Related work in Benchmarking for MapReduce applications

Author/Work	Characteristics
(SANGROYA; SERRANO; BOUCHENAK, 2012)	*MRBS*, a collection of MR benchmarks, covers 5 application domains and a wide range of execution scenarios.
(CHEN et al., 2011), (CHEN et al., 2010b), (CHEN et al., 2010a) and (MISHRA et al., 2010)	Framework for synthesizing workload for MapReduce applications based on 3 main characteristics for the MR jobs: duration, CPU usage and Memory usage Results are close to real workload.
(BABU, 2010) and (HERODOTOU et al., 2011)	Self-tuning of Map Reduce applications. Can automatically change parameters of the application in execution time depending of the running performance.

Table 3.2 summarizes the related work developing benchmarks for MapReduce applications. It can be seen that MapReduce benchmarks attempt to include the most known types of MapReduce applications. Benchmarks show a way to synthesize the workload of MapReduce applications and they are either based on analysis of production traces or based on studying the characteristics of the most common applications.

3.5 Concluding Remarks

In this chapter, it could be seen that there has been a lot of research about performance evaluation, performance modeling and benchmarking in MapReduce. The large amount of research in this field shows that it is a matter of interest and that it can be achieved in several different ways.

The approach intended in this work is to model the execution time in general terms, assigning values to the computing capacity of the MapReduce nodes and the processing cost of the MapReduce applications.

Next chapter will specify the performance model and will talk about the definition of the experiments that were carried out to assess such performance model.

4 PROPOSED MODEL AND EXPERIMENTAL DESIGN

This chapter talks about the proposed performance model for Hadoop and the specifications of the performed experiments. Section 4.1 expresses a form of quantifying the execution time of a Hadoop application as a formula based on the described Hadoop algorithm in section 3.2. The Scenarios section talks about the platforms where the experiments were run, namely: `gradep`, Grid'5000, Amazon Elastic MapReduce, and Windows Azure HDInsight; and it also talks about the applications that were used as benchmark for the experiments. The Specification of Experiments section explains how the experimental design was defined as well as the values of the parameters used to assess the model described in equation 4.12.

4.1 Proposed Hadoop Performance Model

As we referred in section 2.1, in order to carry out a performance evaluation of a computer system, the System Under Test ***SUT*** and the Component Under Study ***CUS*** have to be specified. The ***SUT*** is defined as the set of parts, with its corresponding input and output data that is going to be tested. In our case of evaluating the performance of a distributed system, running a MapReduce application, we have defined the System Under Test ***SUT*** as the cluster of computers, running a set of MapReduce applications that uses the Hadoop framework. The input and output data will be specific for each application in the set, but in a general way, it will be a synthetic workload, specifically created for testing the applications.

Similarly, the Component Under Study ***CUS*** will also be the cluster of computers modeled as a black box. The reason for treating the cluster of computers as a whole is to be able to understand its behavior with a general view, and not introduce any bias that could make the model less general.

For the ***CUS*** there are 4 parameters for evaluation, namely:

1. W, the amount of workload the application is intended to process, expressed in units of storage;

2. p, the number of computers the application will be run on top of, since it is a distributed application, it is supposed to be run on top of a cluster of computers, expressed as an integer;

3. A, the type of the MapReduce application, since MapReduce is a very versatile framework and many different applications can be programmed with it. A will be expressed as a the number of operations per unit of storage performed by the application in Map and Reduce phases; and,

4. *T*, the processing capacity of the p computers that compose the cluster on top of which the application will be run. *T* will be expressed in terms of the number of operations per second that every machine in the cluster is capable of perform.

The first goal of this work is to obtain a mathematical model of the execution time of a Hadoop program, as a function of the 4 parameters indicated above. In order to build such a model, we have to make some considerations:

- Phases of the Hadoop algorithm, as they were described in section 3.2, follow a particular order;
- Even though the *Data Distribution* phase is considered in (WHITE, 2012) as a part of the Hadoop execution algorithm, it is not considered inside the execution time of a Hadoop program, since the execution logs do not quantify this time. Also, when working in a Cloud Computing environment, a provider does not charge a user for the time spent in this phase, so the duration of the *Data Distribution* phase can be disregarded;
- The *Shuffle* phase starts when the Map phase has reached a 5% of its completion, and runs simultaneously with the Map phase;
- For matter of simplifying the model, the *Shuffle* phase time will be considered only as the time when the *Map* phase has already finished;
- The *Reduce* phase starts only when the *Shuffle* phase has been fully completed;
- The *Map* and *Reduce* phases run in *slots*, that means that each node can run more than one Map or Reduce task in parallel; usually, the number of slots is related with the number of cores a node has.

Thus, the execution time of a Hadoop program can be modeled as the sum of the times of all phases:

$$t_{total} = t_{MAP} + t_{SHUFFLE} + t_{REDUCE} \tag{4.1}$$

The time of the Map phase t_{MAP} can be computed as the product of the time of a single Map task times the number of Map tasks per node:

$$t_{MAP} = t_{U_{MAP}} * \frac{n_{MAP}}{p} \tag{4.2}$$

The time of a single Map task $t_{U_{MAP}}$ is computed as the product of the amount of workload for a single Map task W_{MAP} (known as a *chunk size*) times the cost of a single Map task $C_{U_{MAP}}$ (expressed as a relation of the number of floating point operations FLOPs required by the Map task and the chunk size), and divided by the capacity T of a single node (the number of floating point operations FLOPs per second):

$$t_{U_{MAP}} = \frac{W_{MAP} * C_{U_{MAP}}}{T} \tag{4.3}$$

The number of Map tasks for a job is computed as the division between the total Workload and the *chunk size*:

$$n_{MAP} = \left\lceil \frac{W}{chunksize} \right\rceil \tag{4.4}$$

The time of the Reduce phase t_{REDUCE} can be computed as the product of the time of a single Reduce task times the number of Reduce tasks per node:

$$t_{REDUCE} = t_{U_{REDUCE}} * \frac{n_{REDUCE}}{p} \tag{4.5}$$

The time of a single Reduce task $t_{U_{REDUCE}}$ is computed as the product of the amount of input workload for a single Reduce task $W_{U_{REDUCE}}$ times the cost of a single Reduce task $C_{U_{REDUCE}}$, and divided by the capacity T of a single node (the number of floating point operations FLOPs per second):

$$t_{U_{REDUCE}} = \frac{W^{IN}_{U_{REDUCE}} * C_{U_{REDUCE}}}{T} \tag{4.6}$$

Replacing equations 4.3 and 4.6 into equations 4.2 and 4.5 we have:

$$t_{MAP} = \frac{W_{MAP} * C_{U_{MAP}}}{T} * \frac{n_{MAP}}{p} \tag{4.7}$$

$$t_{REDUCE} = \frac{W^{IN}_{U_{REDUCE}} * C_{U_{REDUCE}}}{T} * \frac{n_{REDUCE}}{p} \tag{4.8}$$

Also, the time for the Shuffle phase $t_{SHUFFLE}$ can be computed as the product of the size of the output data of a single Map task times the number of remaining Shuffle tasks divided by the bandwith of the network:

$$t_{SHUFFLE} = \frac{W^{OUT}_{U_{MAP}} * (n_{MAP} mod p)}{B} \tag{4.9}$$

The size of the output data of a single Map task $W^{OUT}_{U_{MAP}}$ can be computed as the division between the total amount of output data of the Map phase divided by the number of Map tasks:

$$W^{OUT}_{U_{MAP}} = \frac{W^{OUT}_{MAP}}{n_{MAP}} \tag{4.10}$$

So, replacing equations 4.7, 4.8, 4.9 and 4.9 in equation 4.1 we have:

$$t_{total} = \frac{W_{MAP} * C_{U_{MAP}}}{T} * \frac{n_{MAP}}{p} + \frac{W^{OUT}_{MAP} * (n_{MAP} mod p)}{n_{MAP} * B} + \frac{W^{IN}_{REDUCE} * C_{U_{REDUCE}}}{T} * \frac{n_{REDUCE}}{p} \tag{4.11}$$

Simplifying W^{IN}_{REDUCE} in 4.11

$$t_{total} = \frac{W_{MAP} * C_{U_{MAP}}}{T} * \frac{n_{MAP}}{p} + \frac{W^{OUT}_{MAP} * (n_{MAP} mod p)}{n_{MAP} * B} + \frac{W^{IN}_{U_{REDUCE}} * C_{U_{REDUCE}}}{T * p} \tag{4.12}$$

Equation 4.12 will serve as the performance model for MapReduce applications as a function of the parameters of the ***CUS***. Defining this equation was one of the main objectives of the present work.

