ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph. D.

BY
Luis Eduardo BAUTISTA VILLALPANDO

A PERFORMANCE MEASUREMENT MODEL FOR CLOUD COMPUTING APPLICATIONS

MONTREAL, JULY 2th, 2014

BOARD OF EXAMINERS (THESIS PH.D.)

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain April, Thesis Supervisor
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

Mr. Alain Abran, Thesis Co-supervisor
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

Mr. Daniel Forgues, President of the Board of Examiners
Département de génie de la construction à l'École de technologie supérieure

Mr. Abdelouahed Gherbi, Member of the jury
Département de génie logiciel et des technologies de l'information à l'École de technologie supérieure

Mrs. Cherifa Mansoura Liamani
Senior consultant à la Banque Nationale du Canada

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

JUNE 20, 2014

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENT

I would like to express my deepest gratitude to my advisor, Dr. Alain April, for his guidance, motivation, patience, and providing me with an excellent atmosphere for doing research as well as financially supporting my publications.

I would like to thank Dr. Alain Abran who let me experienced the research of measurement in software engineering by means of practical issues beyond the textbooks, patiently correcting my writing and also financially supporting my publications.

I would like to thank my wife, Erika Cruz. She was always there cheering me up and she stood by me through the good and challenging times.

I would also like to thank my mother, two brothers, and elder sister. They were always supporting me and encouraging me with their best wishes.

Last but not the least; I would like to thank my father Luis B. Bautista for supporting me spiritually throughout my life. I am sure he would be very proud and happy because of the work done.

A PERFORMANCE MEASUREMENT MODEL FOR CLOUD COMPUTING APPLICATIONS

Luis Eduardo BAUTISTA VILLALPANDO

RÉSUMÉ

L'informatique en nuage est une technologie émergente permettant l'accès, sur demande, à un ensemble partagé de ressources informatiques. L'un des principaux objectifs de cette nouvelle technologie est le traitement et le stockage de très grandes quantités de données. Parfois, des anomalies et des défauts peuvent surgir et ainsi réduire sa performance. Étant donné la nouveauté et la complexité de cette nouvelle technologie il n'existe pas de modèles de mesures visant à évaluer la dégradation de la performance d'une application opérant dans le nuage. Un des défis actuel est de concevoir un modèle de mesure de la performance des applications opérant sur le nuage permettant de prendre en compte les nombreuses caractéristiques qualité d'un logiciel telles que précisées dans la norme ISO25010. De manière pratique, un modèle de mesure permettrait de mesurer, par exemple la performance d'une application, et plus particulièrement aider à détecter la source de la dégradation de performance afin de prendre des actions correctives. Grace à ce modèle il serait alors possible de planifier les ressources nécessaires afin de rencontrer des niveaux de services ciblés. Cette thèse présente la proposition d'une modèle de mesure de la performance d'une application opérant sur le nuage. Un des défis majeurs rencontrés dans la définition de ce modèle a été de déterminer quels types de relations existent entre les différentes mesures directes du logiciel qui reflètent le mieux le concept de performance pour cette nouvelle technologie. Par exemple, nous avons étudié l'étendue de la relation entre le temps d'exécution d'un logiciel sur le processeur et le concept de performance du comportement de temps d'ISO25010. Cette thèse propose donc un modèle de mesure de la performance pour les applications opérant sur le nuage, qui est fondé sur les concepts de la qualité des logiciels proposés par la norme ISO 25010.

Mots-clés: Informatique nuagique, génie logiciel, mesure, performance, technique de Taguchi, ISO 25010, maintenance, Hadoop MapReduce.

A PERFORMANCE MEASUREMENT MODEL FOR CLOUD COMPUTING APPLICATIONS

Luis Eduardo BAUTISTA VILLALPANDO

ABSTRACT

Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. One of the main objectives of this technology is processing and storing very large amounts of data by means of Cloud Computing Applications. Sometimes, anomalies and defects found in the Cloud platforms affect the performance of these applications resulting in degradation of the Cloud performance. These anomalies can be identified by performance concepts of Cloud Computing based on software engineering quality models. One of the challenges in Cloud Computing is how to analyze the performance of Cloud Computing Applications in order to determine the main factors which affect the quality of the Cloud. Performance measurement results are very important because they help to detect the source of the degradation of the Cloud and, as a consequence, improve its performance. Furthermore, such results can be used in future resource planning stages or for the design of Service Level Agreements. This thesis presents Cloud Computing Application concepts that are directly related to the measurement of performance from a quantitative viewpoint. One of the challenges in defining such concepts is how to determine what type of relationships exist between the various performance base measures that define the performance concepts in a Cloud environment. For example, what is the extent of the relationship between CPU processing time and performance concepts such as time behavior? In addition, this thesis proposes a performance measurement model for Cloud Computing Applications, which integrates software quality concepts from ISO 25010 and makes use of the Taguchi´s method for the design of experiments in order to present an example of how to apply the model to a practical case.

Keywords: Cloud Computing, Software engineering, Analysis, Measurement, Performance, Taguchi method, ISO 25010, Maintenance, Hadoop MapReduce.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

ASP	Application service provider
CBSS	Computer-based software system
CC	Cloud Computing
CCC	Cloud Computing cluster
CCA	Cloud Computing architecture
CCF	Common-cause failures
CCS	Cloud computing system
CPU	Central processing unit
DN	Data node
DRAM	Dynamic random access memory
DS	Distributed system
FSM	File system metadata
FSN	File system namespace
GFS	Google file system
HDFS	Hadoop distributed file system
HPA	Hadoop performance analysis
HPC	High performance computing
IaaS	Infrastructure as a service
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IT	Information technology

MPP	Massively parallel processing
NERSC	National Energy Research Scientific Computing Center
NDFS	Nutch distributed file system
NN	Name node
OA	Orthogonal array
OS	Operating system
PaaS	Platform as a service
PMFCC	Performance measurement framework for cloud computing
PMMo	Performance measurement model
PMMoCCA	Performance measurement model for cloud computing application
QEST	Quality Factor + Economic, Social and Technical dimensions
RPC	Remote-procedure call
SaaS	Software as a service
SC	Service component
SPQM-RM	Software product quality measurement reference model
SQuaRE	Systems and software product quality requirements and evaluation
SQL	Structured query language
SLA	Service level agreement
SNR	Signal-to-noise ratio
SSP	Sustained system performance
SUT	System under test
SWEBOK	Software Engineering Body of Knowledge

TA	Test application
TMP	Technical and management process
VIM	International vocabulary of metrology
VM	Virtual machine

INTRODUCTION

Cloud Computing (CC) is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Some CC users prefer not to own physical infrastructure, but instead rent a Cloud infrastructure, or a Cloud platform or software, from a third-party provider. These infrastructure application options delivered as a service are known as Cloud Services (Jin, Ibrahim *et al.* 2010). One of the most important challenges in delivering Cloud Services is to ensure that they are fault tolerant, since as failures and anomalies can degrade these services and impact their quality, and even their availability. According to Coulouris (Coulouris, Dollimore *et al.* 2011), a failure occurs in a distributed system (DS), like a CC system (CCS), when a process or a communication channel departs from what is considered to be its normal or desired behavior. An anomaly is different, in that it slows down a part of a CCS without making it fail completely, impacting the performance of tasks within nodes, and, consequently, of the system itself.

A performance measurement model (PMMo) for CCS, and more specifically for Cloud Computing Applications (CCA), should propose a means to identify and quantify "normal application behavior," which can serve as a baseline for detecting and predicting possible anomalies in the software (i.e. jobs in a Cloud environment) that may impact Cloud application performance. To achieve this goal, methods are needed to collect the necessary base measures specific to CCA performance, and analysis models must be designed to analyze and evaluate the relationships that exist among these measures. This thesis presents the Performance Measurement Model for Cloud Computing Applications (PMMoCCA) which proposes a mean to analyze the performance of Cloud Computing Applications running in Hadoop environments which process and analyze very large amounts of data.

CHAPTER 1

PRESENTATION OF RESEARCH

1.1 Motivation

Cloud Computing (CC) is an emerging technology aimed at processing and storing very large amounts of data. According to the ISO SC38 Study Group on Cloud Computing (ISO/IEC 2012), CC is a paradigm for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable cloud resources accessed through services, that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The ISO SC38 Study Group mentions that Cloud Services are categorized in service models as:

- Infrastructure as a Service (IaaS),
- Platform as a Service (PaaS),
- Software as a Service (SaaS), and
- Network as a Service (NaaS).

These service models include all the technical resources that clouds have in order to process information, like software, hardware, and network elements. For example, the IaaS model is related to hardware architectures and virtualization while the service model that relates most to the software engineering community is the SaaS model. Software engineers focus on software components, and customers use an IT provider's applications running on a Cloud infrastructure to process information according to their processing and storage requirements. One of the main characteristics of SaaS model is that customers do not manage or control the underlying Cloud infrastructure (including network, servers, operating systems, and storage), except for limited user-specific application configuration settings.

Performance measurement models (PMMo) for Cloud Computing Applications (CCA) should propose a means to identify and quantify "normal application behavior" into Cloud

Computing Systems (CCS). One of the main motivations for the creation of PMMo for CCA is the lack of information which helps to understand and define concepts of assurances of availability, reliability and liability in CCA. Concepts such as price, performance, time to completion (availability), likelihood of completion (probability of failure) and penalty (liability) are key to being able to produce a comparison of services, in order to establish Service Level Agreements (SLA) or to improve the performance of CCA.

According to Li (Li, Gillam *et al.* 2010), commercial CCS enable to capture price–performance information relating to specific applications with relatively well-known demands on systems, and to determine how such a comparison service may be formulated. Such a comparison service will necessarily depend on both the performance requirements of the user and the current availability of the system, as well as the price the consumer is willing to pay. Moreover, Gangadharan (Gangadharan and Parrilli 2011) states that the pricing of Cloud Computing services is associated with differentiated levels of service with varying capacity of memory, computing units, and platforms. The pricing also varies with respect to operating systems and geographical locations. The criteria for pricing of platform of Cloud services can be based on the hour, CPU cycle, or otherwise. In addition, Gangadharan mentions that pricing of infrastructural Cloud services depends upon levels of use, layers of service, or hybrids of these options.

Thus, a PMMo for CCA is an important issue for maintainers, users and developers to help to populate SLA as well as to improve CCA performance decreasing the number of failures and anomalies that could affect the system operation and consequently their applications.

1.2 Problem definition

One of the most important challenges in delivering Cloud Services is to ensure that they are tolerant to failures and anomalies which can degrade these services and impact their quality, and even their availability. According to Coulouris (Coulouris, Dollimore *et al.* 2011), a failure occurs in a distributed system (DS), like a CCA, when a process or a communication

channel departs from what is considered to be its normal or desired behavior. An anomaly is different, in that it slows down a part of a CCA without making it fail completely, impacting the performance of tasks within nodes, and, consequently, of the system itself.

Furthermore, CCA, are exposed to common-cause failures (CCF) which are a direct result of a common cause (CC) or a shared root cause, such as extreme environmental conditions, or operational or maintenance errors (Xing and Shrestha 2005). Some examples of CCF in CCA are:

1. **Memory failures.** According to Schroeder (Schroeder, Pinheiro *et al.* 2009), memory failure is one of the main CCF, and errors in dynamic random access memory (DRAM) are a common form of hardware failure in modern computer clusters. He defines a memory error as an event that leads to the logical state of one or more bits being read differently from how they were last written. Schroeder's study included the majority of machines in Google's fleet, and spanned nearly 2.5 years (from January 2006 to June 2008) and six different hardware platforms, where a platform was defined by the motherboard and by memory generation. Schroeder's research shows that close to a third of all the machines in the fleet had had at least one memory error per year.
2. **Storage failures.** Another type of failure commonly present in clusters of computers is the storing failure. According to Bairavasundaram (Bairavasundaram, Goodson *et al.* 2008), a primary cause of data loss is disk drive unreliability. This is because hard drives are mechanical, moving devices that can suffer from mechanical problems leading to drive failures, and hence data loss. Bairavasundaram shows that the most common technique used in storage systems to detect data corruption is to add a higher-level checksum for each disk block to validate each disk block read. A checksum, or hash sum, is fixed-size data computed from an arbitrary block of digital data for the purpose of detecting accidental errors that may have been introduced during transmission or storage. This enables the integrity of the data to be checked at a later time by recomputing

the checksum and comparing it with the stored one. If the checksums match, the data were almost certainly not altered (either intentionally or unintentionally). Bairavasundaram maintains that, on average, every hard disk presents 104 checksum mismatches. Although this is not a large number of errors, it does mean that in critical computers, such as servers, this may result in a critical anomaly.

3. **Processes failures.** Processes failures or applications failures are common in CCA and whether a CCA cluster has from 50 to 100 users running tasks, it makes it difficult to detect the cause of anomalies (Dhruba and Ryan 2010). Dhruba mentions that it is due to the fact that each user can see the performance of one particular task on a specific machine which is taking a long time to be processed: bad ad hoc jobs consume much memory and can create "hung machines" that could impact the periodic pipeline jobs as well as cluster performance.

PMMo for CCA should propose a means to identify and quantify "normal applications behavior," which can serve as a baseline for detecting possible anomalies in the computers (i.e. nodes in a cluster) that may impact cloud application performance. To achieve this goal, methods are needed to collect the necessary base measures specific to CCA performance, and analysis models must be designed to determine the relationships that exist among these measures. The ISO International Vocabulary of Metrology (VIM) (ISO/IEC 2008) defines a measurement method as a generic description of a logical organization of operations used in measurement, and the ISO 15939 standard (ISO/IEC 2008) defines an analysis model as an algorithm or calculation combining one or more measures obtained from a measurement method to produce evaluations or estimates relevant to the information needed for decision making.

1.3 Research question

Research goals of this work are focused on how to develop a Performance Measurement Model for Cloud Computing Applications (PMMoCCA) which defines performance

concepts and their relationship which will help to design future analysis models to detect failures and identify anomalies. More specifically, the objectives of this research are:

- Identify the measures of hardware and software that are related to performance in Cloud Computing Applications.
- Propose a detailed inventory of performance measurement activities and processes in order to carry out a performance measurement of Cloud Computing Applications
- Design a Performance Measurement Framework to identify the concepts involved in the performance measurement of Cloud Computing Application.
- Propose a performance measurement model for Cloud Computing Applications.

The research objectives must also address the following research question:

- How can the performance of Cloud Computing Applications be improved?

and, more specifically:

- What is the measurement process to analyze the performance of CCA?
- Which CCS characteristics are more related with the performance of CCA?
- Is there an existing method able to measure the above characteristics from the perspective of maintainers, developers and users?
- How can the PMMoCCA be used in practice to analyze the performance in order to improve CCA in an organization?

1.4 Methodology

The methodology used to conduct this software engineering research and to attempt to answer the research questions is based on an adapted version of the Basili´s framework (Abran, Laframboise and Bourque, 1999). This research methodology is composed of four

phases: definition, planning, development and interpretation of results. Each of these phases as well as their activities is described next.

1.4.1 Definition

The definition phase consists of identifying the research problem, and possible solutions are explored. Table 1.1 shows the elements of the definition phase.

Table 1.1 Elements of the research definition phase

Motivation	Objective	Proposal	Research Users
Explore how to measure applications performance in the context of Cloud Computing.	• Propose a performance measurement model for Cloud Computing Applications (PMMoCCA) which could identify the main factors that affect the performance of CCA. • Design a measurement method which helps to quantify quality characteristics of a CCS that are related to performance.	Design a performance measurement model for Cloud Computing Applications and an experiment which provides relevant information about the CCA operation to detect possible anomalies.	Students, researchers and industry practitioners of information technology

This phase of the research methodology has the objective to establish the research context and propose the activities to develop it.

1.4.2 Planning

The planning phase identifies research activities and the deliverables to attempt to reach our objective and answer the research questions. In addition, this phase includes the literature review necessary to conduct the research. Table 1.2 describes this phase and presents the inputs and outputs of each research activity.