4.2 Scenarios

4.2.1 Hardware description

In order to verify the model of equation 4.12, experiments have to be performed in several different environments, comprising private and public infrastructure, physical and virtual cloud infrastructure. The environments where the tests were run in, are described following:

1. **Cluster `gradep`**

   The `gradep` is a computer cluster on premises of Institute of Informatics *INF*, in the Federal University of Rio Grande do Sul *UFRGS* composed of 18 nodes in total. Each node has an Intel Pentium 4 2.79 GHz CPU with 2 GB in RAM, and a Gigabit Ethernet connection.

   Each time the applications were run, the worker nodes were randomly chosen amongst these 18 machines to avoid errors produced by running all times in the same cluster nodes.

2. **Grid'5000**

   The Grid'5000 (GRID5000, 2013) is a shared infrastructure built to support research in many fields, housed in several cities, each city hosting a site organized as a collection of clusters. It is developed by the project *INRIA ALADDIN* and supported by the CNRS, RENATER e several universities.

   The experiments were run in two clusters, one in the Nancy site and one in the Grenoble site:

   - In **Nancy griffon** nodes have an Intel Xeon L5420 2.5 Ghz processor, 16GB in RAM, 320GB in SATA II hard drive, and Infiniband-20G Gigabit Ethernet network interfaces. These nodes have an average processing power of 20 GFLOPS.
   - In **Grenoble edel**, nodes have an Intel Xeon E5520 2.27 GHz processor, 24GB in RAM, 64GB in SATA hard drive, and Infiniband-40G Gigabit Ethernet network interfaces.

3. **Amazon EMR**

   Amazon Elastic MapReduce EMR (EMR, 2013) is a web service from Amazon Cloud services that uses Hadoop to process data across a cluster of Amazon EC2 instances (EC2, 2013).

   For the purposes of this research work, the `m1.small` instance type was employed. EC2 instance types are specified in (EC2, 2013). The processing power of Amazon EC2 instances is expressed in ECU, which is a unit with the equivalent CPU power of a 1.0-1.2 GHz 2007 Opteron or Xeon processor as specified to Amazon EC2 documentation. The `m1.small` instance type, for example, has the processing power of 1 ECU.

   According to (IOSUP et al., 2011), the theoretical peak performance can be computed for different instances from the ECU definition: a 1.1 GHz 2007 Opteron can perform 4 flops per cycle at full pipeline, which means at peak performance one

ECU equals 4.4 gigaflops per second. This value will be used for the T parameter described in section 4.1.

4. **Windows Azure HDInsight**

   Windows Azure HDInsight (HDINSIGHT, 2013) is a Hadoop-based service from Microsoft Azure for running an Apache Hadoop solution in a Cloud Computing environment. HDInsight uses the General Purpose Instances from Microsoft Azure.

   In this work we used the Large (A3) compute instance type. According to the Windows Azure documentation, the A3 compute instance type has a 4-core 1.6 GHz processor, and 7GB in RAM.

4.2.2 Software description

In the same way that it is necessary to perform the tests in several different hardware environments, it is also necessary to test the performance model from equation 4.12 with different MapReduce applications. Varying the applications in the tests help the model to identify when it cannot be useful, or if it only serves for a certain type of applications. The applications used for the tests are described following:

- In the `gradep` cluster, two applications, named `sort` and `wordcount`, were used. These applications form part of the `text-processing` benchmark from the MapReduce Benchmark Suite *MRBS* described in (SANGROYA; SERRANO; BOUCHENAK, 2012).

  As it is explained in (O'MALLEY, 2008), the `sort` application takes an input of text registers, and sorts them depending on its contents. And the `wordcount` application, as explained in (DEAN; GHEMAWAT, 2008), takes a text input and counts the occurrences of each word, resulting in a sorted list of words indicating how many times they appeared in the input text.

  The `wordcount` application is *CPU bound*, meaning that the execution time is mainly limited by the processing power of the CPU, and the `sort` application is *CPU bound* and *IO Bound*, meaning that the execution time is mainly limited by the CPU and the IO system.

- In the Grid'5000 environment we used the `text-processing` benchmark from the MapReduce Benchmark Suite *MRBS* described in (SANGROYA; SERRANO; BOUCHENAK, 2012) that includes a mix of basic applications to process text input files. *MRBS* is a software that was programmed in Grid'5000, but can be run on top of over an on-premises private cluster, or in a Cloud environment.

  The `text-processing` benchmark includes `sort` and `wordcount` used in the tests with the `gradep` cluster.

- And finally, in the Cloud environments of *Amazon* and *Azure* it was used a log processing application, which basically is a text processing application similar to the ones used in the environments explained above.

  In 2013, a private company located in Brazil, asked the GPPD/INF research group for help to develop and test a MapReduce application to process large amounts of logs and to be run in a Cloud environment. The project supported the research presented in this dissertation by providing a use case for what it is intended to do, to predict the execution time of a MapReduce application running in a Cloud environment. The log processing application takes the log records and rearranges the contents of each text line, leaving the output slightly smaller than the output. The logs contain the same information but in a different order. More information about the application and the project can be found in (CARRERA et al., 2013).

4.3 Specification of Experiments

In (JAIN, 1991) authors describe several techniques to specify the execution of experiments. The details of the specifications for this work are described following:

4.3.1 Evaluation Technique

In the present work we used two evaluation techniques, one for elaborating the performance model in equation 4.12, and another one to assess said performance model. The evaluation techniques are described following:

In sections 4.1 and 3.2, *Analytical Modeling* was first used as an evaluation technique, where, based on theoretical models of MapReduce, a formula to model its performance was developed. Analytical Modeling characterizes for evaluating performance in a theoretical form, by looking at the definitions of the *SUT* and *CUS* and thus understanding and evaluating their behavior.

The second evaluation technique used in this work is *Experimental Measurement*. In this technique, the analytical/theoretical hypothesis is tested with experimentation. The experiments made to validate the analytical model in equation 4.12 are described in this chapter and their results are described in chapter 5.

4.3.2 Performance Metrics

As it has been discussed earlier in this work, and modeled in equation 4.12, the performance metric used in the experiments is the execution time of the applications, considered from the beginning of the first Map operation, until the end of the last Reduce operation.

Hadoop generates log records in every execution, these logs are recorded . In this logs there are timestamps for each process inside a MapReduce execution and we were able to calculate the execution time by simply subtracting the time.

Subtracting these timestamps return a total execution time in milliseconds that had to be transformed for the calculations in the next chapter.

4.3.3 Workload

In (JAIN, 1991), when the workload used in the experiments is a typical application, and it is applied to test a set of environments, it is called a benchmark. The benchmarks used in the experiments are specified in section 4.2.2. [1]

4.3.4 Parameter Values

As established in section 4.1, the System Under Test ***SUT*** considered for this work is a cluster of computers, forming a distributed system, running a distributed MapReduce

[1]In this work, it has been used the term *benchmark* as a group of applications that assess the performance of a system. Also, in (JAIN, 1991) the Workload type for Performance Evaluation techniques is called a *benchmark* when using typical applications with a synthesized input for evaluating the performance of a system. This last one is the intended concept of the word *benchmark* in this section.

application that uses the Hadoop framework. Is this particular case, the Component Under Study ***CUS*** will also be the cluster of computers.

The values for the 4 parameters for the ***CUS*** depend on the scenarios described in the previous section.

Table 4.1 shows the used values in the different scenarios.

Table 4.1: Parameter Values

Scenario Parameter	`gradep` cluster	Grid'5000	AWS	Azure
W	{1, 5, 10, 20, 25} [GB]	{1, 10, 100} [GB]	{1, 5, 10, 20, 25} [GB]	{1, 5, 10, 20, 25} [GB]
p	{4, 8, 12, 16} [nodes]	{10, 25, 50} [nodes]	{4, 6, 8} [nodes]	{4, 6, 8} [nodes]
A	text processing	text processing	log processing	log processing
T	`gradep`	griffon, edel	m1.small	A3

The values for W for the experiments using the Grid'5000 infrastructure, were based on the specifications of the `text-processing` benchmark of the MapReduce Benchmark Suite *MRBS*. All the other values for W were assigned based on the amount of available machines in the `gradep` cluster and in the Cloud environments.