Table 1.2 Stages of the planning phase

Project Stage	Inputs	Outputs
Stage 1 Literature review	Literature review on : • Distributed System concepts their definition, goals and architectures; • Definition of Distributed System technologies such as CCS; • The main differences between Grid Computing and Cloud Computing; • Cloud Computing concepts such as definition, goals, architectures and infrastructure services such as virtualization; • Hadoop technologies and their relationship with performance aspects and fault tolerance in Cloud Computing.	• Publications, technical reports and identification of working groups in Cloud Computing which are related with the topic of fault tolerance and performance analysis • Set up of a Cloud Computing cluster with the Hadoop Distributed File System (HDFS) to configure Hadoop.
Stage 2 Definition of research problem	• Literature review of topics related to aspects of quality models in software engineering and performance measurement processes: • 25010 software quality model; • ISO 25020 Measurement Reference model • ISO 15939 Measurement process • ISO 14756 Measurement and Rating of Performance of computer-based software systems (CBSS)	• Identification of research topic and unresolved issues; • Development of the research methodology; • Updated literature review report
Stage 3 Research validation	• Reading list (books, articles, etc.) selected by jury members to prepare for the evaluation; • Jury questions on CCA Performance models	Evaluation of the research proposal
Stage 4 Oral debate of the research proposal	Improved literature review, objective, proposal and research questions.	• Oral debate with the PhD jury focused on the research goals, originality, feasibility

		and schedule.
Stage 5 Research activities	• Identify the main characteristics in CCA related to performance concepts; • Design of a first version of a Performance Measurement Framework for CC (PMFCC)	• Proposed model to measure CCA performance characteristics; • Define the relationships between performance characteristics of CCA; • Submission of articles for publication.
Stage 6 Revision and submission of the doctoral thesis	• Develop a case study; • Choose a validation strategy; • Prepare and execute the case study.	Final results • A performance measurement model for CCA • A sub-project framework to be proposed to the current Hadoop project to help to measure the CCA performance • Proposal of a performance analysis method to analyze CCA performance.

1.4.3 Development

The development phase sets up the components that design a solution to the main research question. Table 1.3 presents the elements as well as their validations and analysis.

Table 1.3 Elements of the development phase

Development	Validation	Analysis
Develop the Performance Measurement Framework for Cloud Computing (PMFCC)	Publish the proposed framework in a software engineering journal.	Verify comments by editors, practitioners and users interested in implementing the proposed framework to improve it.
Develop the Performance measurement model for Cloud Computing Applications (PMMoCCA)	Define the performance measures of CCA and CCS in order to map them onto the performance concepts defined in PMFCC; Determine the degree of	Test the hypothesis with an experiment at the laboratory

	relationships between the above performance measures in order to analyze the performance of CCA.	
Redefine the performance model proposal	Define an experimentation methodology to determine the relationships between performance measures; Prepare a case study and execute the experiment	Analyze the experiment results in order to obtain conclusions and improve the PMMoCCA.

1.4.4 Interpretation

The interpretation phase consists in reviewing the research problem and analyzing the proposed solution to obtain conclusions, assess the proposed solution for the industry, and finally identify the future work. The table 1.4 presents the interpretation phase components as well as their explication.

Table 1.4 Elements of interpretation phase

Results	Extrapolation	Future Works
• The research addresses the problem of measurement of performance of CCA. • Measurement performance concepts for CCA are clearly identified as part of this thesis • Implementation of the performance measurement process and analysis methodology is used to represent the performance of CCA • The experiment presented in this thesis shows that is feasible the use of the PMMoCCA to analyze and represent the performance of CCA	• The results based on an experiment with performance measures extracted from CCA and CCS were used to verify the PMMoCCA.	• Extension of the PMMoCCA to advanced clouds such elastic clouds which change their size according to user needs combining different development frameworks. • Design a repository of performance measures to provide information to facilitate the design, validation, and comparison of new performance analysis models and algorithms for CCS and CCA. • Design new models to *forecast* failures and anomalies in CCA which can be used to prevent failures.

Figure 1.1 presents the graphical representation of the research methodology to use to conduct this software engineering research.

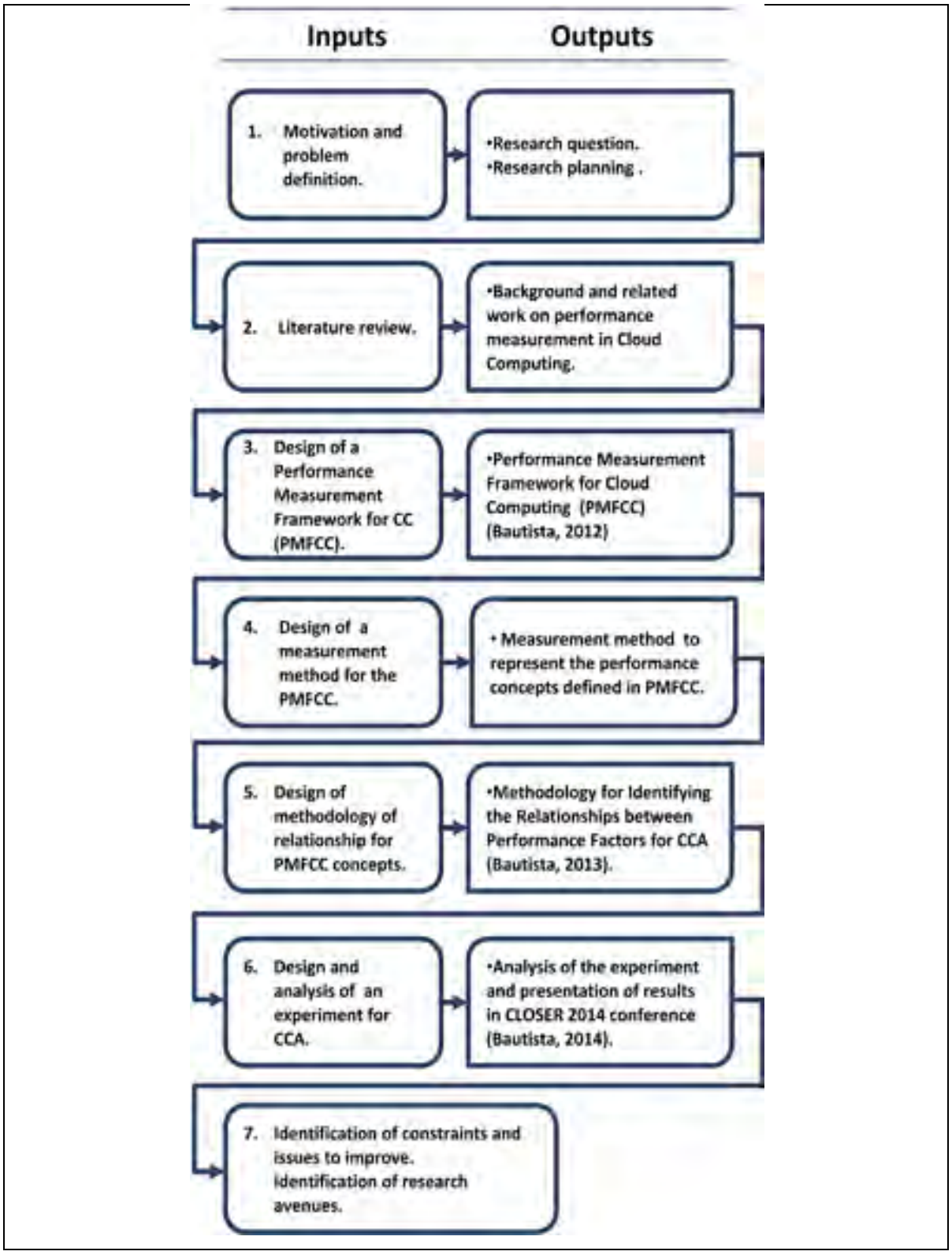


Figure 1.1 Graphical representation of research methodology.

The next chapter presents the literature review that introduce the concepts of quality models in software engineering, performance measurement process, Cloud Computing technology and the Hadoop CC technology used in this research.

CHAPTER 2

LITERATURE REVIEW

This chapter presents the literature review. Section 2.1 presents the most important quality model in software engineering: the ISO 25010 (SQuaRE) – System and software quality models which is an improvement of the ISO 9126: Software Product Evaluation Quality Characteristics and Guidelines. The study of this model is necessary to determine the main quality characteristics that are related to the performance concept of CCA in order to develop the PMMoCCA. Section 2.2 presents the measurement process in software engineering: this section describes the measurement reference model based on the ISO 25020 guide and also presents the ISO 15939 measurement process which describes the steps required to perform a measurement process. Once measurement concepts in software engineering are defined, section 2.3 describes the different approaches for performance measurement that could be used in the context of CCA. First, it summarizes the ISO 14756 standard which describes the measurement process for computer-based software systems. Then, it presents the different approaches to measure performance of CCS and CCA. Section 2.4 introduces the Cloud Computing paradigm and presents its architecture, type de services and concepts that are used by CC community. Finally, Section 2.5 describes the Hadoop technology which is emerging and used to process and store large amounts of data as well as to execute CCA. Finally it presents two open source CC technologies: the Hadoop distributed file system (HDFS), and the MapReduce programming model.

2.1 Quality models in software engineering

Over the last years, the software development industry has focused on improving the processes to develop products that satisfy user quality requirements. This has been known as "the user experience", which refers to software characteristics such as ease-of-use, security, stability and reliability (Côté, Suryn *et al.* 2006). In addition, Côté, Suryn *et al.* mentions that the software industry has defined the system quality as a very important part on the user experience. For example, the international standard ISO 25010 (ISO/IEC 2011) defines the

quality of a system as the degree to which the system satisfies the stated and implies needs of its various stakeholders, and thus provides value. Both, the software characteristics and stakeholders needs have been defined in a number of international standards by using quality models that categorize the software product quality and allow its evaluation.

As a result, there are different quality models proposals in software engineering which help in defining quality requirements and establishing the mechanisms to evaluate them from different "point of views". Next, we present the most recent quality model published in software engineering today.

2.1.1 ISO 25010 (SQuaRE) – System and software quality models

The ISO 25010 Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models (ISO/IEC 2011), revises the ISO 9126-1 standard and incorporates some new characteristics and corrections. These characteristics and corrections have been listed in the standard and are shown next according as they appear in the original document.

- The scope of the quality models has been extended to include computer systems, and quality in use from a system perspective.
- Context coverage has been added as a quality in use characteristic, with sub-characteristics context completeness and flexibility.
- Security has been added as a characteristic, rather than a subcharacteristic of functionality, with subcharacteristics confidentiality, integrity, non-repudiation, accountability, and authenticity.
- Compatibility (including operability and co-existence) has been added as a characteristic.
- The following subcharacteristics have been added: functional completeness, capacity, user error protection, accessibility, availability, modularity, and reusability.

- The compliance subcharacteristics have been removed as compliance with laws and regulations is part of overall system requirements, rather than specifically part of quality.
- The internal and external quality models have been combined as the product quality model.
- When appropriate, generic definitions have been adopted, rather than using software-specific definitions.
- Several characteristics and subcharacteristics have been renamed.

In addition, the ISO 25010 standard redefines two quality models which constitute the standard and are described as:

- The quality in use model which is composed of five characteristics that relate to the outcome of interaction when a product is used in a particular context of use. This model is applicable to the complete human-computer system, including both computer systems in use and software products in use and,
- The product quality model which is composed of eight characteristics that related to static properties of software and dynamics properties of computer systems. This model is applicable to both computer systems and software products

2.1.1.1 Quality in use model

The ISO 25010 standard mentions that the quality in use of an system characterizes the impact that the product (system or software product) has on stakeholders and it is determined by the quality of the software, hardware and operating environment, and the characteristics of the users, tasks and social environments (ISO/IEC 2011). The figure 2.1 presents the five characteristics of the quality in use model.

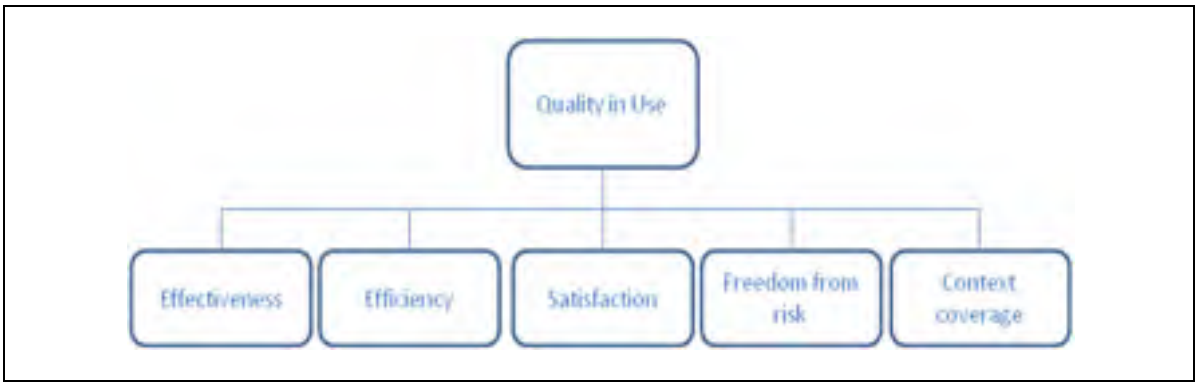


Figure 2.1 ISO 25010 Quality in use model characteristics

Next, the five characteristics are presented:

- Effectiveness. Accuracy and completeness with which users achieve specific goals.
- Efficiency. Resources expanded in relation to the accuracy and completeness with which users achieve goals.
- Satisfaction. Degree to which user needs are satisfied when a product or system is used in a specified context of use.
- Freedom from risk. Degree to which a product or system mitigates the potential risk to economic status, human life, health or environment.
- Context coverage. Degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in both specified contexts of use and in contexts beyond those initially explicitly identified.

2.1.1.2 Product quality model

Product quality model categorizes system and product quality properties which can be applied to a software product or to a computer system. The figure 2.2 shows the eight characteristics and subcharacteristics of the product quality model.

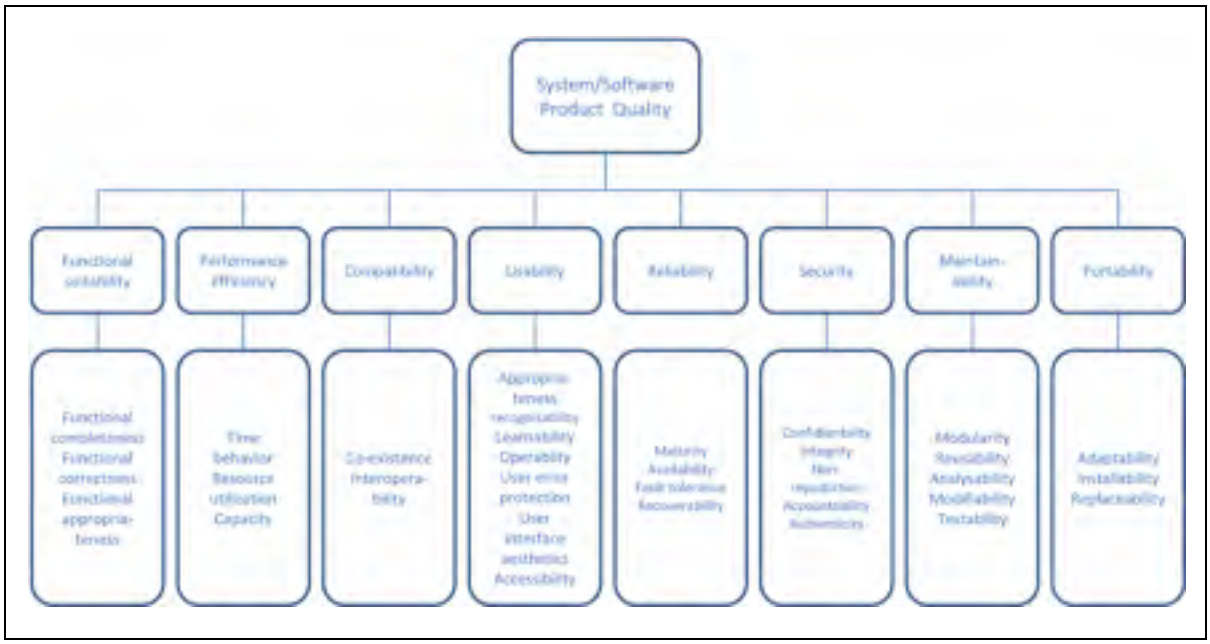


Figure 2.2 ISO 25010 characteristics and subcharacteristics of product quality model

In addition, the ISO 2010 (ISO/IEC 2011) standard mentions that the product quality model focuses on the target computer system which could include an information system, one or more computer systems and communication systems such as local area network and the internet. Each characteristic is defined as:

- Functional suitability: Degree to which a product or system provides functions that meet stated and implied needs when it is used under specified conditions.
- Performance efficiency: Performance relative to the amount of resources used under stated conditions.
- Compatibility: Degree to which a product, system or component can exchange information with other products, system or components, and/or perform its required functions, while sharing the same hardware or software environment.
- Usability: Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

- Reliability: Degree to which a system, product or component performs specified functions under specified conditions for a specific period of time.
- Security: Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.
- Maintainability: Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers.
- Portability: Degree of effectiveness and efficiency with which a product, system or component can be transferred from one hardware, software or other operational or usage environment to another.