In table 4.1, W, the amount of workload is expressed in GB; p, the number of nodes is an integer; A, the type of the Map Reduce application is expressed as an application type, while in the formula this categorical value has to be changed for a numeric one in $[flops/GB]$; and T, the node capacity is expressed as the name of the cluster the node belongs to, or the instance type if it belongs to a Cloud environment, while in the formula this categorical value has to be changed for a numeric one in $[flops]$.

4.4 Concluding Remarks

In this chapter, the performance model for MapReduce was presented. The model is a main objective of this work and it will be assessed with experimentation in order to see how it behaves and if it is capable o predict the execution time of a MapReduce application. Also, the experiments are specified using the concepts described in chapter 2, as well as the parameter values for the performance evaluation are described.

In the next chapter the results of the performed experiments will be presented.

5 RESULTS

In this chapter, the results of the experiments described in chapter 4 are presented, as well as how the mathematical model of the execution time was specified as a function of the variables. Section 5.1 talks about the experiments in physical clusters like `gradep` and Grid'5000, an explanation about the results and the obtained formula; and section 5.2 talks about the results in cloud environments of Amazon Web Services and Windows Azure, as well as its explanation about how the results were got and the formula.

The performed experiments had two objectives: the first one was to determine the values of A, the number of operations per unit of storage performed by the application in the Map and Reduce phases; and the second one is to assess the execution model of equation 4.12 to verify its capacity for predicting the behavior of MapReduce applications running in a Cloud environment.

One thing that was not said in the previous chapter is that the experiments in physical clusters like `gradep` and Grid'5000 do not form part of the objectives of the present work, the experiments were performed as a training and to see if it was possible to predict the execution time of a distributed application.

5.1 Experiments on Physical Clusters

5.1.1 Private Cluster `gradep`

As shown in table 5.1, in the `gradep` cluster, two applications were run to obtain and test the performance model. These applications were taken from the text-processing benchmark of MRBS (SANGROYA; SERRANO; BOUCHENAK, 2012). Tests were run with the combination of the following values for the parameters:

Table 5.1: Parameter values for `gradep` cluster

Parameter	Value
W	$\{1, 5, 10, 20, 25\}$
p	$\{4, 8, 12, 16\}$
A	sort, wordcount
T	`gradep`

Values for W and p in table 5.1 were arbitrarily established in function of the available nodes in `gradep` cluster. Values in table 5.1 were combined for each execution of the experiments, and each combination was run 20 times, for a total of 800 executions.

After the executions for the `sort` application, we noticed that the Map phase output W_{MAP}^{OUT} is proportional to the Map phase input W_{MAP}. This makes sense because the amount of Map output data would depend on the amount of Map input data, and the following relation can be established:

$$W_{MAP}^{OUT} \propto W_{MAP} \tag{5.1}$$

$$W_{MAP} = k * W_{REDUCE}^{IN} \tag{5.2}$$

And when replacing equation 5.2 into equation 4.12, it simplifies to:

$$t_{total} = \frac{W_{MAP}}{p*T} * (C_{U_{MAP}} + k * C_{U_{REDUCE}}) + \frac{W_{MAP}^{OUT} * (n_{MAP} mod p)}{n_{MAP} * B} \tag{5.3}$$

Without making these simplifications in equation 4.12, we could not be able to calculate the values for A. Equation 5.3 shows the linear relation between the execution time t and the workload W_{MAP}.

Also, since it was required to quantify the computational power of the cluster nodes, that can be done using a benchmark. However, the tests were already performing a benchmark over the cluster, so, with the experimental data, it was possible compute the relation between the cost for Map and Reduce phases $C_{U_{MAP}} + k * C_{U_{REDUCE}}$ and the computational power of the nodes T using the linear regression tool of the software R (R, 2011). The value of the relation between the cost for Map and Reduce phases and the computational power was computed to:

$$\frac{C_{U_{MAP}} + k * C_{U_{REDUCE}}}{T} = 6.05e^{-7} \left[\frac{s}{Byte}\right] \tag{5.4}$$

With an standard error of $SE = 1.15e^{-8}$ and a coefficient of determination of $R^2 = 0.88$. A value of $R^2 = 0.88$ gives the idea that the model is close to an optimal fit, but considering this model being general and that it has to serve different values, it can be recognized as a very good model. The final model for this type of application and environment remains like this:

$$t_{total} = \frac{W_{MAP}\,[Byte]}{p} * (6.05e^{-7} \left[\frac{s}{Byte}\right]) + \frac{W_{MAP}^{OUT}\,[Byte] * (n_{MAP} mod p)}{n_{MAP} * B \left[\frac{Byte}{s}\right]} \tag{5.5}$$

Following, in figure 5.1 it is shown the experimental data, grouped by the number of nodes used in each experiment and the corresponding line for the mathematical model with two auxiliary lines showing the 95% confidence interval of the model.

Figure 5.1 shows that the experimental results follow a clear trend that is followed by the model. A 95% confidence interval of the model means that, in average, 95% of the results will fall in the space between the two auxiliary lines.

After modeling the execution time, giving values to the formula of equation 5.3, more experiments were run to assess the model, with the results detailed in table 5.2.

Figure 5.1: Execution time vs. Workload for the `gradep` cluster

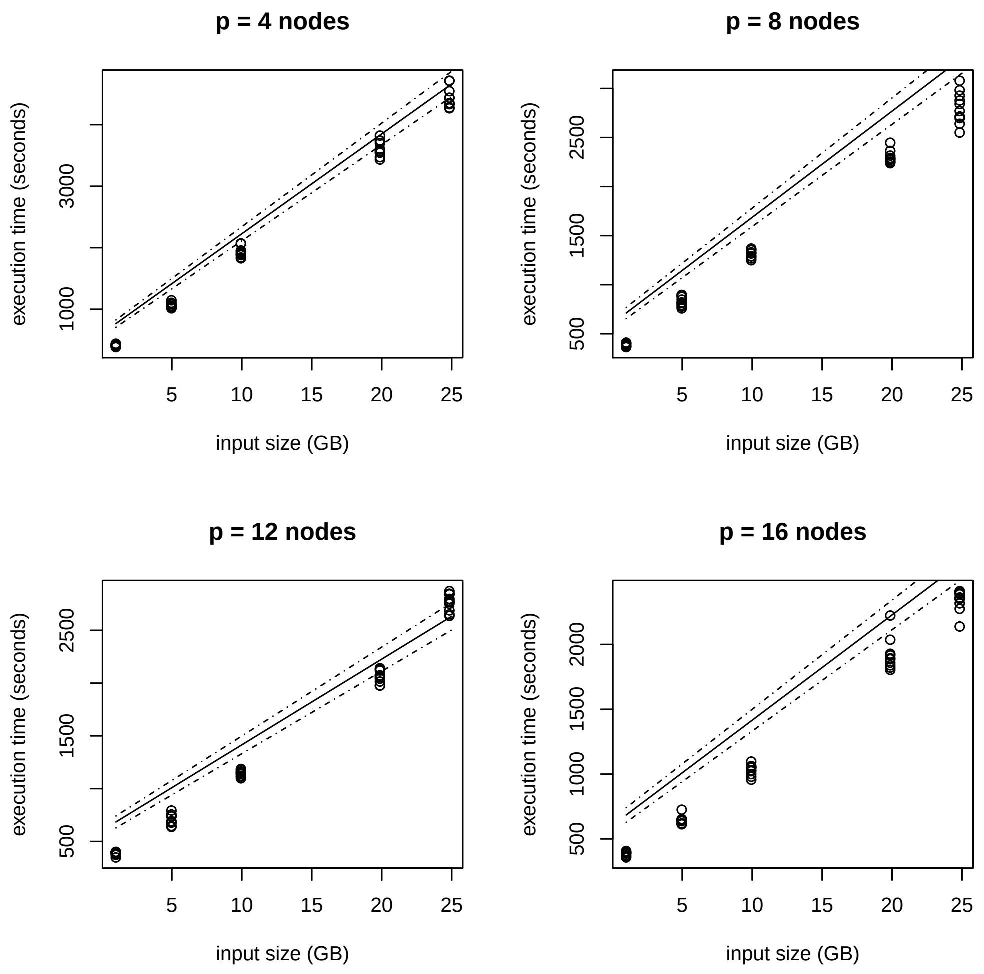


Table 5.2: Results for model assessing on the `gradep` cluster

Values	Predicted Value	Exp. Value	Error
W=1, p=6	401.6	430.7	7.24%
W=1, p=10	373.4	374.8	0.38%
W=1, p=14	339.7	387.3	14.0%
W=5, p=6	899.3	962.1	6.98%
W=5, p=10	752.9	734.7	2.42%
W=5, p=14	687.9	653.1	5.05%
W=10, p=6	1521.4	1442.0	5.22%
W=10, p=10	1227.3	1241.0	1.12%
W=10, p=14	1123.0	1250.3	11.33%
W=20, p=6	2765.6	2526.2	8.66%
W=20, p=10	2176.0	2454.4	12.8%
W=20, p=14	1993.3	2021.5	1.41%
W=25, p=6	3387.7	3392.4	0.14%
W=25, p=10	2560.3	2557.0	3.52%
W=25, p=14	2428.5	2296.9	5.42%

Results in table 5.2 show that the model from equation 5.3, using the values in 5.4, was actually able to predict the execution time of the MapReduce applications with a reasonable error. This error is small considering that when expressed in absolute value it reduces to a few minutes of execution. We have to keep in mind that these models are meant to be used in Cloud infrastructures, where a few minutes of difference do not mean excessive changes in the cost of using the platforms.

5.1.2 Grid'5000

Similarly to the experiments performed in the `gradep` cluster, the applications used to obtain and assess the model were taken from the text-processing benchmark from MRBS (SANGROYA; SERRANO; BOUCHENAK, 2012). In table 5.3, it can be seen the parameter values for the experiments on Grid'5000:

Table 5.3: Parameter values for Grid'5000

Parameter	Value
W	$\{1, 10, 100\}$
p	$\{10, 25, 50\}$
A	`text-processing`
T	griffon, edel

The text-processing benchmark contains 4 applications, from where there were as well chosen two: `sort` and `wordcount`.

Experiments were run on two different clusters from Grid'5000, namely, Griffon and Edel clusters. Each combination of the values described in table 5.3 was run at least 45 times, for a total of 1620 executions.

In the Griffon cluster, the computing power T was established with the value of 20 GFlops per second, the cost for Map and Reduce phases was computed to:

- $Cu_{MAP} + k * Cu_{REDUCE} = 5.73e^{-4}$, with a standard error of $SE = 3.01e^{-5}$ and a coefficient of determination of $R^2 = 0.32$ for the sorter application, and,
- $Cu_{MAP} + k * Cu_{REDUCE} = 9.70e^{-4}$, with a standard error of $SE = 4.59e^{-5}$ and a coefficient of determination of $R^2 = 0.55$ for the wordcount application.

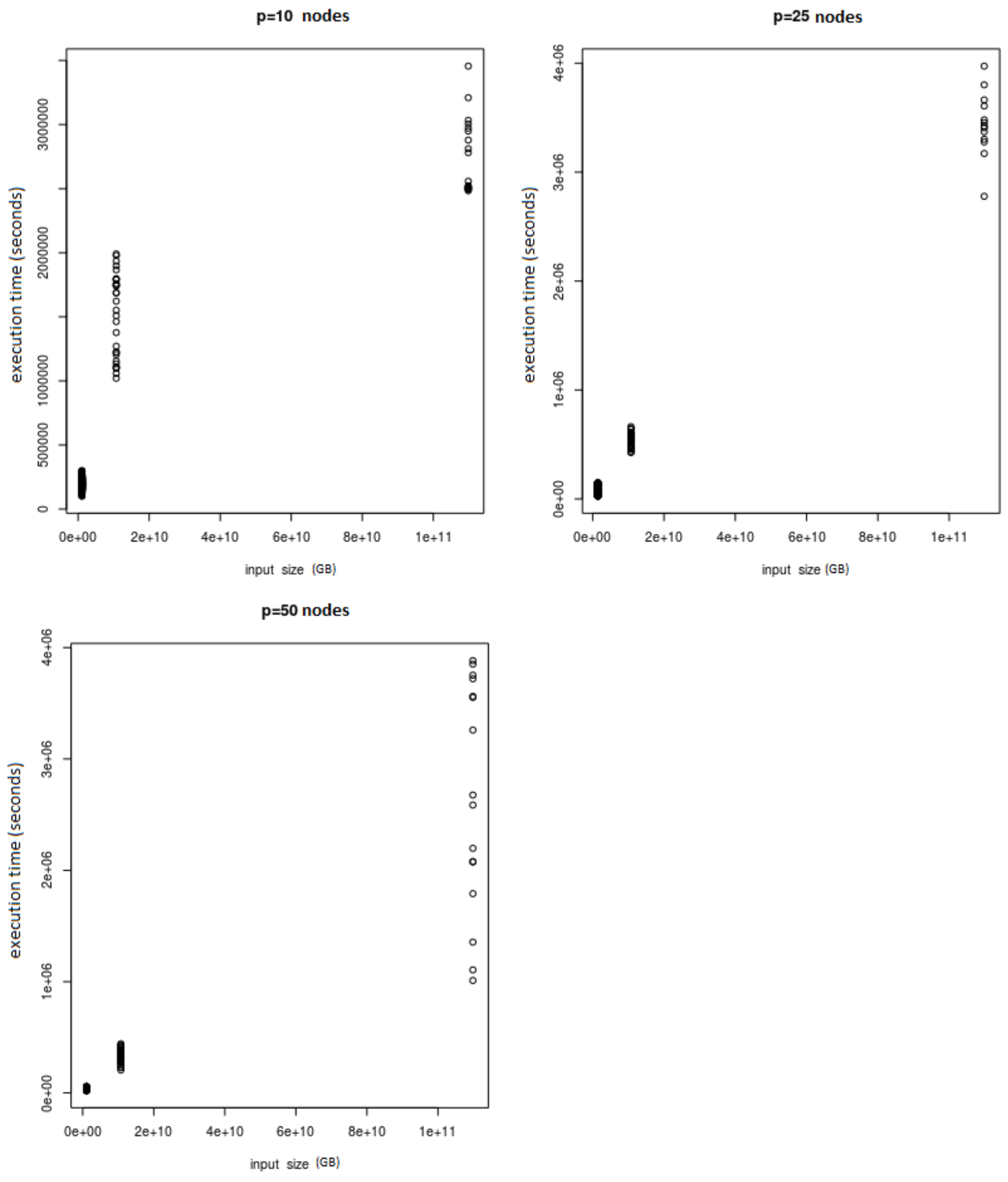
In the Edel cluster, the computing power T was established with the value of 16 GFlops per second, the cost for Map and Reduce phases was computed to:

- $Cu_{MAP} + k * Cu_{REDUCE} = 1.06e^{-3}$, with a standard error of $SE = 3.71e^{-5}$ and a coefficient of determination of $R^2 = 0.55$ for the sorter application, and,
- $Cu_{MAP} + k * Cu_{REDUCE} = 7.83e^{-4}$, with a standard error of $SE = 5.41e^{-5}$ and a coefficient of determination of $R^2 = 0.31$ for the wordcount application.

The values of 20 and 16 GFlops per second were taken from the Grid'5000 web page (GRID5000, 2013). The values for the cost for Map and Reduce phases were computed like in the previous subsection using the software R (R, 2011).

Figures 5.2 and 5.3 show the execution time for the applications run in Grid'5000, with the specifications shown in table 5.3. As it can be seen in figures 5.2 and 5.3, and in the

Figure 5.2: Execution time vs. Workload for the Griffon cluster running the Sorter application