Once defined, the quality properties must be associated with quality measures. According to ISO 25010 standard (ISO/IEC 2011), measures of the quality characteristic or subcharacteristic, can be directly measured, or a collection of properties need to be identified that together cover such characteristic or subcharacteristic, obtain quality measures for each, and combine them computationally. The measurement process of quality characteristics is an important part in the research because this defines the form in which the performance of cloud computing applications will be evaluated. The next section presents the measurement process used in software engineering in order to arrive to derive quality measures corresponding to the quality characteristic or subcharacteristic of a software system.

2.2 Measurement process in software engineering

The establishment of performance measurement models for CCA should be based on sound theory that defines a measurement process. Next a literature review is presented aimed at summarizing models for the measurement process and methods to represent results of analysis of performance.

2.2.1 ISO 25020 Software quality requirements and evaluation (SQuaRE) – Quality measurement – Measurement reference model and guide

The goal of the SQuaRE series of standards is to move to a logically organized, enriched and unified series of three complementary measurement processes: requirement specification, measurement and evaluation. As mentioned, the SQuaRE international standards include a series of technical reports describing quality model and measures, as well as quality requirements and evaluation which replace the previous ISO 9126 series.

According to ISO 25010 (ISO/IEC 2007), this standard provides a reference model and guide for measuring the quality characteristics defined in ISO 25020 standard and it is intended to be used together.

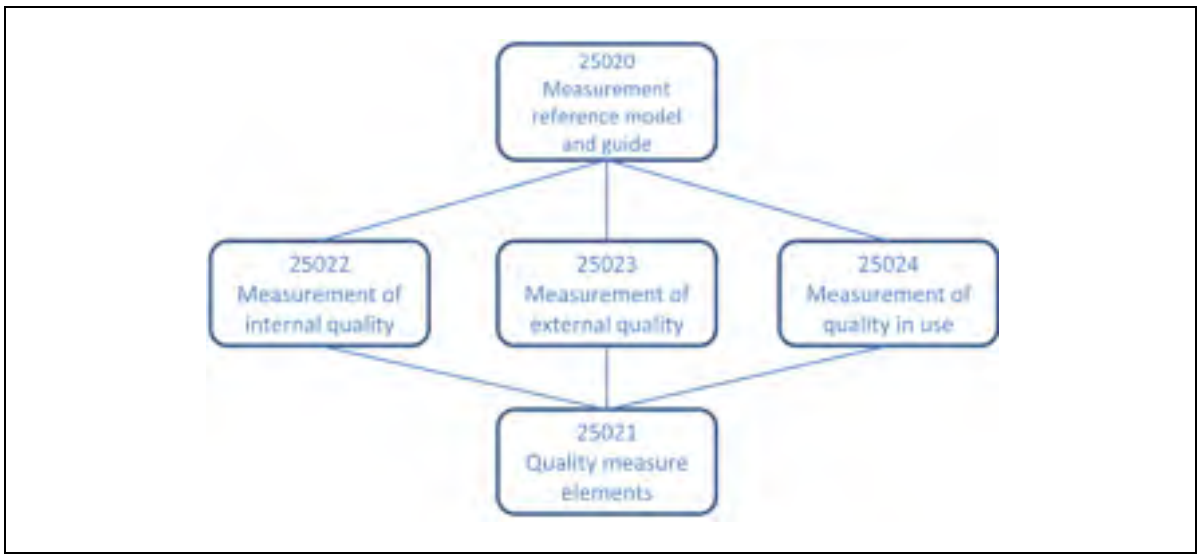


Figure 2.3 Relationship between SQuaRE series of standards

According to ISO 25020 the software product quality measurement reference model (SPQM-RM) provides information and guidance about how to measure the characteristics and subcharacteristics of a quality model. This standard provides a reference model and guide for measuring the quality characteristics defined in ISO 2501n Quality Model Division. The reference model depicts the relationship between quality measures, measurement functions,

measurement methods, and measure elements (ISO/IEC 2007). In addition, this standard mentions that quality measures are constructed by applying a measurement function to quality measure elements. Quality measure elements may be either base or derived measures and result from applying a measurement method to an attribute for the measurement of quality of a software product. The figure 2.4 shows the software product quality measurement reference model (SPQM-RM).

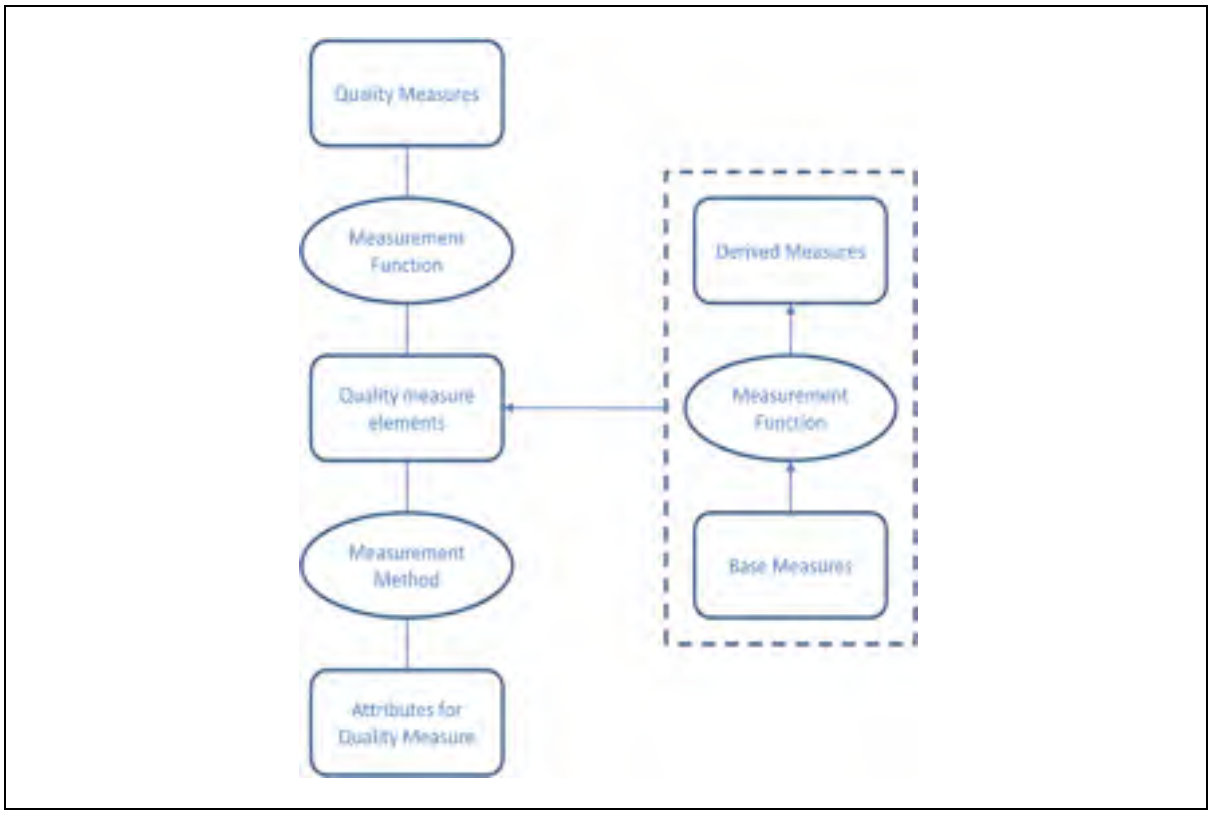


Figure 2.4 Software product quality measurement reference model (SPQM-RM)

It is important to note that the user of the ISO 25020 standard should plan and perform measurements following the SPQM-RM reference model using the procedure described in the ISO 15939 standard (ISO/IEC 2008) in order to obtain relevant measurements. The following section presents the ISO 15939 standard which defines the process to carry out the measurement of quality characteristics of software products.

2.2.2 ISO 15939 Measurement process

ISO 15939 model (ISO/IEC 2008) identifies the activities and tasks that are necessary to identify, define, select, apply and improve measurement within an overall project or organizational measurement structure. It also provides definitions for measurement terms commonly used within the system and software industries. On the other hand, this International Standard does not define an organizational model for measurement allowing to the user of this standard decide whether a separate measurement function is necessary to be integrated within different projects or across projects, based on the current organizational structure, culture and constrains. Thus, the ISO 15939 model should be integrated with the current organizational quality system and with a method to measurement of quality attributes.

2.2.2.1 Goal

According to ISO 15939 model, the purpose of the measurement process is to collect, analyze, and report data relating to the products developed and process implemented within the organizational unit, to support effective management of the process, and to objectively demonstrate the quality of the products (ISO/IEC 2008). The model defines that a product is the result of a process and can be classified into any of the four agreed product categories: hardware, software, services and processed materials.

2.2.2.2 Measurement process activities

ISO 15939 defines four sequenced activities into an iterative cycle allowing for continuous feedback and improving of the measurement process:

- Establish & Sustain Measurement Commitment: This activity consists of two tasks, (1) accept the requirements for measurement and (2) assign resources. Accept the requirements for measurement involves defining the scope of measurement such as a

single project, a functional area, the whole enterprise, etc., as well as the commitment of management and staff to measurement; this means that the organizational unit should demonstrate its commitment through policies, allocation of responsibilities, budget, training, etc. In addition, the assign resources task involves the allocation of responsibilities to individuals as well as to provide resources to plan the measurement process.

- Plan the Measurement Process: This activity consists of a series of activities such as identify information needs, select measures, define data collection, define criteria for evaluating the information of products and process. Also it includes the activities to review, approve and provide resources for measurement tasks.
- Perform the Measurement Process: This activity performs the tasks defined into the planning of measurement process across of the following sub-activities: integrate procedures, collect data, analyze data and development information products and finally communicate results.
- Evaluate Measurement: This activity evaluates the information products against the specified evaluation criteria providing conclusions on strengths and weaknesses of the information products and the measurement process. Also, this activity must identify potential improvements to the information products. For instance, changing the format of an indicator, changing from linear measure to an area measure, minutes to hours, or a line of code size measure, etc.

In addition, this model includes the Technical and Management Process (TMP) of an organization unit or project which is not within the scope of this standard but is an important external interface to the measurement activities.

The figure 2.5 illustrates the four activities in the process model and the relationship between them.

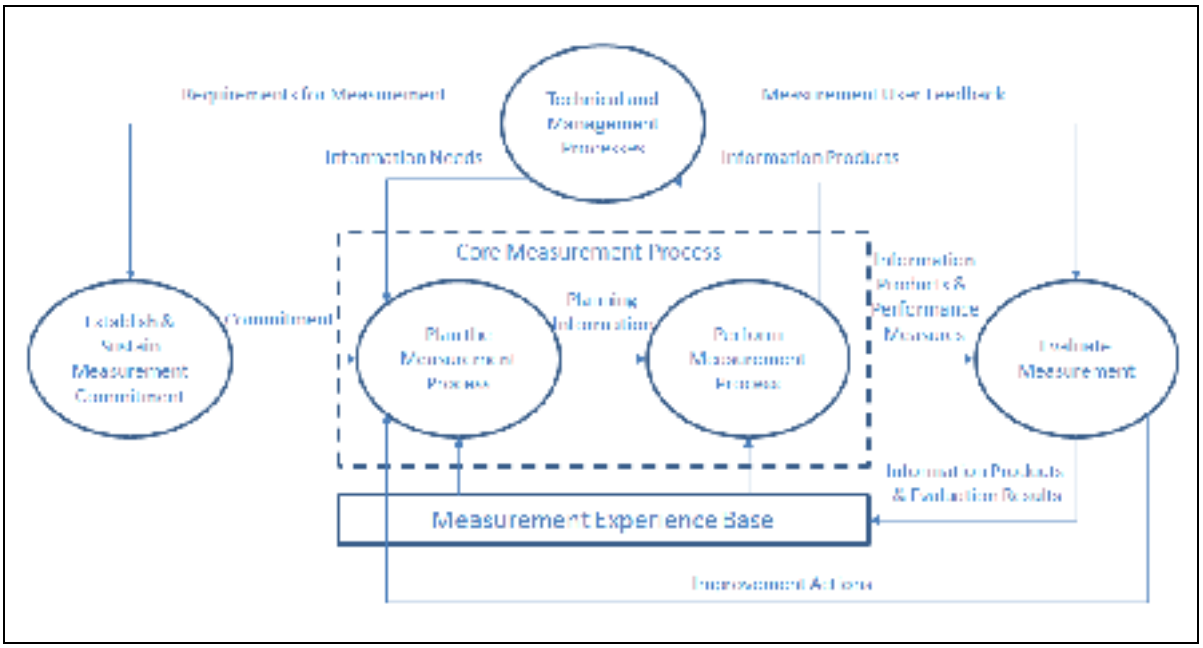


Figure 2.5 ISO 15939 Measurement process model activities

Two activities are considered to be the Core Measurement Process: 1) Plan the Measurement Process, and 2) Perform the Measurement Process; the other two activities (Establish and Sustain Measurement Commitment and Evaluate Measurement) provide a foundation for the Core Measurement Process and provide feedback to it (ISO/IEC 2008).

The Measurement Experience Base is intended to capture information products from past iterations of the cycle, previous evaluations of information products, and evaluations of previous iterations of the measurement process. Since the process model is cyclical, subsequent iterations may only update measurement products and practices.

The ISO 15939 model presents the process to identify, define, select, apply and improve measurement in a project, organizational structure or software product. This model is an important part in this research because the activities proposed for the measurement process of performance of CCA are tied to the different activities described in ISO 15939 measurement process. The next section presents different methods for measuring the performance of computer systems which include computer based software systems (CBSS), CCS and CCA.

2.3 Performance measurement of computer systems

2.3.1 ISO 14756 measurement process model for computer based software systems (CBSS)

ISO 14756 Measurement Process Model (ISO/IEC 1999) measures and rates the performance of computer-based software systems (CBSS). A CBSS includes hardware and all its software (system software and application software) which are needed to realize the data processing functions required by the user. The measurement consists in calculating the performance values of throughput and execution time.

The final result of a performance assessment of a Computer-Based Software System (CBSS) consists of the rating of these values which are gained by comparing the calculated performance values with the user's requirements. Also, with the guidance of this international standard it is possible to rate the performance values of CBSS under test by comparing them with some CBSS referenced values. For instance having the same hardware configuration but another version of the application program with the same functionality (ISO/IEC 1999).

ISO 14756 standard, defines also how the user oriented performance of CBSS may be measured and rated. Thus, the specific performance values are those which describe the execution speed of user orders (tasks) as for example, execution time, throughput and timeliness. In this case, a task may be a job, transaction, process or a more complex structure, but with a defined start and end depending on the needs of the evaluator. Also, it is possible to use this standard for measuring the time behavior with reference to business transaction times and other individual response times.

2.3.1.1 Recommended steps of measurement process of CBSS

ISO 14756 standard states that CBSS to be measured shall consist of; a specified configuration of its hardware, its system software and application software. This means, all

the hardware components and all software components shall be specified in detail and none of them shall have any change or special modification while the measurement process to getting better results in the measurement.

Once a system configuration is specified, it is possible to perform the measurement process which consists of three basic time phases:

1. Stabilization phase: The stabilization phase is needed to bring the system under test (SUT) in a stable state of operation. During this phase the tasks should be submitted according to the workload parameter set.
2. Rating interval phase. During the rating interval each task submitted is taken into account for rating. Its duration has to be chosen appropriately to the application which is represented by the software under test.
3. Supplementary run phase. At the end of the rating interval, the test application (TA) shall not be stopped yet. It shall continue submitting tasks as specified by the workload parameter set until all tasks (including those which were submitted within the rating interval) are completed.

The figure 2.6 presents the three basic time phases of the measurement process of CBSS as recommended by ISO 14756.

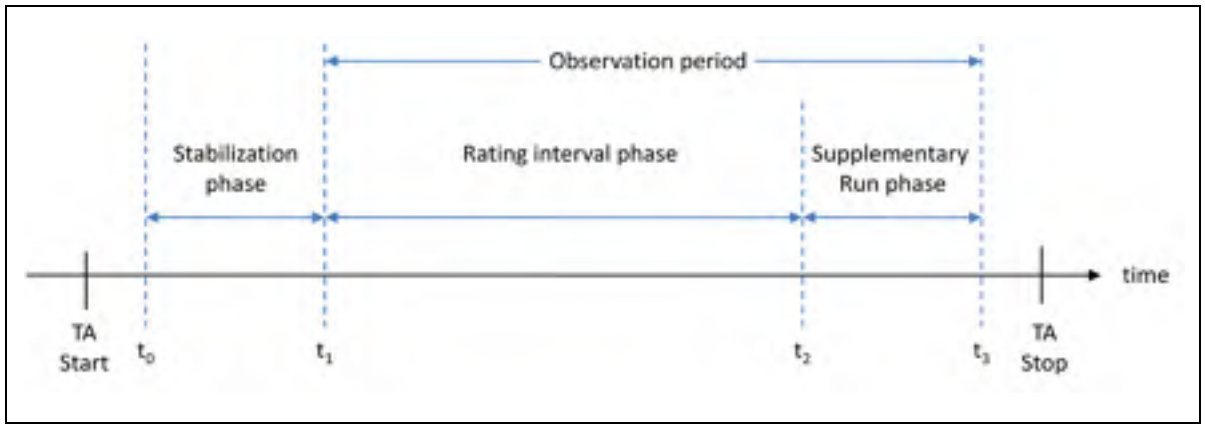


Figure 2.6 Time phases of the measurement process of CBSS (ISO 14756)

In addition, ISO 14756 provides specific guidelines to measure and rate the performance of CBSS that have random user behavior when accuracy and repeatability is required. The guideline specifies in detail how to prepare and carry out the measurement process describing the analysis of the measured values, the formulas for computing the performance value and the rating values.

Next, examples are presented that have been used to measure Cloud Computing technologies such as Cloud Computing systems (CCS) and Cloud Computing Applications (CAA).

2.3.2 Performance measurement of cloud computing systems

Many publications on the performance measurement of technologies related with Cloud Computing Systems (CCS) and Cloud Computing Applications are found in the literature. Included is a summary of recent research. Each research has been developed from specific viewpoints such as: load balancing, network intrusion detection, or maintenance of the host state. According to Burgess, modern computer systems are complex for many reasons: they are organisms composed of many interacting subsystems, whose collective behavior is intricate and, at the same time, determines the system performance (Burgess, Haugerud *et al.* 2002).

Burgess notes that complex systems are characterized by behavior at many levels or scales and that in order to extract knowledge from a complex system it is necessary to focus on an appropriate scale. Burgess mentions that three scales are usually distinguished in many complex systems: the microscopic scale which is related with atomic transactions in the order of milliseconds such as: system calls. The mesoscopic scale which includes clusters and patterns of systems calls or other process behaviors such as a group of process owned by a single user (on the order of seconds). Finally, the macroscopic scale which is related with the user activities, in scales such as minutes, hours, days and weeks. The figure 2.7 shows the three scales used to measure complex systems according to Burgess.

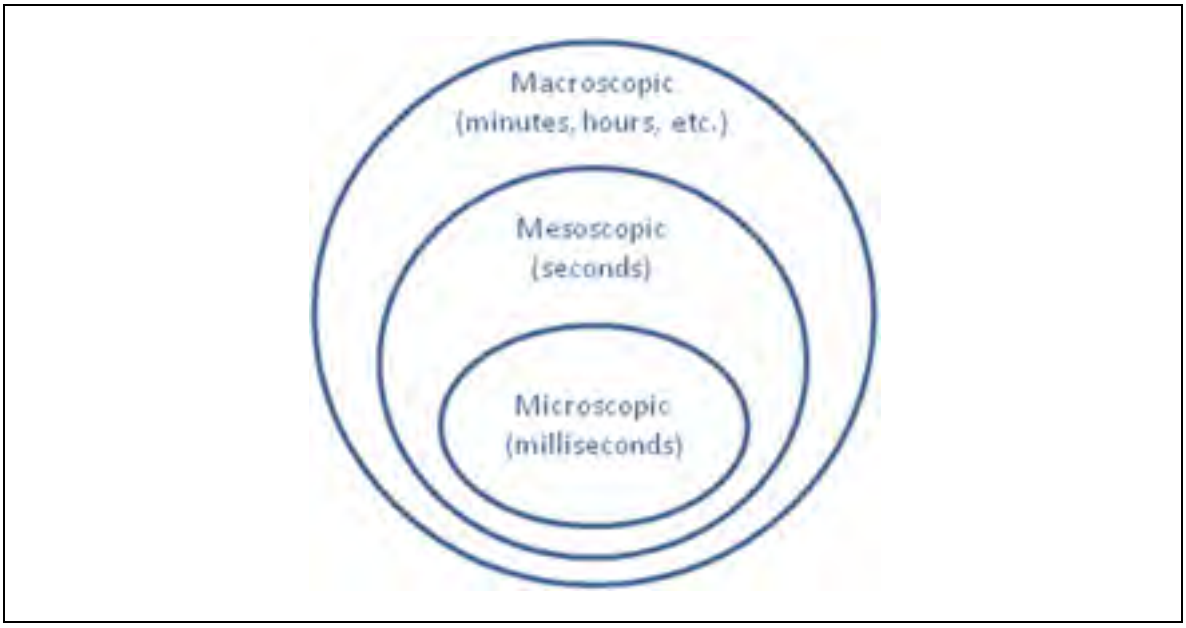


Figure 2.7 Scales to measurement of complex systems

According to Burgess, two of the main problems on measurement of system performance are data collection and definition of variables which describe how the system will behave. Burgess mentions that many system variables have long terms rhythms that closely follow human patterns of behavior while others may have trends that arise due to other environmental effects such as system policies or artifacts of collaborating subsystems where such patterns need to be subtracted. (Burgess, Haugerud *et al.* 2002).

On the other hand, one important aspect in system performance is to define what normal behavior means. To Burgess, normal behavior means average behavior from a statistical point of view. This means that normal behavior is determined by learning about past events and by modeling future behavior based on a statistical trend of the past and an observation of present (this assumes that an average or expectation value can be defined). Therefore, normal system behavior could be defined using data collection and the measurements made during a discrete time interval which could help to determine system performance.

Other authors have tried to predict the performance of complex systems, such as clusters of computers, by simulating the cluster behavior using a virtual environment. For instance, Rao (Rao, Upadhyay *et al.* 2009) estimates the variation of cluster performance through changes in task sizes as well as the time taken to run a process solution for a particular problem. As a result at this experiment, Rao built a predictive model using regression analysis which enables to model the behavior of the system and try to predict the performance of a real cluster. However, this study does not show the real performance of the cluster of computers because it presents only a model of the possible behavior of one cluster according to very specific controlled variables and well defined tasks, leaving out variables related to users experience and applications behavior.

Other authors have focused on modeling the reliability of large, high-performance, computer systems to try to measure the system performance. For example, Smith (Smith, Guan *et al.* 2010) mentions that failure occurrence has an impact on system performance as well as operational costs and proposes an automated model to detect anomalies that aims to identify the root causes of problems or faults. According to Smith localizing anomalies in large scale-complex systems is a difficult task. This is due to the data volume and its diversity, its dependency to external factors, the anomaly characteristics, etc. Thus, finding anomalies in such amount of diverse data is a challenge, especially how to discover their dependency between multiple factors and how to remove noise in itself.

To try to address these issues, Smith proposes an automatic anomaly detection framework that can process massive volume of diverse health-related data by means of pattern recognition technologies. In his case study, health-related data is collected from the system and sent for analysis that includes: data transformation, feature extraction, clustering and outlier identification. Taking into account the structure of cloud computing systems, Smith proposes health-related variables which are used by his anomaly detection framework and are related with the system or user utilization, CPU idle time, memory utilization, volume de I/O operations, and many other measures. The analysis results are classified into multiple groups using clustering, and an outlier detector technique identifies the nodes that are far away from the majority as potential anomalies. Finally, the potential list of anomalies is sent to system administrators for manual validation and to combine it with human expertise to quickly discover anomalies with high accuracy (Smith, Guan *et al.* 2010).

Although the anomaly detection framework proposed by Smith has many interesting and valuable proposals it does not analyze important aspects related with other potential factors which can affect system performance, such as: the design and use of applications, ordering and prioritization of the submitted tasks or variations in data distribution across the system besides user behaviors. Therefore, improvements to his performance measurement model, which includes these aspects, may allow a more realistic view of a system performance measurement.

2.3.3 Performance measurement of cloud computing applications

Other researchers have also analyzed the performance of CCA from various viewpoints. For example, Jackson (Jackson, Ramakrishnan *et al.* 2010) analyzed high performance computing applications of the Amazon Web Services. His purpose was to examine the performance of existing CC infrastructures and create a model to quantitatively evaluate them. He is focused on the performance of Amazon EC2 in which Jackson quantitatively examined the performance of a set of benchmarks designed to represent a typical High Performance Computing (HPC) workload running on Amazon EC2. Timing results from

different application benchmarks are captured to compute the Sustained System Performance (SSP) measure to assess the performance delivered by the workload of a computing system. According to the National Energy Research Scientific Computing Center (NERSC) (Kramer, Shalf et al. 2005), SSP is useful to measure a system performance across any time frame, and can be applied to any set of systems, any workload, and/or benchmark suite, and for any duration. SSP measures time to solution across different application areas and could be used to evaluate absolute performance and performance relative to cost (in dollars, energy or other value propositions). The research results show a strong correlation between the percentage of time an application spends communicating, and its overall performance on EC2. The more communication there is, the worse the performance becomes. Jackson also concludes that in this situation the communication pattern of an application can have a significant impact on performance.

Other researchers have focused on applications in virtualized Cloud environments. For instance, Mei (Mei, Liu *et al.* 2010) studies performance measurement and analysis of application network I/O (network-intensive applications) in a virtualized Cloud. The objective of his research is to understand the performance impact of co-locating applications in a virtualized Cloud, in terms of throughput performance and resource sharing effectiveness. Mei addresses issues related to managing idle instances, which are processes running in an operating system (OS) that are executing idle loops. His results show that when two identical I/O applications are running currently, schedulers can approximately guarantee that each has its fair share of CPU slicing, network bandwidth consumption, and resulting throughput. He also demonstrates that the duration of performance degradation experienced by the system administrator is related to machine capacity, workload degree in the running domain, and number of new virtual machine (VM) instances to start up.

Authors like Alexandru (Alexandru 2011) analyze the performance of CC services for Many-Task Computing (MTC) system. According to Alexandru, scientific workloads often require High-Performance Computing capabilities: this means high performance execution of loosely coupled applications comprising many tasks. By means of this approach systems can operate

at high utilizations, like to current production grids. Alexandru analyzes the performance based on the premise that CCS can execute MTC-based scientific workload with similar performance and at lower cost than scientific processing systems. For this, the author focuses on Infrastructures as a Service (IaaS) providers on public clouds not restricted within an enterprise. Alexandru selected four public clouds providers (Amazon EC2, GoGrid, ElasticHosts and Mosso) to perform a traditional system benchmarking to provide a first order estimate of the system performance. Alexandru mainly uses measures related to disk, memory, network and cpu to determine the performance through the analysis of MTC workloads which comprise tens of thousands to hundreds of thousands of tasks. His main finding is that the compute performance of the tested clouds is low compared to traditional systems of high performance computing. In addition, Alexandru found that while current cloud computing services are insufficient for scientific computing at large, they are a good solution for scientists who need resources instantly and temporarily.

Although these publications present interesting models for performance measurement of CCA, their approach is from an infrastructure standpoint and does not consider CCA performance factors from a software engineering application perspective. Consequently, the focus of this thesis is on the performance evaluation of CCA, and more specifically on performance measurement of data intensive applications like Hadoop CCA, and by integrating software quality concepts from ISO 25010 and frameworks for CCS performance measurement.

The next section introduces Cloud Computing concepts.

2.4 Cloud computing

Cloud Computing is an Internet-based technology in which several distributed computers work together to process information in a more efficient way and deliver results more quickly to the users who require them. In general, users of Cloud Computing do not necessarily own the physical technology. Instead, they can rent the infrastructure, the platform, or the

software from a third-party provider. This section presents an overview of Cloud Computing, as well as a definition of this new technology, its organization, and its usefulness.

2.4.1 Definition and type of services in cloud computing

There are many definitions of Cloud Computing (CC), mainly because it is a new computing paradigm and it makes use of several distributed systems (DS) tools, architectures, and platforms in order to offer High-Performance Computing (HPC) that can be shared by different users in the same geographical location or in distant locations. Some of the features that have led to the rapid growth of CC are its low cost and its relatively simple implementation compared to traditional HPC technologies, such as Grid Computing or Massively Parallel Processing (MPP) systems. It is clear, for example, that CC eliminates much of the work of application deployment, thanks to its simple but powerful application development frameworks.

According to the ISO subcommittee 38 – the study group on Cloud Computing (ISO/IEC 2012), CC is a paradigm for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable cloud resources accessed through services, that can be rapidly provisioned and released with minimal management effort or service provider interaction. Other authors such as Jin (Jin, Ibrahim et al. 2010), define CC as the hardware, system software, and applications delivered as services over the Internet, where a cloud is called a public cloud when it is made available in a pay-as-you-go manner to the general public, and is called a private cloud when the cloud infrastructure is operated solely for a business or an organization".

Jin also describes a combination of these two types of cloud, which is called a Hybrid Cloud, in which a private cloud is able to maintain high service availability by scaling up its system from a public cloud. In addition, the ISO Information technology - Cloud Computing - Reference Architecture (ISO/IEC 2013) mentions that cloud services are grouped into categories, where each category possesses some common set of qualities. Cloud service

categories include: 1) Infrastructure as a Service (IaaS), 2) Platform as a Service (Paas), 3) Software as a Service (SaaS) and 4) Network as a Service (NaaS), and services in each category may include capabilities from one or more than one of the cloud capabilities type. These four service models include all the technical resources that clouds need in order to process information, like software, hardware, and network elements:

- Software-as-a-Service (SaaS): This type of service offers applications which are accessible from several client devices through a thin client interface, such as a Web browser or light interface application. This kind of service is also known as the application service provider (ASP) model, as the provider is responsible for the application where common resources, the application, and databases support multiple clients simultaneously. Examples of these service providers are SalesForce.com, NetSuit, Oracle, IBM, and so on.
- Platform-as-a-Service (PaaS): In this category, the provider supplies all the systems (operating systems, applications, and development environment) and delivers them as a cloud-based service, and the client is responsible for the end-to-end life cycle in terms of developing, testing, deploying, and hosting sophisticated Web applications. Examples of this kind of provider are Google's appEngine, Microsoft´s Azure, etc. Thus, in this type of service the provider manages the storing and development frameworks such as Windows Azure Development Tools or SQL Azure. The main difference between the SaaS and the PaaS is that in the former the client uses a commercial application and is not involved in any of the development stages.
- Infrastructure-as-a-Service (IaaS): In this category, the service (storage and computing power) is offered through a Web-based access point, where the service client does not need to manage or control the underlying cloud infrastructure, but has control over the operating system, storage, and applications. Some examples of infrastructure providers are GoGird, AppNexus, Eucalyptus, Amazon EC2, etc. In this type of service the client manages the storing and development environments for Cloud Computing application such as the Haddop Distributed File System (HDFS) and the MapReduce development framework.

- Network-as-a-Service (NaaS): In this category, the service (transmission over the network) is offered by means of opening the network to value added subscriber services, created by third party developers, and charging for the use of the service on a pay. It provides services that leverage the power of the network-enabled IT utilization. Network-as-a-service (NaaS) is a business model for delivering network services virtually over the Internet on a pay-per-use or monthly subscription basis. From the customer's point of view, the only thing required to create an information technology (IT) network is one computer, an Internet connection and access to the provider's NaaS portal. This concept can be appealing to new business owners because it saves them from spending money on network hardware and the staff it takes to manage a network in-house. In essence, the network becomes a utility, paid for just like electricity or water or heat. Because the network is virtual, all its complexities are hidden from view.

Figure 2.8 presents an overview of Cloud Computing components and their interactions.