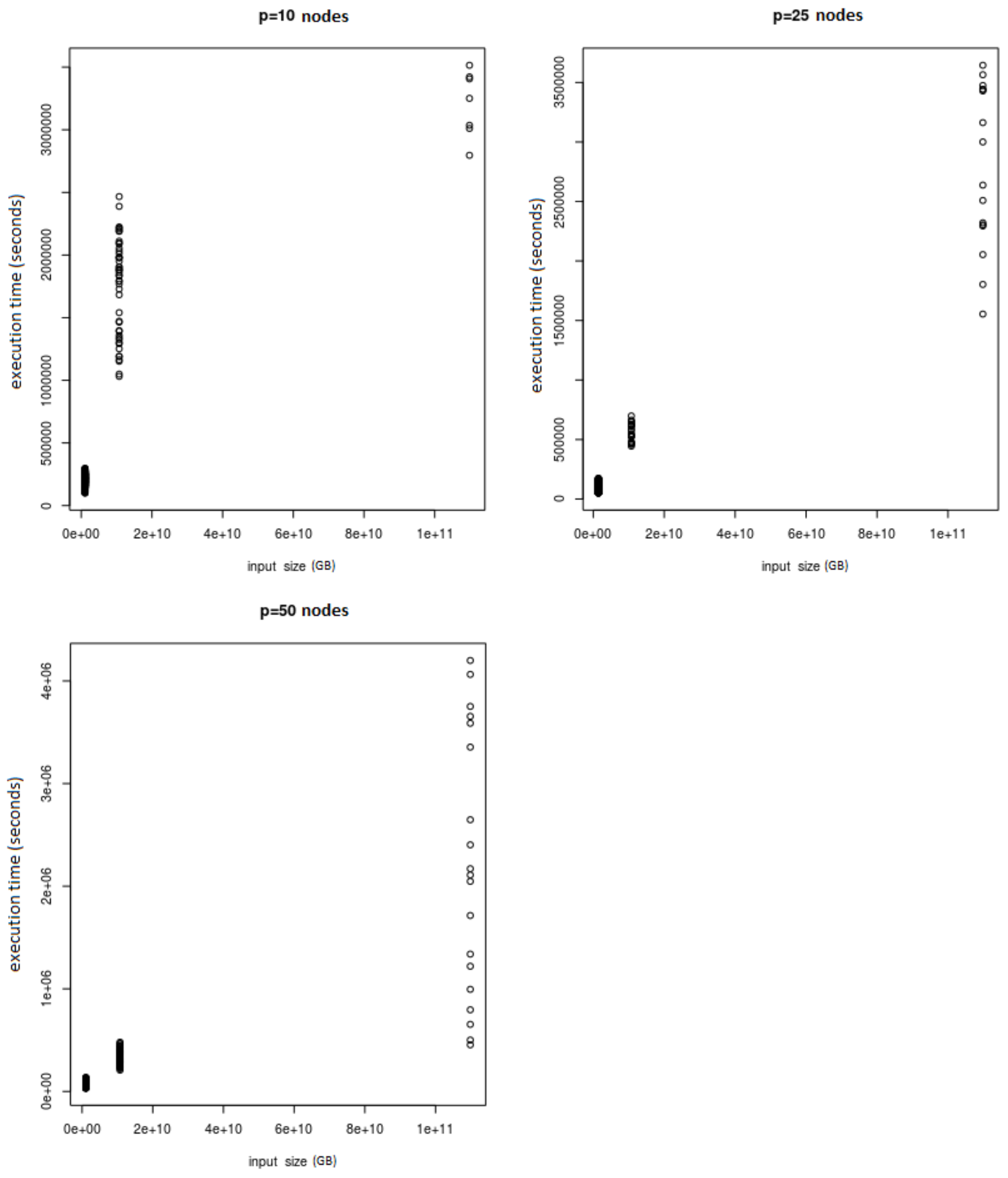


values of R^2 for the Grid'5000 experiments, the results of the tests show a considerable variation between executions, specially with a large workload. What was interesting is that most executions did not end with errors, yet the execution times varied a lot.

One of the most probable reasons for this behavior in a distributed system can be the fact that the infrastructure was shared amongst a large quantity of users, with no network isolation. When the nodes ran the MapReduce applications they were also competing with other cluster nodes running other applications. The main difference with the `gradep` cluster is that the it is, in fact, private, so a shared network is no problem.

Further research is encouraged to assess the influence of network isolation in the variability of the execution time of distributed applications, specifically with MapReduce applications. Further research might obtain a plausible answer for our variable behavior found in Grid'5000.

Figure 5.3: Execution time vs. Workload for the Griffon cluster running the Word-count application



5.2 Experiments on Cloud Infrastructure

As it has been said previously, the main goal for this research work is to obtain a model able to predict the behavior of a MapReduce application in terms of execution time. In this section we will see if the main goal of this work is achieved, and under what conditions.

5.2.1 Amazon Elastic MapReduce

The experiments using the cloud infrastructure of Amazon Elastic MapReduce followed the values shown in table 5.4. Each combination was run 10 times, for a total of 150 executions.

The main restrictions for only running 10 times was the budget and the fact that the account for running the tests in Amazon Web Services belonged to the enterprise that asked the GPPD/INF research group for assistance as it was explained in subsection 4.2.2 and in (CARRERA et al., 2013).

Table 5.4: Parameter values for Amazon Elastic MapReduce environment

Parameter	Value
W	$\{1, 5, 10, 20, 25\}$
p	$\{4, 6, 8\}$
A	log processing
T	`m1.small`

The log processing application, as explained in section 4.2.2 is an example of a sort application, meaning that the relation explained in equation 5.1 serves in this case as well. So, the model to follow with this environment is the one explained in equation 5.3.

The value the cost for Map and Reduce phases was computed using the software *R* (R, 2011) to:

$$C_{U_{MAP}} + k * C_{U_{REDUCE}} = 2.16e^{6}\left[\frac{flop}{Byte}\right] \tag{5.6}$$

With an standard error of $SE = 6.46e^{4}$ and a coefficient of determination of $R^2 = 0.88$.

The formula for predicting the execution time of a MapReduce application in this case would remain as:

$$t_{total} = \frac{W_{MAP}\,[Byte]}{p * T\left[\frac{flop}{s}\right]} * (2.16e^{6}\left[\frac{flop}{Byte}\right]) + \frac{W_{MAP}^{OUT}\,[Byte] * (n_{MAP} mod p)}{n_{MAP} * B\left[\frac{Byte}{s}\right]} \tag{5.7}$$

Following, in figure 5.4 it is shown the experimental data, grouped by the number of nodes used in each experiment and the corresponding line for the mathematical model with two auxiliary lines showing the 95% confidence interval of the model.

After modeling the execution time, giving values to the formula of equation 5.3, more experiments were performed in order to assess the model, with the results described in table 5.5:

Figure 5.4: Execution time vs. Workload for the Amazon EMR Environment

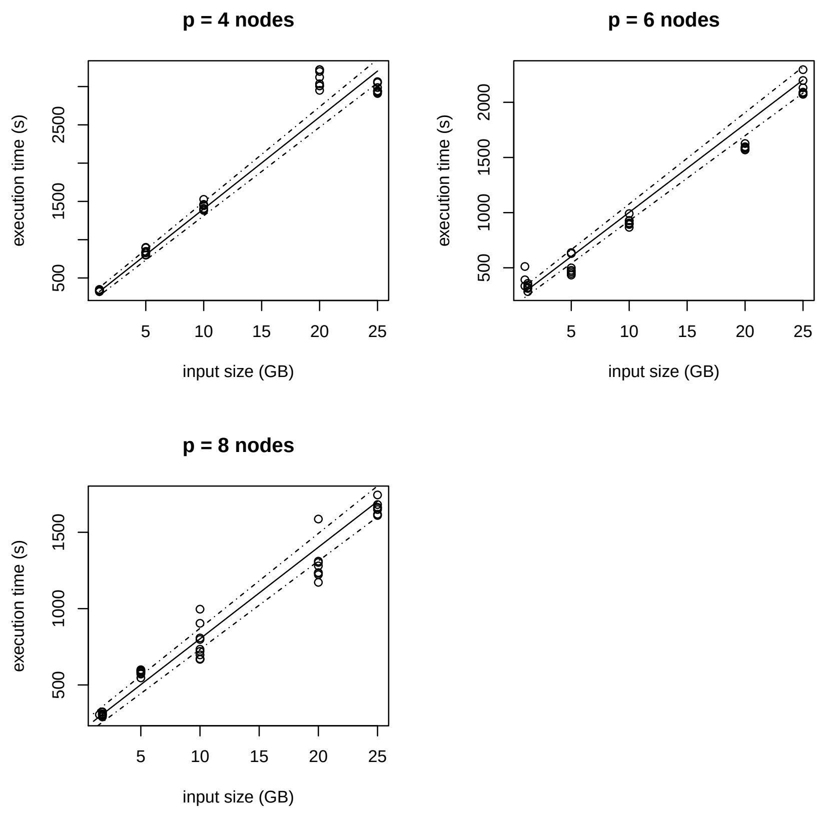


Table 5.5: Results for model assessing of the Amazon measures

Values	Predicted Value	Exp. Value	Error
W=1, p=5	531.01	458.22	13.71%
W=1, p=7	488.23	361.53	25.95%
W=5, p=5	1061.4	870.24	18.01%
W=5, p=7	860.64	780.91	9.26%
W=10, p=5	1724.38	1556.93	9.71%
W=10, p=7	1326.15	1186.78	10.51%
W=20, p=5	3050.35	2826.88	7.33%
W=20, p=7	2257.17	2928.47	29.74%
W=25, p=5	3713.33	3556.38	4.23%
W=25, p=7	2722.68	2414.57	11.32%

Table 5.5 shows the percentage error from comparing the execution times obtained. Errors show a gap between experimental and theoretical values, however, considering that in a Cloud environment the execution times are charged by the hour or fraction, this values should be considered important only when reaching values close to one hour.