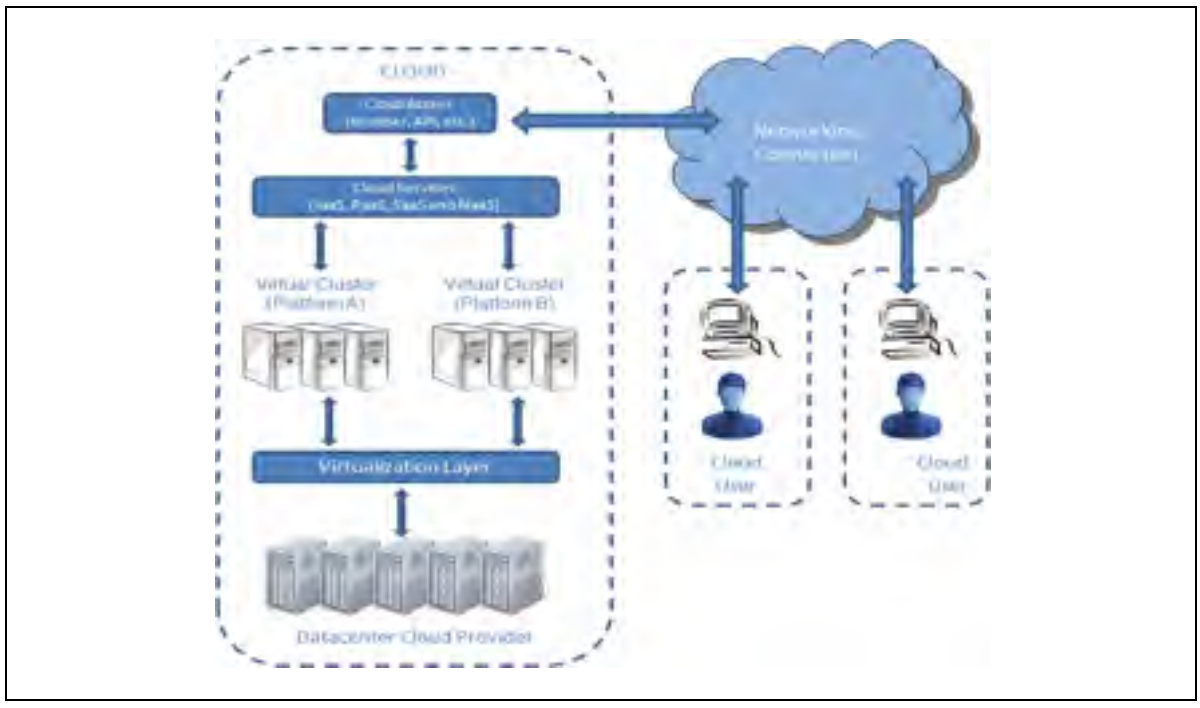


Figure 2.8 Basic components of Cloud Computing

Prasad mentions (Prasad, Choi *et al.* 2010) that one goal of this computing model is to make better use of distributed resources, put them together in different ways to achieve higher throughput, and be able to resolve large-scale computation problems. So, Cloud Computing addresses various domains of information technology and software engineering like virtualization, scalability, interoperability, quality of service, failover mechanisms, and so on. As a result, it is necessary to establish an architecture which defines the basic elements of CC technology and the form in which they are organized.

2.4.2 Architecture

A number of Cloud Computing architectures have been proposed by analysts, academics, industry practitioners, and IT companies (Prasad, Choi *et al.* 2010). However, they all have one characteristic in common: they organize the elements of Cloud Computing from the perspective of the end-user. Prasad proposes an architecture which includes the basic elements of Cloud Computing, and where the architecture consists of the design of software applications that make use of resources and services on-demand through the Internet.

As a result, the Cloud Computing architecture underlies an infrastructure that is used only when it is needed to draw the necessary resources on-demand and perform a specific job, and then the unneeded resources are released and the used resources are disposed of after the job has been completed. Figure 2.9 summarizes the elements of a CC architecture.

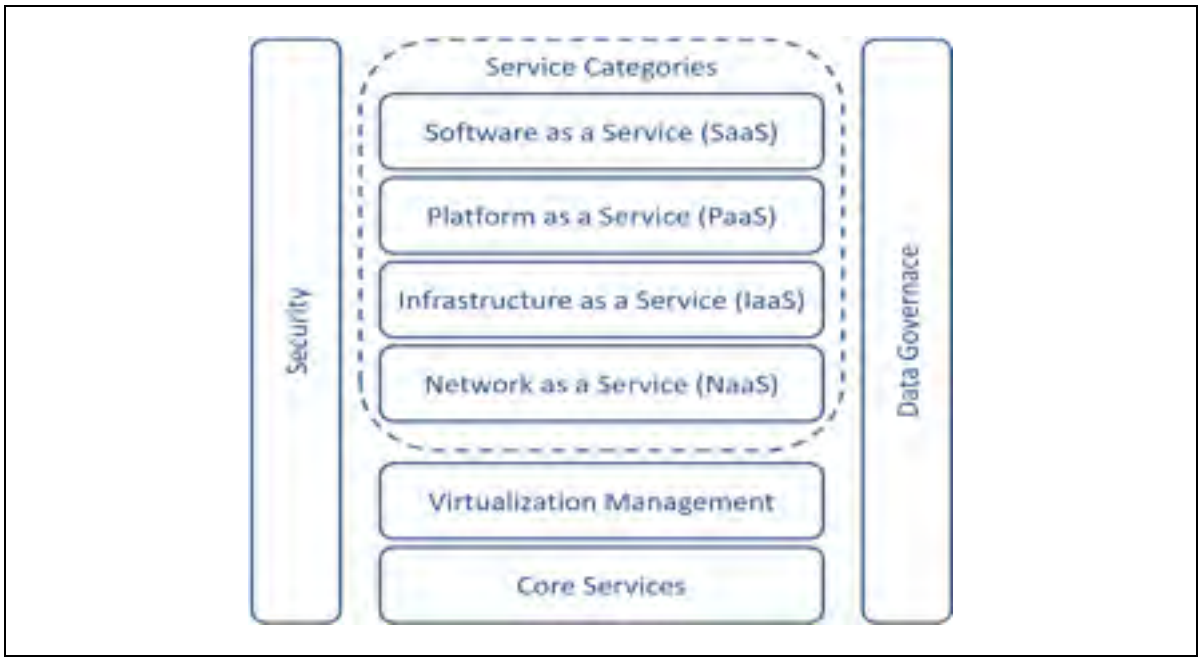


Figure 2.9 Components of the Cloud Computing architecture

The above architecture defines five main components which group elements of Cloud Computing such as: Core Services, Virtualization Management, Services, Security, and Data Governance. Table 2.1 shows a description of each of the components.

Table 2.1 Description of Cloud Computing architecture components

Component	Description
Core Services	The Core Services are resource management services that deal with the protocols of different kinds of infrastructure environments: the load balancing that prevents system bottle-necks because multiple accesses and the simultaneous use of replication schemes, as well as discovery services to allow for reusability and changes to the basic infrastructure configuration.
Vitalization Management	Virtualization Management is the technology that abstracts the coupling between the hardware and the operating system. There are several types of virtualization: server virtualization, storage virtualization, and network virtualization in which the resources in a virtualized environment can be dynamically created, expanded, shrunk, or moved as the demand varies.
Services	Services are the final products in Cloud Computing that are delivered and consumed in real time over the Internet. There are four types of services in Cloud Computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) and Network as a Service.
Security	Security is one of the main issues for enterprises considering moving their in-house data to public clouds, and much of the discussion around Cloud Computing is focused on this topic. One of the main concerns with respect to security is that client and employee information, consumer profiles, business plans, etc. could fall into the wrong hands, creating the potential for civil liability and criminal charges.
Data Governance	When data begin to move out of organizations, it becomes vulnerable to disclosure or loss. The act of moving sensitive data outside the organizational boundary may also violate national privacy regulations. Governance in the Cloud places a layer of processes and technology around services (location of services, services dependencies, service monitoring, and service security), so that anything that occurs will be quickly known.

Although each of the components of this new computing paradigm is an object of research and discussion, there are some major aspects to consider at the design and implementation stages of the Cloud Computing technologies. One of these is the highly relevant topic of the fault tolerance which is related to reliability of service in the case of system failure, system anomalies and cloud performance. To illustrate, Microsoft Azure had an outage that lasted 22 hours on March 13-14, 2008 (Dohnert 2012), and Google has had numerous performance difficulties with its Gmail application services. So, finding new mechanisms to provide

service reliability in Cloud Computing has recently become an issue of major interest, and our study will concern aspects related to service components at the operating system level.

In summary, to deliver highly available and flexible services, Cloud providers must find mechanisms to offer reliability different types of configurations in their data centers, and in this form to be able to divide and measure resources between different types of clients. This is achieved through Cloud technologies such as the Hadoop project. The Hadoop technology which is used in this thesis to develop the PMMoCCA is presented next.

2.5 Hadoop technology

In the previous section a CC architecture which shows different components in Cloud Computing technologies was defined. The Service Component (SC) is one of the most important elements in this architecture since it groups the SaaS, PaaS, IaaS and NaaS service categories. The SC uses different technologies to offer storing, processing, and developing through different frameworks for managing CCA. Hadoop is one of the most used technologies within SC because it offers open source tools and utilities for Cloud Computing environments. Although there are several kinds of application development frameworks for CC, such as GridGain, Hazelcast, and DAC, Hadoop has been widely adopted because of its open source implementation of the MapReduce programming model which is based on Google's MapReduce framework (Dean and Ghemawat 2008). In addition, Hadoop includes a set of libraries and subsystems which permit the storage of large amounts of information, enabling the creation of very large data tables or summarize data with tools of data warehouse infrastructure.

2.5.1 Description

Hadoop is the Apache Software Foundation's top-level project that holds the various Hadoop subprojects. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines,

each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. The Hadoop project provides and supports the development of open source software that supplies a framework for the development of highly scalable distributed computing applications which handles the processing details, leaving developers free to focus on application logic (Hadoop 2014).

2.5.2 Hadoop subprojects

According to White (White 2012), Hadoop is divided into nine subprojects that fall under the umbrella of infrastructure for distributed computing:

- Common. A set of components and interfaces for distributed file systems and general I/O (serialization, Java RPC, persistent data structures).
- Ambari: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and the ability to view MapReduce, Pig and Hive applications visually along with features to diagnose their performance characteristics in a user-friendly manner.
- Avro. This is a data serialization system for efficient, cross language RPC, and persistent data storage.
- Cassandra. A scalable multi-master database with no single points of failure.
- Chukwa. This is a data collection system for monitoring large distributed systems. It is built on top of the Hadoop Distributed File System (HDFS) and the MapReduce framework, and includes a flexible and powerful toolkit to produce reports based on the data collected.
- HBase. A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce. HBase uses

random, real-time, read/write access to substantial amounts of data is required. The project's goal is to host very large tables on top of clusters of commodity hardware.

- HDFS. The Hadoop Distributed File System, the primary storage system used by Hadoop applications, creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable and extremely rapid computations.
- Hive. This is a distributed data warehouse infrastructure to manage data stored in the HDFS. It provides tools to make it easy to summarize data, and for ad hoc querying and analysis of large datasets through a query language based on SQL.
- Mahout: A Scalable machine learning and data mining library.
- MapReduce. This is a distributed data processing model and execution environment that runs on large clusters of commodity machines.
- Oozie: A service for running and scheduling workflows of Hadoop jobs (including Map-Reduce, Pig, Hive, and Sqoop jobs).
- Pig. This is a platform for analyzing large datasets which consists of high-level language for expressing data analysis programs coupled with infrastructure for evaluating these programs. It runs on HDFS and MapReduce clusters.
- Spark: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- Sqoop: A tool for efficient bulk transfer of data between structured data stores (such as relational databases) and HDFS.

  ZooKeeper. This is a distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.

The Figure 2.10 shows the Hadoop subprojects and where they are located.

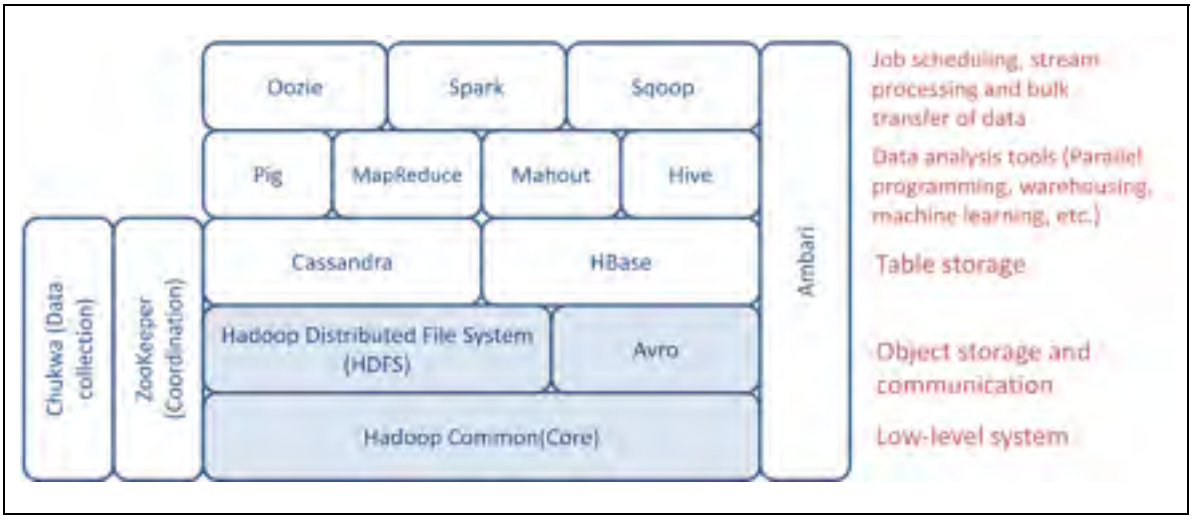


Figure 2.10 Hadoop subprojects and their location

Next, a more detailed description of HDFS and Mapreduce subproject are described, since these technologies will be used in this research.

2.5.3 Hadoop Distributed File System (HDFS)

2.5.3.1 HDFS Goals

HDFS is focused on storing very large files with streaming data access patterns, running on clusters on commodity hardware. According to White, the HDFS has been designed to achieve three main goals, which are:

- To handle very large files. The HDFS is capable of handling files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.
- Streaming data access. The HDFS is built around the idea that the most efficient data processing pattern is write once, read many times, where a dataset is typically generated or copied from a source, after which various analyses are performed on the

dataset over time, and each analysis can involve a large portion, if not all, of the dataset.

- Commodity hardware. Hadoop has been designed to continue to run smoothly on a commodity hardware cluster when a node fails, without a noticeable interruption, which is an important feature, as the probability of such a failure across the cluster, especially a large one, is high.

2.5.3.2 HDFS Architecture

According to Dhurba (Dhruba 2010), the HDFS has a master/server architecture, where an HDFS cluster consists of a single NameNode (NN), which is a server that manages the File System Namespace (FSN) and regulates access by the clients to files. In addition, there are a number of Data Nodes (DN), usually one per node in the cluster which manages the storage attached to the nodes on which they run.

Internally, a file is split into one or more blocks, and these blocks are stored in a set of DN. DN are responsible for handling read and write requests from the file system's clients. They also perform block creation, deletion, and replication with instructions from the NN. Both DN and NN are pieces of software designed to run on commodity machines with a GNU/Linux operating system. The HDFS was built using the Java language, so any machine that supports Java can run the DN or NN software. The existence of a single NN in the cluster greatly simplifies the architecture of a system where the NN is the arbitrator and repository for all HDFS metadata. Figure 2.11 shows the HDFS architecture.

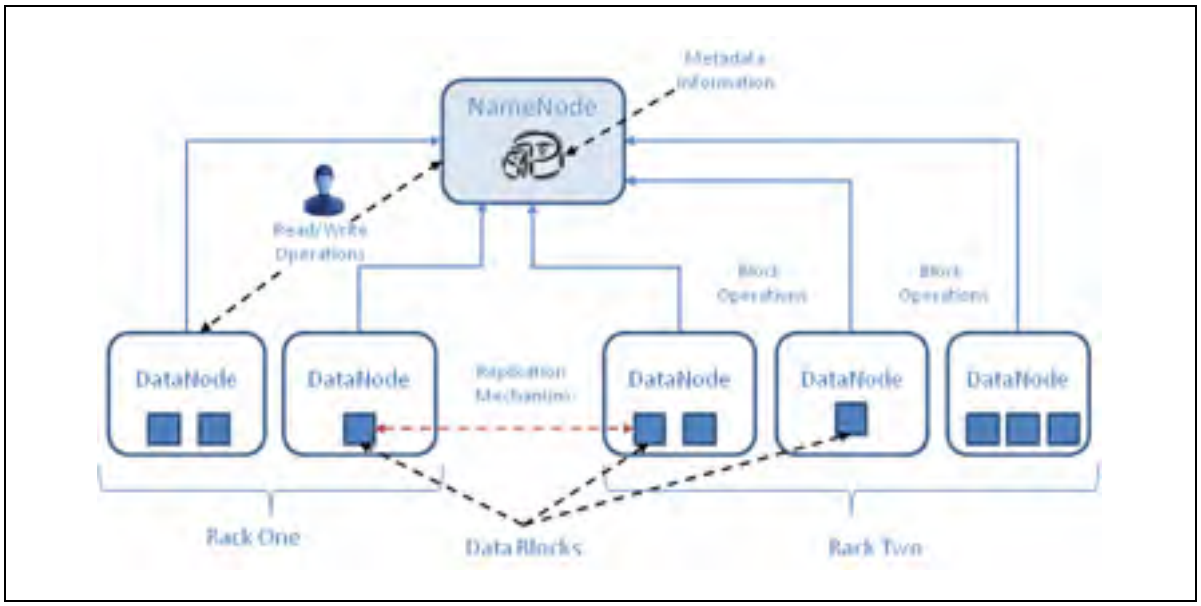


Figure 2.11 Hadoop Distributed File System Architecture

2.5.3.3 HDFS Components

As it was mentioned, the HDFS architecture consists of a single NameNode (NN) and several DataNodes (DN). The NN server is made up of two main elements which expose the file system to users and allow data storage retrieval. These elements are the File System Namespace (FSN) and the File System Metadata (FSM).

- File System Namespace. The HDFS supports traditional hierarchical file organization, where a user or an application can create directories and store files in them. The hierarchy of the FSN is similar to that of other file systems on which the basic operations on files (create, delete, move, or rename) can be performed; however, the HDFS does not implement user quotes, access permission, or hard links and soft links. Thus, any change to the FSN or its properties is recorded by the NN.
- File System Metadata. When the FSN is stored by the NN, it uses a transaction log, called EditLog, to persistently record every change that occurs to file system

metadata. For example, creating a new file in the HDFS causes the NN to insert a record into the EditLog indicating this. Similarly, changing a file causes a new record to be inserted into the EditLog, and so on. The entire FSN, including the mapping of blocks to files and file systems properties, is stored in a file, called the FsImage, which is stored as a file in the NN's local file system. Both the EditLog and FsImage are part of FSM.

On the other hand, the DN stores HDFS data in files in its local file system, but has no knowledge of the HDFS files. It stores each block of HDFS data in a separate file using a heuristic to determine the optimal number of files per directory and creates subdirectories as required. Storing all the local files in the same directory is not optimal, because the local file system might not be able to efficiently support a huge number of files in a single directory. When the DN starts up, it sends a *Blockreport* to the NN, which is a list of all the HDFS data blocks located in its local file system. At the same time, each block is replicated through several DN to create a distributed file system that will be reliable if a DN failure occurs.

2.5.4 Hadoop MapReduce programming model

MapReduce is a programming model and an associated implementation developed by Google for processing and generating large data sets (Dean and Ghemawat 2008). According to Dean, programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. On the other hand, Lin mentions (Lin and Dyer 2010) that the approach to tackling large-data problems today is to divide and conquer, in which the basic idea is to partition a large problem into smaller sub problems. Thus, those sub problems can be tackled in parallel by different workers for example, threads in a processor core, cores in a multi-core processor, multiple processors in a machine or many machines in a cluster. In this form, intermediate results from each individual worker are then combined to yield the final output.

Other authors as Venner mentions (Venner 2009) that Hadoop support the MapReduce model making use of the HDFS within a cluster of inexpensive machines to run MapReduce applications where the operation of MapReduce model is based on two main stages to get results which are:

- Map stage or also called mapping, in this phase a list of data elements are provided, one at time, to a function called the Mapper, which transforms each element individually to an output data element. The figure 2.12 shows the mapping stage which creates a new output list by applying a function to individual's elements of the input list.

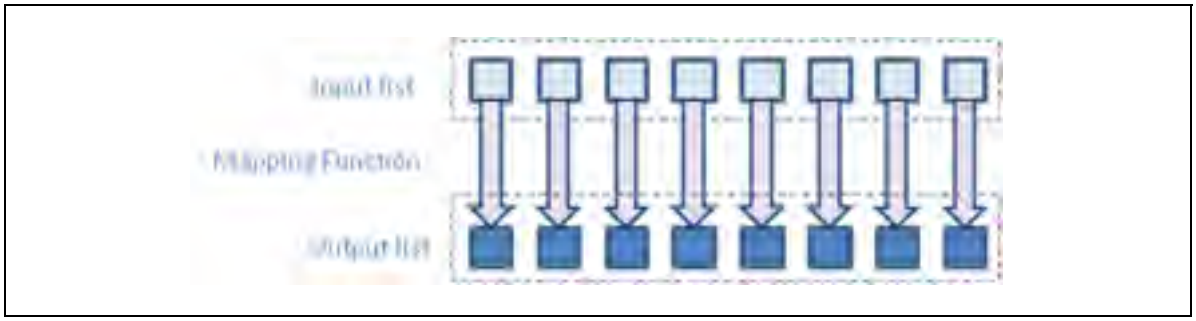


Figure 2.12 Mapping stage that creates a new output list

- The second stage is the Reduce stage or also called reducing, this phase lets to aggregate values together. A reducer function receives an iterator of input values from an input list. This then combines these values together returning a single output value. Reducing stage is often used to produce "summary" data, turning a large volume of data into a smaller summary of itself. Figure 2.13 show the reduce stage over the input values to produce an aggregate value as output.

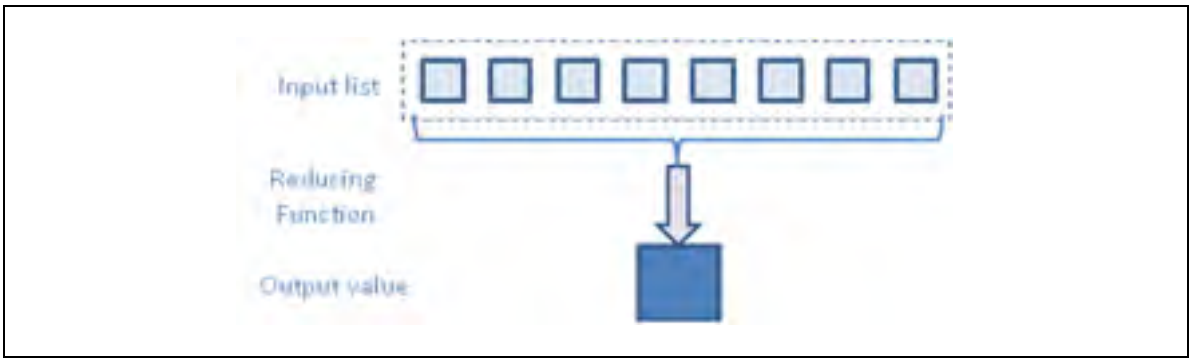


Figure 2.13 Reducing stage over input values

MapReduce inputs typically come from input files stored in a HDFS cluster. These files are distributed across all commodity hardware that is running HDFS data nodes. According to Yahoo (Yahoo! 2012), when a mapping stage is started any mapper (node) can process any input file or part of an input file. In this form, each mapper loads the set of local files to be able to processes them.

When a mapping phase has been completed, an intermediate pair of values (key, value) must be exchanged between machines to send all values with the same key to a single reducer. Like map tasks, reduce tasks are spread across the same nodes in the cluster and do not exchange information with one another, nor are they aware of one another´s existence. Thus, all data transfer is handled by the Hadoop MapReduce platform itself, guided implicitly by the different keys associated with values. Figure 2.14 shows a high-level data flow into the MapReduce tasks.

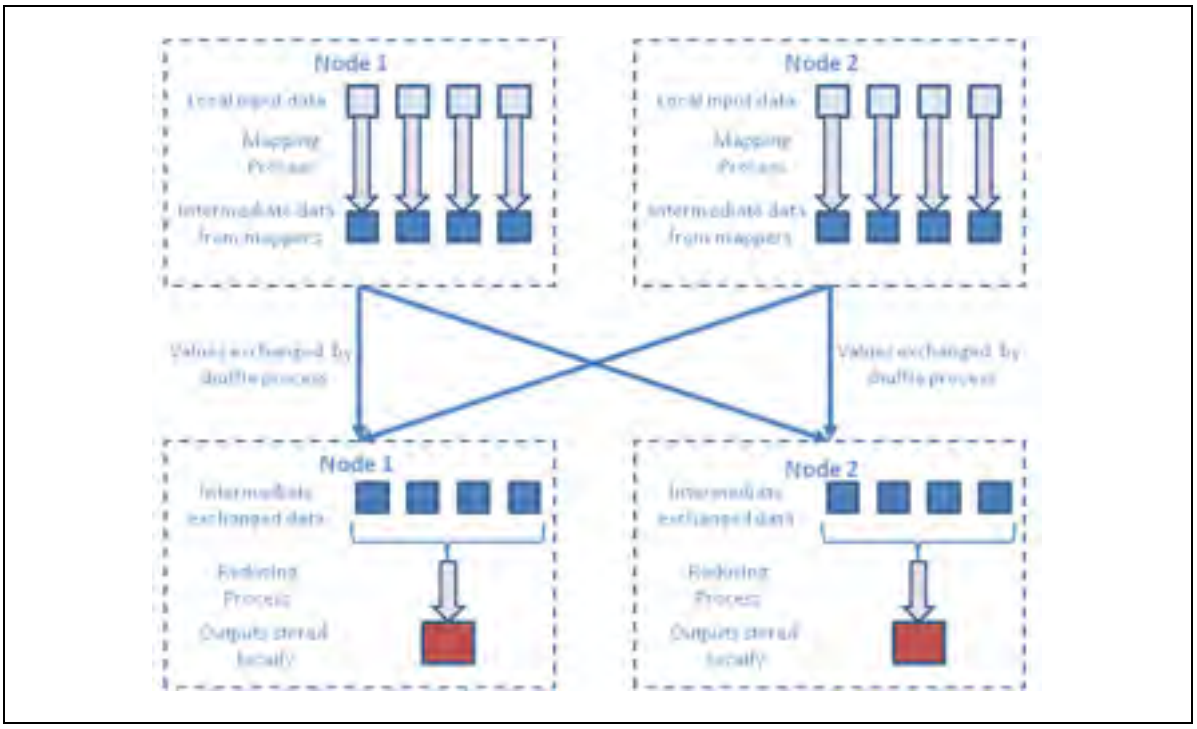


Figure 2.14 High-level data flow into the MapReduce tasks

2.5.4.1 MapReduce execution phases

According to Dean (Dean and Ghemawat 2008), Map invocations are distributed across multiple machines by automatically partitioning the input data into a set of M splits. The figure 2.15 presents the sequence of actions that occur during a MapReduce application execution.

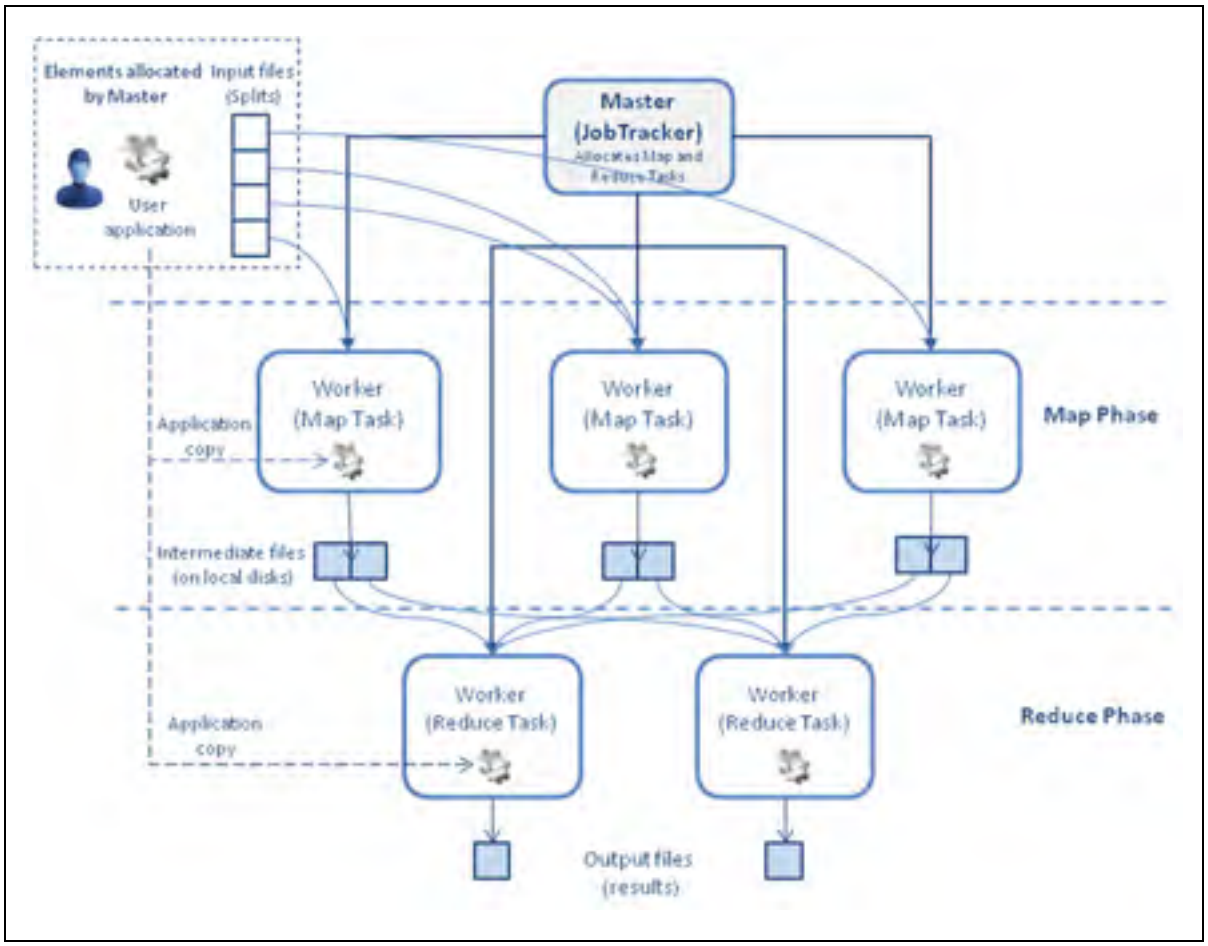


Figure 2.15 Actions that occur during a MapReduce application execution

Dean mentions that when a user program calls the MapRecue function, the following sequence of actions occurs into a MapRecduce cluster:

1 The MapReduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines.

2 One of the copies is special – the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assign each one a map task or a reduce task.

3 A worker who is assigned a map task reads the content of the corresponding input split. It parses key/values pairs out of the input data and passes each pair to the user-defined Map function. The intermediate key/value pairs produced by the Map function are buffered in memory.

4 Periodically, the buffered pairs are written to local disk, partitioned into R regions by the partitioning function. The locations of these buffered pairs on the local disk are passed back to the master, who is responsible for forwarding these locations to the reduce workers.

5 When a reduce worker is notified by the master about these locations, it uses remote procedure calls to read the buffered data from the local disks of the map workers. When a reduce worker has read all intermediate data, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together. The sorting is needed because typically many different keys map to the same reduce task. If the amount of intermediate data is too large to fit in memory, an external sort is used.

6 The reduce worker iterates over the sorted intermediate data and for each unique intermediate key encountered, it passes the key and the corresponding set of intermediate values to the user's Reduce function. The output of the Reduce function is appended to a final output file for its reduce partition.

7 When all map tasks and reduce tasks have been completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

This section concludes the literature review process which covers the topics that are used in the research. The next chapter presents the core of this thesis which develops the Performance measurement model for cloud computing applications (PMMoCCA).