5.2.2 Windows Azure HDInsight

Experiments using the Windows Azure HDInsight cloud infrastructure used values from the table 5.6. Each combination was run 10 times, for a total of 150 executions.

Similarly to the experiments in Amazon Elastic MapReduce, the main restrictions for only running 10 times each combination of parameter values were the budget and the fact that we were not the owners of the account for running the tests.

Table 5.6: Parameter values for Windows Azure HDInsight

Parameter	Value
W	$\{1, 5, 10, 20, 25\}$
p	$\{4, 6, 8\}$
A	log processing
T	A3

As in the previous case, the log processing application being an example of a sort application, follows the model explained in equation 5.3. And, since there was no data to assign a value to T for the A3 Azure node type, the relation between the cost for Map and Reduce phases $C_{U_{MAP}} + k * C_{U_{REDUCE}}$ and the computational power T was computed to:

$$\frac{C_{U_{MAP}} + k * C_{U_{REDUCE}}}{T} = 4.14e^{-4}\left[\frac{s}{Byte}\right] \tag{5.8}$$

With an standard error of $SE = 1.06e^{-5}$ and a coefficient of determination of $R^2 = 0.86$.

$$t_{total} = \frac{W_{MAP}\,[Byte]}{p} * (4.14e^{-4}\left[\frac{s}{Byte}\right]) + \frac{W_{MAP}^{OUT}\,[Byte] * (n_{MAP} mod p)}{n_{MAP} * B\left[\frac{Byte}{s}\right]} \tag{5.9}$$

In figure 5.5 it is shown the experimental data, grouped by the number of nodes used in each experiment and the corresponding line for the mathematical model with two auxiliary lines showing the 95% confidence interval of the model.

After the execution time model was obtained, giving values to the formula of equation 5.3, more experiments were run to assess the model and to verify that it was able to predict the performance of a MapReduce application running in the Azure HDInsight cloud environment. Each combination of W and p was run 10 times as well, in order to obtain the average and error values shown in table 5.7.

The values in table 5.7 show the percentage error from comparing the execution times obtained. As in the previous section, since experiments deal with commercial Cloud environments, where execution time is charged by the hour or fraction, this values should be considered big when reaching values close to one hour.

Figure 5.5: Execution time vs. Workload for the Azure HDInsight Environment

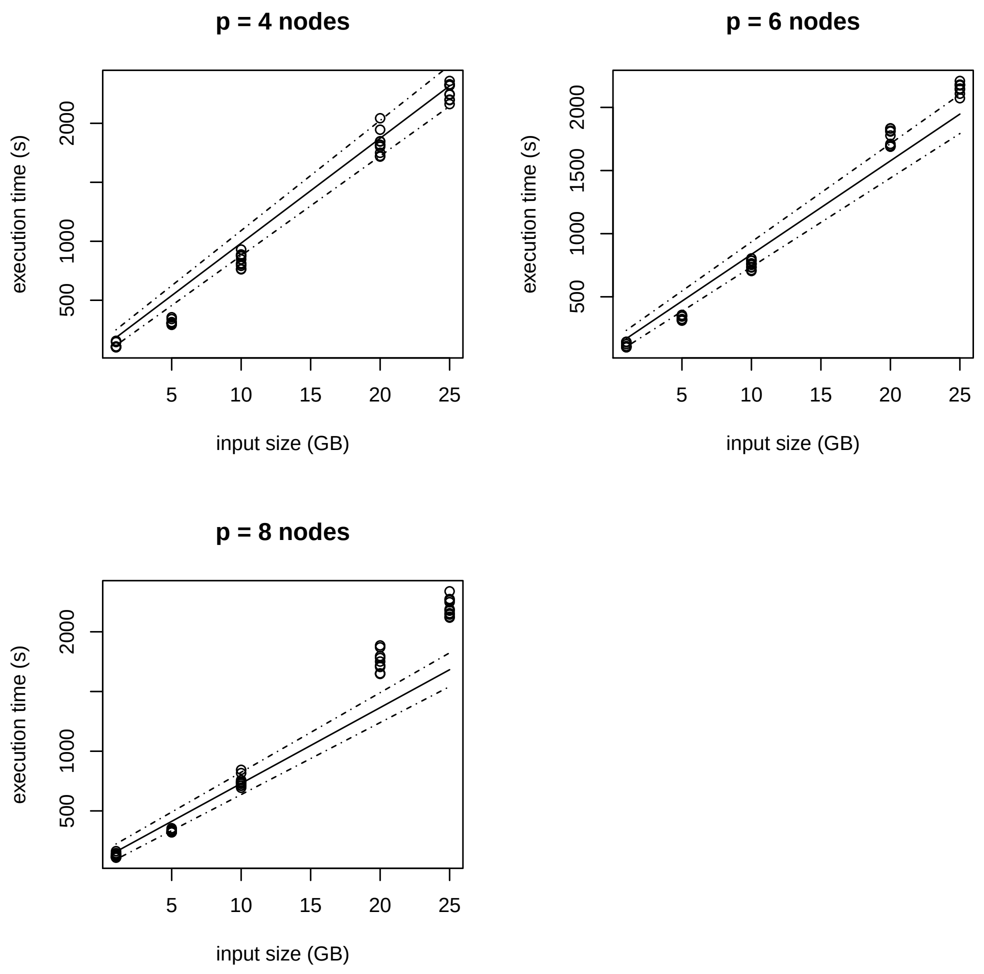


Table 5.7: Results for model assessing the measures on the Azure HDInsight Environment

Values	Predicted Value	Exp. Value	Error
W=1, p=6	401.6	430.7	7.24%
W=1, p=10	373.4	374.8	0.38%
W=1, p=14	339.7	387.3	14.0%
W=5, p=6	899.3	962.1	6.98%
W=5, p=10	752.9	734.7	2.42%
W=5, p=14	687.9	653.1	5.05%
W=10, p=6	1521.4	1442.0	5.22%
W=10, p=10	1227.3	1241.0	1.12%
W=10, p=14	1123.0	1250.3	11.33%
W=20, p=6	2765.6	2526.2	8.66%
W=20, p=10	2176.0	2454.4	12.8%
W=20, p=14	1993.3	2021.5	1.41%
W=25, p=6	3387.7	3392.4	0.14%
W=25, p=10	2560.3	2557.0	3.52%
W=25, p=14	2428.5	2296.9	5.42%

Results for Azure HDInsight, shown in table 5.7 differ from the experimental results from the executions in Amazon Elastic MapReduce, shown in table 5.5. This difference is due to the characteristics of the virtual machines in both Cloud providers, which are different, meaning that with different virtual machine characteristics, the execution time of MapReduce applications would have to be also different. Values for the types of the virtual machines are shown in table 4.1, and addressed in subsections 5.2.1 and 5.2.2, where it can be seen the values T for both Cloud environments.

Finally, as it can be seen from the results of tables 5.5 and 5.7, it is possible to predict the performance of a MapReduce application running in a Cloud Computing environment, thus achieving the main goal of this work.

5.3 Concluding Remarks

In a general way, we can see that we were able to predict the execution time. The error rates from tables 5.2, 5.5 and 5.7 show that the execution model is suitable for the applications, and, since the predictions were done with a generalization of the applications, considering the parameter A as the amount of computing power required to process a unit of storage, the model should work as well for other types of applications.

Also, from the results shown in section 5.1 we can say that a prediction model can be useful in environments where there is some control of the communication layer amongst the cluster nodes. This supposition will have to be proven accurate or inaccurate with more experiments focusing on discovering where the big fluctuation of results come from.

Results in tables 5.5 and 5.7 show a validation of the model, within a certain range of values. It could be said that this range is a constraint for the validity of the model. There could be a divergence when performing experiments with values outside the range used in this experiments, which is the main reason why further work, with different experimental values, is encouraged to be performed by researchers looking to apply the approach described in this research with a broader reach.