CHAPTER 3

A PERFORMANCE MEASUREMENT MODEL FOR CLOUD COMPUTING APPLICATIONS

This chapter presents the design of the Performance Measurement Model for Cloud Computing Applications (PMMoCCA). Section 3.1 introduces to the performance measurement framework for CC (PMFCC) which defines the components involved in the process of measurement of CCA. The framework presented in section 3.1 uses quality concepts that are related with performance from an international standard point of view such as ISO 25010. Section 3.2 presents the process of design to develop the PMMoCCA which uses two techniques to find out the key performance concepts that best represent the performance of a CCA. In addition, this section proposes a method to determine the relationships among the many CCA performance measures and the key performance concepts. This method is based on the Taguchi method, for the design of experiments, which helps to identify the relationships between the various performance measures and the performance concepts defined in the PMFCC. Section 3.3 presents an experiment which describes the procedure to carry out a performance measurement of a CCA. Section 3.4 presents the results and their interpretation by means of techniques of statistical analysis. Finally, Section 3.5 presents a summary of the experiment defining the relationship between the analysis results and the performance concepts defined in the PMMoCCA.

3.1 Performance measurement framework for cloud computing (PMFCC)

This section presents the concepts, sub concepts and relationships used in the design of a performance measurement framework for Cloud Computing Applications. This framework defines the components involved in the performance measurement process of CCA using software quality concepts. The design of this framework is based on the concepts of metrology, along with aspects of software quality directly related to the performance concept which are addressed in the ISO 25010. According to Abran, metrology is the foundation for the development and use of measurement instruments and measurement processes (Abran

2010). In the literature the performance efficiency and reliability concepts are closely associated with the measurement perspective of Jain (Jain 1991) and, as a result, this framework integrates ISO 25010 concepts into Jain´s perspective for the performance measurement of CCS and CCA.

3.1.1 Performance Concepts as software system requirements

The ISO 25030 (ISO/IEC 2006) defines system quality requirements and states that software systems have a variety of stakeholders who may have an interest in the software system throughout its life cycle. Stakeholders include end users, organizations, developers, maintainers, etc., who have a legitimate interest in the software system. Each stakeholder has different need and expectation of the software system, and these may evolve during the software systems life cycle. Stakeholder needs can be either explicitly stated or implied, and often they are unclear. Performance requirements need to be established and should be expressed in order to ensure that a specific software system will be able to perform an efficient and reliable service under stated conditions and meet the end user need and expectations. ISO 19759 – Guide to the Software Engineering Body of Knowledge (SWEBOK) (ISO/IEC 2005) defines a requirement as a property that must be exhibited in order to solve real-world problems.

According to ISO 25030, stakeholders' needs and expectations can be identified through requirements, and can be transformed into technical views of software system requirements through a design process that can be used to realize the intended software system. Technical views of user requirements are often called system requirements. These should state which characteristics the system is to have, and be verifiable, in order to satisfy the stakeholder's user requirements, which are defined as perceived needs.

ISO 25030 proposes that a system consists of a number of interacting elements that can be defined and categorized in different ways, and system requirements can, for example, include

requirements for software, computer hardware, mechanical systems, and so on. Section 3.1.2 presents the system requirements that are involved in the analysis of CCA performance.

3.1.2 Definition of system performance concepts

A well known perspective for system performance measurement was proposed by Jain (Jain 1991) who maintains that a performance study must first establish a set of performance criteria (or characteristics) to help to carry out the system measurement process. He notes that system performance is typically measured using three sub concepts, if it is performing a service correctly: 1) responsiveness, 2) productivity, and 3) utilization, and proposes a measurement process for each. In addition, Jain notes that there are several possible outcomes for each service request made to a system, which can be classified into three categories. The system may: 1) perform the service correctly, 2) perform the service incorrectly, or 3) refuse to perform the service altogether. Moreover, he defines three sub concepts associated with each of these possible outcomes which affect system performance: 1) speed, 2) reliability, and 3) availability. Figure 3.1 presents the possible outcomes of a service request to a system and the sub concepts associated with them.

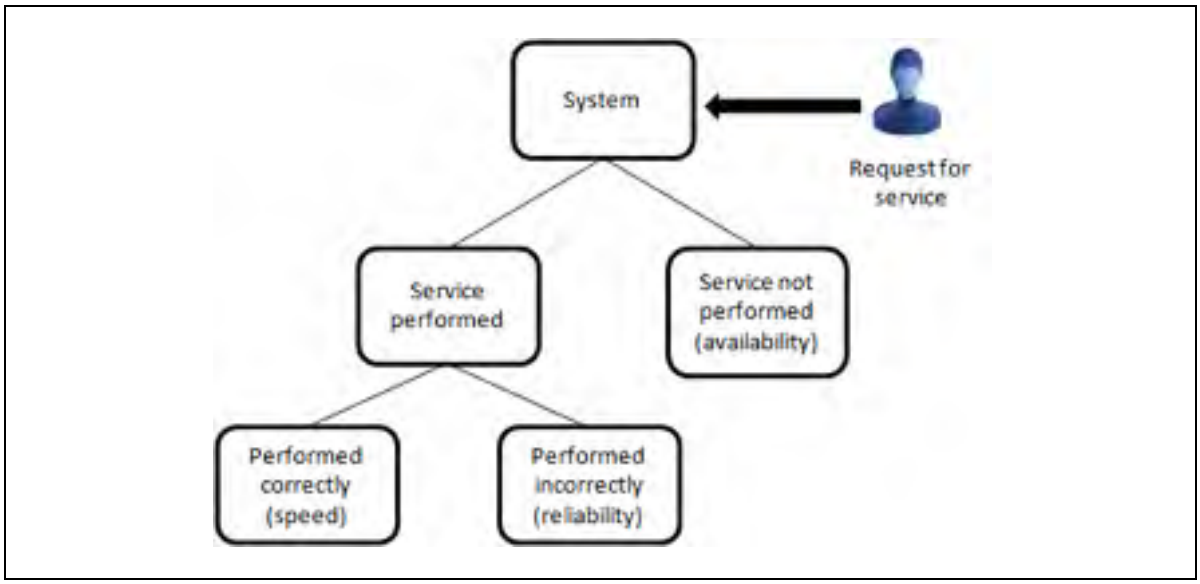


Figure 3.1 Possible outcomes of a service request to a system, according to Jain

3.1.3 Definition of the performance concept for cloud computing application

The ISO 25010 (ISO/IEC 2011) defines software product and computer system quality from two distinct perspectives: 1) a quality in use model, and 2) a product quality model:

1. The quality in use model is composed of five characteristics that relate to the outcome of an interaction when a product is used in a particular context of use. This quality model is applicable to the entire range of use of the human-computer system, including both systems and software.
2. The product quality model is composed of eight characteristics that relate to the static properties of software and the dynamic properties of the computer system.

This product quality model is applicable to both systems and software. According to ISO 25010, the properties of both determine the quality of the product in a particular context, based on user requirements. For example, *performance efficiency and reliability* can be specific concerns of users who specialize in areas of content delivery, management, or maintenance. The performance efficiency concept proposed in ISO 25010 has three sub concepts: 1) time behavior, 2) resource utilization, and 3) capacity, while the reliability concept has four sub concepts: 1) maturity, 2) availability, 3) fault tolerance, and 4) recoverability. This thesis selects the concepts of *performance efficiency and reliability* as baseline for determining the performance of cloud computing applications (CCA).

Based on the performance perspectives presented by Jain and the product quality characteristics defined by ISO 25010, we propose the following definition of cloud computing application performance measurement:

"The performance of a Cloud Computing Application is determined by an analysis of the characteristics involved in performing an efficient and reliable service that meets requirements under stated conditions and within the maximum limits of the system parameters."

Although at first sight this definition may seem complex, it only includes the sub concepts necessary to carry out cloud computing application performance analysis. Furthermore, from the literature review, a number of sub concepts have been identified that could be directly related to the concept of performance, such as:

- Performance efficiency: The amount of resources used under stated conditions. Resources can include software products, the software and hardware configuration of the system, and materials.
- Time behavior: The degree to which the response and processing times and the throughput rates of a product or system, when performing its functions, meet requirements.
- Capacity: The degree to which the maximum limits of a product or system parameter meet requirements.
- Resource utilization: The degree to which the amounts and types of resources used by a product or system when performing its functions meet requirements.
- Reliability: The degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time.
- Maturity: The degree to which a system meets needs for reliability under normal operation.
- Availability: The degree to which a system, product or component is operational and accessible when required for use.
- Fault tolerance: The degree to which a system, product, or component operates as intended, in spite of the presence of hardware or software faults.
- Recoverability: The degree to which a product or system can recover data directly affected in the event of an interruption or a failure, and be restored to the desired state.

3.1.4 Relationship between performance measurement concepts and sub concepts

Now that the performance measurement concepts and sub concepts have been introduced, a relationship model will be helpful to show the relationship between the performance concepts proposed by ISO 25010 and the performance measurement perspective presented by Jain. In addition, this model shows the logical sequence in which the concepts and sub concepts appear when a performance issue arises in a CCA (see figure 3.1).

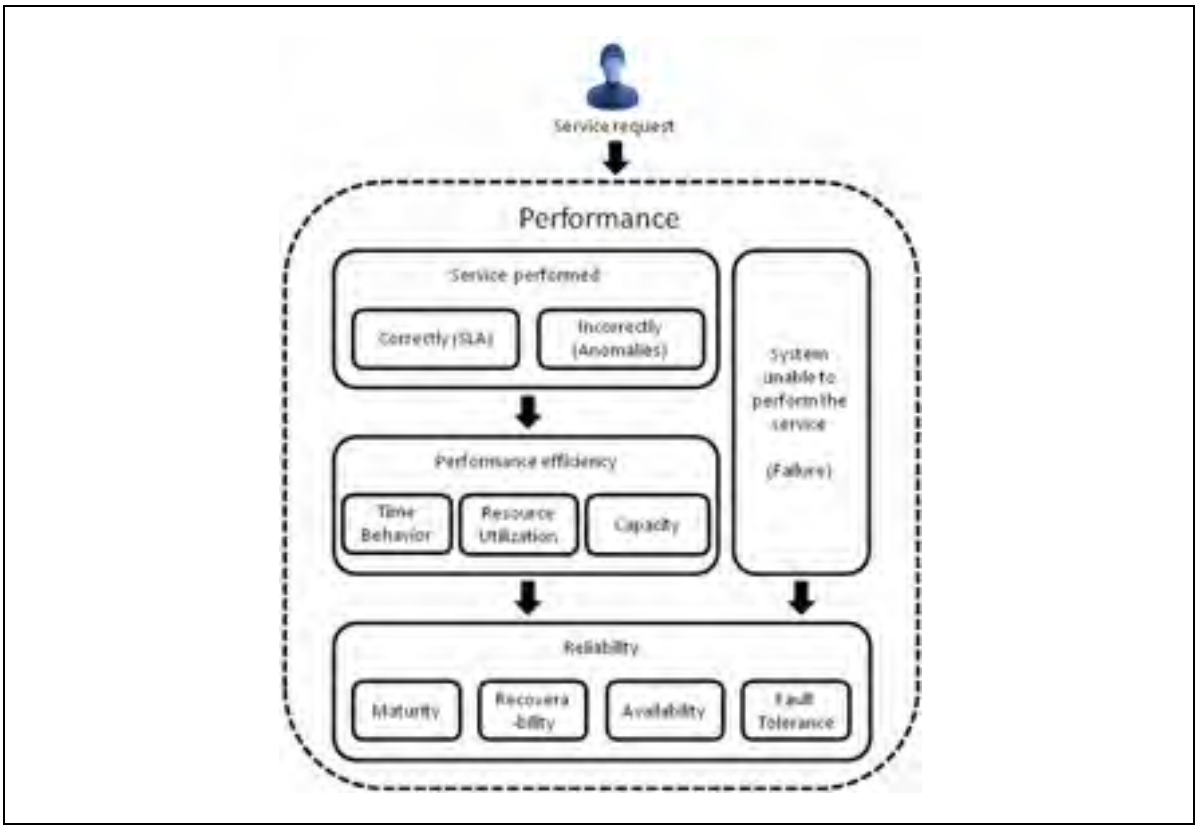


Figure 3.2 Model of the relationships between performance concepts and sub concepts

In figure 3.2, system performance is determined by two main sub concepts: 1) performance efficiency, and 2) reliability. We have seen that when a CCA receives a service request, there are three possible outcomes (the service is performed correctly, the service is performed incorrectly, or the service cannot be performed). The outcome will determine the sub concepts that will be applied for performance measurement. For example, suppose that the

CCS performs a service correctly, but, during its execution, the service failed and was later reinstated. Although the service was ultimately performed successfully, it is clear that the system availability (part of the reliability sub concept) was compromised, and this affected CCS performance.

3.1.5 The performance measurement framework for cloud computing (PMFCC)

The foundation for the proposal of a performance measurement model for cloud computing application is based on the performance measurement framework for cloud computing (PMFCC) which is shown in figure 3.2. The performance measurement framework defines the base measures related to the performance concepts that represent the system attributes, and which can be measured to assess whether or not the CCA satisfies the stated requirements from a quantitative viewpoint. These base measures have been adapted from ISO 25023 – Measurement of system and software product quality which provides measures, including associated measurement methods and quality measure elements for the quality characteristics in a product quality model (ISO/IEC 2013). These base measures are grouped into collection functions, which are responsible for conducting the measurement process using a combination of base measures through a data collector. They are associated with the corresponding ISO 25010 quality derived measures, as presented in Table 3.1

Table 3.1 Functions associated with Cloud Computing performance concepts

Base Measures	Collection Functions for Measures	ISO 25010 Quality Characteristics
Failures avoided Failures detected Failures predicted Failures resolved	Failure function	Maturity Resource utilization Fault tolerance
Breakdowns Faults corrected Faults detected Faults predicted	Fault function	Maturity Fault tolerance
Tasks entered into recovery Tasks executed Tasks passed Tasks restarted Tasks restored Tasks successfully restored	Task function	Availability Capacity Maturity Fault tolerance Resource utilization Time behavior
Continuous resource utilization time Down time Maximum response time Observation time Operation time Recovery time Repair time Response time Task time Time I/O devices occupied Transmission response time Turnaround time	Time function	Availability Capacity Maturity Recoverability Resource utilization Time behavior
Transmission errors Transmission capacity Transmission ratio	Transmission function	Availability Capacity Maturity Recoverability Resource utilization Time behavior

The base measures presented in Table 3.1 are categorized as collection functions in the PMFCC (see figure 3.3); These collection functions were designed to be interconnected through an intermediate service (IS) that shares intermediate results from common base

measures, reducing the number of operations in the measurement process at the time of calculation.

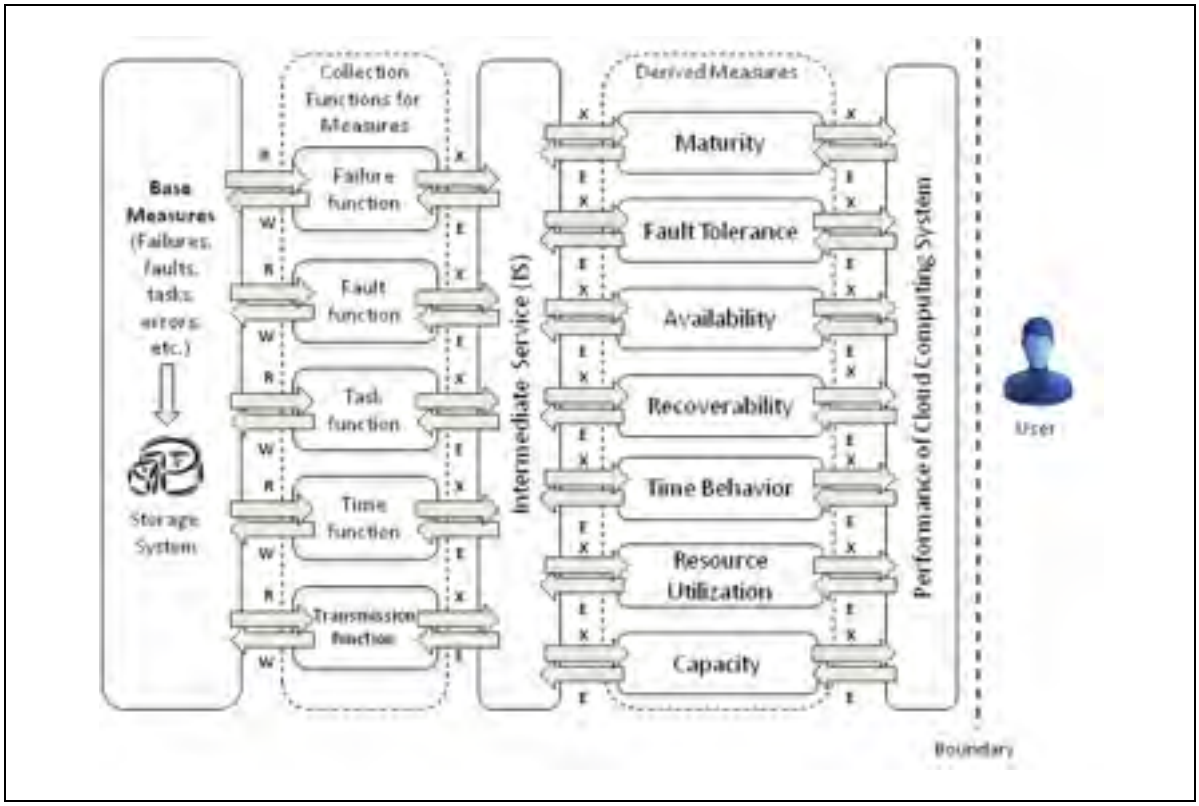


Figure 3.3 Performance measurement framework for Cloud Computing

This framework determines how to measure a quality characteristic: for example, how can be measured the CCS availability characteristic (presented in Table 3.1) using the PMFCC? To start with, three collection functions are needed: 1) the time function, 2) the task function, and 3) the transmission function. The time function can use several different measured attributes, such as CPU utilization by the user, job duration, and response time. These measures are obtained using a data collector, and then inputted to a time function that calculates a derived measure of the time concept. The IS combines the results of each function to determine a derived measure of the availability that contributes to CCS performance, as defined in the framework.