In the next chapter, all the conclusions for this work will be summed up.

6 CONCLUSIONS

6.1 About the Results

This master's thesis succeeded in accomplishing its objectives about predicting the performance in terms of execution time of MapReduce applications in the Cloud. The predictions were made using a mathematical model and a methodology to extend this approach to other distributed applications. The mathematical model has the intended form: $t = f(W, p, A, T)$.

The intention for using a general methodology is to be able to extend this approach, as said in (CARRERA; GEYER, 2013), to other types of distributed applications running in a Cloud Computing environment. The idea explained in (CARRERA; GEYER, 2013), where it was said that following a performance evaluation methodology, it should be possible to predict the execution time of a distributed application in a Cloud environment and, conversely, given a time constraint we could be able to know what virtual infrastructure can be enough to execute the distributed application was validated and further research is encouraged in order to test this approach in a more general manner.

Based on the Hadoop Algorithm model described in section 3.2, is was possible to develop a successful mathematical model, expressed in section 4.1, for computing the execution time of MapReduce applications. Similarly, following the procedures for performance modeling and capacity planning described in sections 2.2 and 2.3, it was possible to assess the mathematical model, and to find some practical simplifications to it, as it can be seen in chapter 5.

The formula expressed in equation 4.12 is suitable for modeling the execution time of MapReduce applications. Amongst its strengths, it can be said that it is a general model, since it does not rely its computation on a specific type of application or hardware. It models the execution time with general parameters like the computing power of nodes, expressed as a number of operations per second performed by their processors, and the computing cost of Map and Reduce phases, expressed as the number of operations that the nodes have to process per unit of storage. This model should be able to compute the execution time of a large set of MapReduce applications.

In the mathematical model expressed in equation 4.1, the communication amongst worker nodes is represented by the corresponding $t_{SHUFFLE}$, which is detailed in 5.3 as inversely proportional to the bandwidth B and directly proportional to the number of nodes. In the experiments for this research work, this time was found to be very small compared with the other two values t_{MAP} and t_{REDUCE}, however, this value could not be disregarded in Cloud Computing environments where other users can have an effect over the network usage and thus compromise the prediction of the execution of MapReduce applications.

Findings about the proportionality between the input and the output in the Map phase of `sort` and `wordcount` applications permitted the simplifications of the mathematical model of the execution time of MapReduce applications. The simplifications of the theoretical model in equation 4.12, shown in chapter 5, favor us to see, with more ease, the factors that have a direct impact on the execution time of the MapReduce applications. The linear proportionality of the execution time and the amount of workload reflects an important characteristic of MapReduce applications, which is that the applications mainly depend on the size of the workload, and giving a lesser importance to other factors.

Understanding and modeling MapReduce applications as a relation between the amount of processing operations per unit of storage processed shows a handy way for evaluating the and predicting the performance of this type of applications. This approach is present in other research projects like (KOLBERG et al., 2013) and (KONDO; ANDRZEJAK; ANDERSON, 2008). This approach of thinking of MapReduce applications as operations with units of storage also permitted to model the type of working nodes in Cloud environments considering mostly their processors.

For Cloud users, the capability to predict the execution time of applications running in a Cloud model like *Software as a Service* or *Infrastructure as a Service* is important when estimating the costs that could involve to deploy an infrastructure for an application. This capability could be an interesting feature for Cloud services, and could help users plan their developments.

6.2 Further Work

As it was stated in the motivation of this work, further work is encouraged to test the approach described in this work with different distributed applications. Testing with different MapReduce applications should allow to strengthen the mathematical model, and better understand the behavior of the applications.

From the point of view of a Cloud provider, being able to tell the time that a distributed application will take to run *'a-priori'* is useful in business models like the Spot Instances of Amazon (EC2, 2013). In said business model, providers offer virtual machines with lower prices, based on the fact that they were reserved for other users for a longer time than the one they were actually busy. So, the provider can re-lease the virtual machines in lower prices. If a provider is able to know when a virtual machine will be freed, it can predict when it can be used as a Spot Instance.

With the models exposed in chapter 5, a Cloud provider can use the approach explained in this work to program a feature of its Cloud platform where a Cloud user can know how much time a given MapReduce application will take to run, letting him program his budget.

As it was shown in the results in subsection 5.1.2, when running the experiments in the Grid'5000 environment we were not able to carry out the performance model of the MapReduce applications. There was a considerable variation amongst the execution time measures, which we believe that was caused because of the variability of load over the shared network through the execution of experiments. This considerable variation was not found when running in Cloud Computing environments. With further experimentation we should be able to identify the actual cause of this difference.

REFERENCES

ABAD, C. L.; ROBERTS, N.; LU, Y.; CAMPBELL, R. H. A storage-centric analysis of MapReduce workloads: file popularity, temporal locality and arrival patterns. In: WORKLOAD CHARACTERIZATION (IISWC), 2012 IEEE INTERNATIONAL SYMPOSIUM ON, 2012. **Proceedings...** [S.l.: s.n.]. p.100–109, 2012.

ALMEIDA, V. A.; MENASCÉ, D. A. Capacity planning an essential tool for managing Web services. **IT professional**, [S.l.], v.4, n.4, p.33–38, 2002.

ANDRZEJAK, A.; KONDO, D.; ANDERSON, D. P. Exploiting non-dedicated resources for cloud computing. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2010 IEEE, 2010. **Proceedings...** [S.l.: s.n.]. p.341–348, 2010.

ANDRZEJAK, A.; KONDO, D.; YI, S. Decision model for cloud computing under sla constraints. In: MODELING, ANALYSIS & SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), 2010 IEEE INTERNATIONAL SYMPOSIUM ON, 2010. **Proceedings...** [S.l.: s.n.]. p.257–266, 2010.

ANJOS, J.; KOLBERG, W.; GEYER, C. R.; ARANTES, L. B. Addressing Data-Intensive Computing Problems with the Use of MapReduce on Heterogeneous Environments as Desktop Grid on Slow Links. In: COMPUTER SYSTEMS (WSCAD-SSC), 2012 13TH SYMPOSIUM ON, 2012. **Proceedings...** [S.l.: s.n.]. p.148–155, 2012.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. et al. A view of cloud computing. **Communications of the ACM**, [S.l.], v.53, n.4, p.50–58, 2010.

BABU, S. Towards automatic optimization of MapReduce programs. In: ACM SYMPOSIUM ON CLOUD COMPUTING, 1., 2010. **Proceedings...** [S.l.: s.n.]. p.137–142, 2010.

BOUTABA, R.; CHENG, L.; ZHANG, Q. On Cloud computational models and the heterogeneity challenge. **Journal of Internet Services and Applications**, [S.l.], v.3, n.1, p.77–86, 2012.

CARRERA, I.; GEYER, C. Impressionism in Cloud Computing. A Position Paper on Capacity Planning in Cloud Computing environments. In: INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS (ICEIS), 15., 2013. **Proceedings...** [S.l.: s.n.]. p.333–338, 2013.

CARRERA, I.; SCARIOT, F.; TURIN, P.; GEYER, C. An Example for Performance Prediction for Map Reduce Applications in Cloud Environments. In: ESCOLA REGIONAL DE REDES DE COMPUTADORES ERRC - RS RIO GRANDE DO SUL, 2013. **Proceedings...** [S.l.: s.n.]. 2013.

CHEN, Y.; GANAPATHI, A.; GRIFFITH, R.; KATZ, R. The case for evaluating MapReduce performance using workload suites. In: MODELING, ANALYSIS & SIMULATION OF COMPUTER AND TELECOMMUNICATION SYSTEMS (MASCOTS), 2011 IEEE 19TH INTERNATIONAL SYMPOSIUM ON, 2011. **Proceedings...** [S.l.: s.n.]. p.390–399, 2011.

CHEN, Y.; GANAPATHI, A. S.; GRIFFITH, R.; KATZ, R. H. Analysis and lessons from a publicly available google cluster trace. **EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95**, [S.l.], 2010.

CHEN, Y.; GANAPATHI, A. S.; GRIFFITH, R.; KATZ, R. H. Towards understanding cloud performance tradeoffs using statistical workload analysis and replay. **University of California at Berkeley, Technical Report No. UCB/EECS-2010-81**, [S.l.], 2010.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, [S.l.], v.51, n.1, p.107–113, 2008.

DEELMAN, E.; SINGH, G.; LIVNY, M.; BERRIMAN, B.; GOOD, J. The cost of doing science on the cloud: the montage example. In: ACM/IEEE CONFERENCE ON SUPERCOMPUTING, 2008., 2008. **Proceedings...** [S.l.: s.n.]. p.50, 2008.

EC2. Amazon Web Services - EC2 Elastic Compute Cloud `http://aws.amazon.com/ec2` acessed on 07/23/2013.

EMR. Amazon Web Services - EMR Elastic MapReduce `http://aws.amazon.com/elasticmapreduce` acessed on 07/23/2013.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud computing and grid computing 360-degree compared. In: GRID COMPUTING ENVIRONMENTS WORKSHOP, 2008. GCE'08, 2008. **Proceedings...** [S.l.: s.n.]. p.1–10, 2008.

GRID5000. Grid 5000 Project `https://www.grid5000.fr/` acessed on 07/23/2013.

HADOOP. Apache Hadoop `https://www.grid5000.fr/` acessed on 12/28/2013.

HARBAOUI, A.; SALMI, N.; DILLENSEGER, B.; VINCENT, J.-M. et al. Automatic performance modelling of black boxes targetting self-sizing. **INRIA Tech. Rep. 7027**, [S.l.], 2009.

HDINSIGHT. Windows Azure HDInsight `http://azure.microsoft.com/en-us/documentation/services/hdinsight/` acessed on 12/02/2014.

HERODOTOU, H. Hadoop performance models. Technical Report CS-2011-05. **Duke Computer Science**, [S.l.], 2011.

HERODOTOU, H.; BABU, S. Profiling, what-if analysis, and cost-based optimization of MapReduce programs. In: VLDB ENDOWMENT, 2011. **Proceedings...** [S.l.: s.n.]. v.4, n.11, p.1111–1122, 2011.

HERODOTOU, H.; DONG, F.; BABU, S. No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics. In: ACM SYMPOSIUM ON CLOUD COMPUTING, 2., 2011. **Proceedings...** [S.l.: s.n.]. p.18, 2011.

HERODOTOU, H.; LIM, H.; LUO, G.; BORISOV, N.; DONG, L.; CETIN, F. B.; BABU, S. Starfish: a self-tuning system for big data analytics. In: FIFTH CIDR CONF, 2011. **Proceedings...** [S.l.: s.n.]. 2011.

IBRAHIM, S.; JIN, H.; LU, L.; QI, L.; WU, S.; SHI, X. Evaluating mapreduce on virtual machines: the hadoop case. In: **Cloud Computing**. [S.l.]: Springer, 2009. p.519–528, 2009.

IOSUP, A.; OSTERMANN, S.; YIGITBASI, M. N.; PRODAN, R.; FAHRINGER, T.; EPEMA, D. H. Performance analysis of cloud computing services for many-tasks scientific computing. **Parallel and Distributed Systems, IEEE Transactions on**, [S.l.], v.22, n.6, p.931–945, 2011.

IOSUP, A.; YIGITBASI, N.; EPEMA, D. On the performance variability of production cloud services. In: CLUSTER, CLOUD AND GRID COMPUTING (CCGRID), 2011 11TH IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2011. **Proceedings...** [S.l.: s.n.]. p.104–113, 2011.

JAIN, R. **The art of computer systems performance analysis**. [S.l.]: John Wiley & Sons Chichester, 1991. v.182.

JIANG, D.; OOI, B. C.; SHI, L.; WU, S. The performance of mapreduce: an in-depth study. In: VLDB ENDOWMENT, 2010. **Proceedings...** [S.l.: s.n.]. v.3, n.1-2, p.472–483, 2010.

KARLOFF, H.; SURI, S.; VASSILVITSKII, S. A model of computation for MapReduce. In: TWENTY-FIRST ANNUAL ACM-SIAM SYMPOSIUM ON DISCRETE ALGORITHMS, 2010. **Proceedings...** [S.l.: s.n.]. p.938–948, 2010.

KOLBERG, W.; B. MARCOS, P. de; ANJOS, J. C.; MIYAZAKI, A. K.; GEYER, C. R.; ARANTES, L. B. {MRSG} – A MapReduce simulator over SimGrid. **Parallel Computing**, [S.l.], v.39, n.4–5, p.233 – 244, 2013.

KONDO, D.; ANDRZEJAK, A.; ANDERSON, D. P. On correlated availability in internet-distributed systems. In: IEEE/ACM INTERNATIONAL CONFERENCE ON GRID COMPUTING, 2008., 2008. **Proceedings...** [S.l.: s.n.]. p.276–283, 2008.

MELL, P.; GRANCE, T. The NIST definition of cloud computing (draft). **NIST special publication**, [S.l.], v.800, p.145, 2011.

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

MIETZNER, R.; LEYMANN, F. Towards provisioning the cloud: on the usage of multi-granularity flows and services to realize a unified provisioning infrastructure for saas applications. In: SERVICES-PART I, 2008. IEEE CONGRESS ON, 2008. **Proceedings...** [S.l.: s.n.]. p.3–10, 2008.

MISHRA, A. K.; HELLERSTEIN, J. L.; CIRNE, W.; DAS, C. R. Towards characterizing cloud backend workloads: insights from google compute clusters. **ACM SIGMETRICS Performance Evaluation Review**, [S.l.], v.37, n.4, p.34–41, 2010.

O'MALLEY, O. Terabyte sort on apache hadoop. **Yahoo, available online at: http://sortbenchmark. org/Yahoo-Hadoop. pdf**, [S.l.], p.1–3, 2008.

R. **R**: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0 `http://www.R-project.org/`.

SANGROYA, A.; SERRANO, D.; BOUCHENAK, S. Benchmarking Dependability of MapReduce Systems. In: RELIABLE DISTRIBUTED SYSTEMS (SRDS), 2012 IEEE 31ST SYMPOSIUM ON, 2012. **Proceedings...** [S.l.: s.n.]. p.21–30, 2012.

SHI, J. Y.; TAIFI, M.; KHREISHAH, A. Resource planning for parallel processing in the cloud. In: HIGH PERFORMANCE COMPUTING AND COMMUNICATIONS (HPCC), 2011 IEEE 13TH INTERNATIONAL CONFERENCE ON, 2011. **Proceedings...** [S.l.: s.n.]. p.828–833, 2011.

SHVACHKO, K.; KUANG, H.; RADIA, S.; CHANSLER, R. The hadoop distributed file system. In: MASS STORAGE SYSTEMS AND TECHNOLOGIES (MSST), 2010 IEEE 26TH SYMPOSIUM ON, 2010. **Proceedings...** [S.l.: s.n.]. p.1–10, 2010.

TIAN, F.; CHEN, K. Towards optimal resource provisioning for running mapreduce programs in public clouds. In: CLOUD COMPUTING (CLOUD), 2011 IEEE INTERNATIONAL CONFERENCE ON, 2011. **Proceedings...** [S.l.: s.n.]. p.155–162, 2011.

VERMA, A.; CHERKASOVA, L.; CAMPBELL, R. H. ARIA: automatic resource inference and allocation for mapreduce environments. In: ACM INTERNATIONAL CONFERENCE ON AUTONOMIC COMPUTING, 8., 2011. **Proceedings...** [S.l.: s.n.]. p.235–244, 2011.

VIANNA, E.; COMARELA, G.; PONTES, T.; ALMEIDA, J.; ALMEIDA, V.; WILKINSON, K.; KUNO, H.; DAYAL, U. Analytical performance models for MapReduce workloads. **International Journal of Parallel Programming**, [S.l.], v.41, n.4, p.495–525, 2013.

WANG, P.; HUANG, W.; VARELA, C. A. Impact of virtual machine granularity on cloud computing workloads performance. In: GRID COMPUTING (GRID), 2010 11TH IEEE/ACM INTERNATIONAL CONFERENCE ON, 2010. **Proceedings...** [S.l.: s.n.]. p.393–400, 2010.

WHITE, T. **Hadoop**: the definitive guide. [S.l.]: O'Reilly, 2012.

YELICK, K.; COGHLAN, S.; DRANEY, B.; CANON, R. S. et al. The Magellan report on cloud computing for science. **US Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR) December**, [S.l.], 2011.

ZHANG, Z.; CHERKASOVA, L.; LOO, B. T. Performance modeling of mapreduce jobs in heterogeneous cloud environments. In: IEEE SIXTH INTERNATIONAL CONFERENCE ON CLOUD COMPUTING, 2013. **Proceedings...** [S.l.: s.n.]. p.839–846, 2013.