3.2 Performance measurement model for cloud computing applications (PMMoCCA)

Performance analysis models for CCA serve as a baseline for detecting and predicting possible anomalies in the cloud computing software that may impact CCS. To be able to design such PMMoCCA, methods are needed to collect the necessary base measures specific to performance, and the PMFCC is used to determine the relationships that exist among these measures. One of the challenges in designing PMMoCCA is how to determine what types of relationships exist between the various base measures and the performance quality concepts defined in international standards such as ISO 25010: Systems and software product Quality Requirements and Evaluation (SQuaRE), System and software quality models. For example, what are the relationships between the amounts of physical memory used by a CCA and performance concepts such as resource utilization or capacity? This section proposes the use of statistical methods to determine how closely performance parameters (base measures) are related with software engineering performance concepts.

3.2.1 Relationship between measures of cloud computing applications and performance concepts

In order to determine the degree of relationship between performance measures extracted from CCA, and performance concepts and sub concepts defined in the PMFCC (Figure 3.3), first it is necessary to map performance measures from the CCA onto the performance quality concepts previously defined. For this, measures need to be collected by means of extracted data from CCA log files and system monitoring tools (see Table 3.2). This data is obtained from a Hadoop cluster system in which measures are generated and stored (see ANNEX I for a complete list of performance measures).

Table 3.2 Extract of collected performance measures from CCA.

Measure	Source	Description
jobs:clusterMapCapacity	Jobs of CCA	Maximum number of available maps to be created by a job
jobs:clusterReduceCapacity	Jobs of CCA	Maximum number of available reduces to be created by a job
jobs:finishTime	Jobs of CCA	Time at which a job was completed
jobs:JobSetupTaskLaunchTime	Jobs of CCA	Time at which a job is setup in the cluster for processing
jobs:jobId	Jobs of CCA	Job ID
jobs:launchTime	Jobs of CCA	Time at which a job is launched for processing
jobs:Status	Jobs of CCA	Job status after processing (Successful or Failed)
jobs:submitTime	Jobs of CCA	Time at which a job was submitted for processing
disk:ReadBytes	CC System	Amount of HD bytes read by a job
disk:WriteBytes	CC System	Amount of HD bytes written by a job
memory:Free	CC System	Amount of average free memory on a specific time
memory:Used	CC System	Amount of average memory used on a specific time
network:RxBytes	CC System	Amount of network bytes received on a specific time
network:RxErrors	CC System	Amount of network errors during received transmission on a specific time
network:TxBytes	CC System	A mount of network bytes transmitted on a specific time
network:TxErrors	CC System	Amount of network errors during transmission on a specific time

Once the performance measures are collected, they are mapped onto the performance concepts defined in the PMFCC by means of the formulae defined in the ISO 25023 (ISO/IEC 2013). It is important to mention that such formulae were adapted according to the different performance measures collected from the CCA system in order to represent the different concepts in a coherent form. Table 3.3 presents the different CCA performance measures after being mapped onto the PMFCC concepts and sub concepts.

Table 3.3 CCA performance measures mapped onto PMFCC concepts and sub concepts.

PMFCC concept	PMFCC sub concepts	Description	Adapted formula
Performance efficiency			
Time behavior	Response time	Duration from a submitted CCA Job to start processing till it is launched	submitTime - launchTime
Time behavior	Turnaround time	Duration from a submitted CCA Job to start processing till completion of the Job	finishTime – submitTime
Time behavior	Processing time	Duration from a launched CCA Job to start processing till completion of the Job	finishTime-launchTime
Resource utilization	CPU utilization	How much CPU time is used per minute to process a CCA Job (percent)	100 – cpuIdlePercent
Resource utilization	Memory utilization	How much memory is used to process a CCA Job per minute (percent)	100 – memoryFreePercent
Resource utilization	Hard disk bytes read	How much bytes are read to process a CCA Job per minute	Total of bytes read per minute
Resource utilization	Hard disk bytes written	How much bytes are written to process a CCA Job per minute	Total of bytes written per minute
Capacity	Load map tasks capacity	How many map tasks are processed in parallel for a specific CCA Job	Total of map tasks processed in parallel for a specific CCA Job
Capacity	Load reduce tasks capacity	How many reduce tasks are processed in parallel for a specific CCA Job	Total of reduce tasks processed in parallel for a specific CCA Job
Capacity	Network Tx bytes	How many bytes are transferred while a specific CCA Job is processed	Total of transferred bytes per minute
Capacity	Network Rx bytes	How many bytes are received while a specific CCA Job is processed	Total of received bytes per minute
Reliability			
Maturity	Task mean time between	How frequently does a task of a specific CCA Job fail in operation	Number of tasks failed per minute

	failure		
Maturity	Tx network errors	How many transfer errors in the network are detected while processing a specific CCA Job	Number of Tx network errors detected per minute
Maturity	Rx network errors	How many reception errors in the network are detected while processing a specific CCA Job	Number of Rx network errors detected per minute
Availability	Time of CC System Up	Total time that the system has been in operation	Total minutes of the CC system operation
Fault tolerance	Network Tx collisions	How many transfer collision in the network occurs while processing a specific CCA Job	Total of Tx network collisions per minute
Fault tolerance	Network Rx dropped	How many reception bytes in the network are dropped while processing a specific CCA Job	Total of Rx network bytes are dropped per minute
Recoverability	Mean recovery time	What is the average time the CC system take to complete recovery from a failure	Average recovery time of CC system

3.2.2 Selection of key PMFCC concepts to represent the performance of CCA

Once the performance measures extracted from the CCA system mapped onto the performance quality concepts (see Table 3.3), the next step is to select a set of key sub concepts of PMFCC that best represent the performance of CCA. For this, two techniques for feature selection are used in order to determine the most relevant features (PMFCC sub concepts) from a data set. According to Kantardzic (Kantardzic 2011), feature selection is a set of techniques that select relevant features (PMFCC sub concepts) for building robust learning models by removing most irrelevant and redundant features from the data. Kantardzic establishes that feature selection algorithms typically fall into two categories: feature ranking and subset selection. Feature ranking ranks all features by a specific base measure and eliminates all features that do not achieve an adequate score while subset selection, searches the set of all features for the optimal subset in which selected features are not ranked. The next subsections present two techniques of feature ranking which are used in the PMMoCCA in order to determine the most relevant performance sub concepts (features) that best represent the performance of CCA.

3.2.2.1 Feature selection based on comparison of means and variances

The feature selection based on comparison of means and variances is based on the distribution of values for a given feature, in which it is necessary to compute the mean value and the corresponding variance. In general, if one feature describes different classes of entities, samples of two different classes can be examined. The means of feature values are normalized by their variances and then compared. If the means are far apart, interest in a feature increases: it has potential, in terms of its use in distinguishing between two classes. If the means are indistinguishable, interest wanes in that feature. The mean of a feature is compared in both cases without taking into consideration relationship to other features. The next equations formalize the test, where A and B are sets of feature values measured for two different classes, and n1 and n2 are the corresponding number of samples:

$$SE(A-B)=\sqrt{\left(\frac{\mathrm{var}(A)}{n_1}+\frac{\mathrm{var}(B)}{n_2}\right)} \tag{1}$$

$$Test:\frac{|mean(A)-mean(B)|}{SE(A-B)}>threshold\_value \tag{2}$$

In this approach to feature selection, it is assumed that a given feature is independent of others. A comparison of means is typically a natural fit to classification problems. For ***k*** classes, ***k*** pair wise comparisons can be made, comparing each class with its complement. A feature is retained if it is significant for any of the pair wise comparisons as shown in formula 2.

3.2.2.2 Relief algorithm

Another important technique for feature selection is the Relief algorithm. The Relief algorithm is a feature weight-based algorithm which relies on relevance evaluation of each feature given in a training data set in which samples are labeled (classification problems).

The main concept of this algorithm is to compute a ranking score for every feature indicating how well this feature separates neighboring samples. The authors of the Relief algorithm, Kira and Rendell (Kira and Rendell 1992), proved that ranking score becomes large for relevant features and small for irrelevant ones.

The objective of the relief algorithm is to estimate the quality of features according to how well their values distinguish between samples close to each other. Given a training data ***S***, the algorithm randomly selects subset of samples size ***m***, where ***m*** is a user defined parameter. The algorithm analyses each feature based on a selected subset of samples. For each randomly selected sample ***X*** from a training data set, it searches for its two nearest neighbors: one from the same class, called nearest hit ***H***, and the other one from a different class, called nearest miss ***M***.

The Relief algorithm updates the quality score ***W*(*Ai*)** for all feature ***Ai*** depending on the differences on their values for samples ***X***, ***M***, and ***H*** as shown in formula 3.

$$W_{new}(A_i) = \frac{W_{old}(A_i) - (diff(X[A_i], H[A_i])^2 + diff(X[A_i], M[A_i])^2)}{m} \quad (3)$$

The process is repeated ***m*** times for randomly selected samples from the training data set and the scores ***W*(*Ai*)** are accumulated for each sample. Finally, using threshold of relevancy τ, the algorithm detects those features that are statistically relevant to the target classification, and these are the features with ***W*(*Ai*)** $\geq \tau$. The main steps of the Relief algorithm are formalized in Algorithm 1.

Algorithm 1 Relief Algorithm

Initialize $W(A_j) = 0$; $i = 1, 2. \ldots, n$ (where n is the number of features)

For $i = 1$ *to* m

Randomly select X from training data set S

Find nearest hit H and nearest miss M samples

For $j = 1$ *to* n

$$W(A_j) = W(A_j) - (\text{diff}(X[A_j], H[A_j])^2 + \text{diff}(X[A_j], M[A_j])^2)/m$$

End

End

Output: Subset of feature where $W(A_j) \geq \tau$

3.2.3 Choosing a methodology to analyze relationships between performance concepts

Once that a subset of the most important features (key performance sub concepts) has been selected, the next step is to determine the degree of relationship that exist between such subset of features and the rest of performance sub concepts defined by means of PMFCC. For this, the use of Taguchi's experimental design method is proposed: it investigates how different features (performance measures) are related, and to what degree. Understanding these relationships will enable us to determine the influence each of them has in the resulting performance concepts. The PMFCC shows many of the relationships that exist between the base measures which have a major influence on the collection functions. However, in CCA and more specifically in the Hadoop MapReduce application case study, there are over a hundred possible performance measures (including system measures) which could contribute to the analysis of CCA performance. A selection of these performance measures has to be included in the collection functions so that the respective performance concepts can be obtained and, from there, an indication of the performance of the applications. One key design problem is to establish which performance measures are interrelated and how much they contribute to each of the collection functions.

In traditional statistical methods, thirty or more observations (or data points) are typically needed for each variable, in order to gain meaningful insights and analyze the results. In addition, only a few independent variables are necessary to carry out experiments to uncover potential relationships, and this must be performed under certain predetermined and controlled test conditions. However, this approach is not appropriate here, owing to the large number of variables involved and the considerable time and effort required. Consequently, an analysis method that is suited to our specific problem and in our study area is needed.

A possible candidate method to address this problem is Taguchi's experimental design method, which investigates how different variables affect the mean and variance of a process performance characteristics, and helps in determining how well the process is functioning. This Taguchi method proposes a limited number of experiments, but is more efficient than a factorial design in its ability to identify relationships and dependencies. The next section presents the method to find out the relationships.

3.2.3.1 Taguchi method of experimental design

Taguchi's Quality Engineering Handbook (Taguchi, Chowdhury et al. 2005) describes the Taguchi method of experimental design which was developed by Dr. Genichi Taguchi, a researcher at the Electronic Control Laboratory in Japan. This method combines industrial and statistical experience, and offers a means for improving the quality of manufactured products. It is based on a 'robust design' concept, according to which a well designed product should cause no problem when used under specified conditions.

According to Cheikhi (Cheikhi and Abran 2012), Taguchi's two phase quality strategy is the following:

- Phase 1: The online phase, which focuses on the techniques and methods used to control quality during the production of the product.

- Phase 2: The offline phase, which focuses on taking those techniques and methods into account before manufacturing the product, that is, during the design phase, the development phase, etc.

One of the most important activities in the offline phase of the strategy is parameter design. This is where the parameters are determined that makes it possible to satisfy the set quality objectives (often called the objective function) through the use of experimental designs under set conditions. If the product does not work properly (does not fulfill the objective function), then the design constants (also called parameters) need to be adjusted so that it will perform better. Cheikhi explains that this activity includes five (5) steps, which are required to determine the parameters that satisfy the quality objectives:

1. Definition of the objective of the study, that is, identification of the quality characteristics to be observed in the output (results expected).
2. Identification of the study factors and their interactions, as well as the levels at which they will be set. There are two different types of factors: 1) control factors: factors that can be easily managed or adjusted; and 2) noise factors: factors that are difficult to control or manage.
3. Selection of the appropriate orthogonal arrays (OA) for the study, based on the number of factors, and their levels and interactions. The OA show the various experiments that will need to be conducted in order to verify the effect of the factors studied on the quality characteristic to be observed in the output.
4. Preparation and performance of the resulting OA experiments, including preparation of the data sheets for each OA experiment according to the combination of the levels and factors for the experiment. For each experiment, a number of trials are conducted and the quality characteristics of the output are observed.
5. Analysis and interpretation of the experimental results to determine the optimum settings for the control factors, and the influence of those factors on the quality characteristics observed in the output.

According to Taguchi's Quality Engineering Handbook (Taguchi, Chowdhury *et al.* 2005), the OA organizes the parameters affecting the process and the levels at which they should vary. Taguchi's method tests pairs of combinations, instead of having to test all possible combinations (as in a factorial experimental design). This approach can determine which factors affect product quality the most in a minimum number of experiments.

Taguchi's OA arrays can be created manually or they can be derived from deterministic algorithms. They are selected by the number of parameters (variables) and the number of levels (states). An OA array is represented by Ln and Pn, where Ln corresponds to the number of experiments to be conducted, and Pn corresponds to the number of parameters to be analyzed. Table 3.4 presents an example of Taguchi OA L12, meaning that 12 experiments are conducted to analyze 11 parameters.

Table 3.4 Taguchi´s Orthogonal Array L12

No. of Experiments (L)	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	2	2	2	2	2	2
3	1	1	2	2	2	1	1	1	2	2	2
4	1	2	1	2	2	1	2	2	1	1	2
5	1	2	2	1	2	2	1	2	1	2	1
6	1	2	2	1	2	2	1	2	1	2	1
7	1	2	2	2	1	2	2	1	2	1	1
8	2	1	2	1	2	2	2	1	1	1	2
9	2	1	1	2	2	2	1	2	2	1	1
10	2	2	2	1	1	1	1	2	2	1	2
11	2	2	1	2	1	2	1	1	1	2	2
12	2	2	1	1	2	1	2	1	2	2	1

An OA cell contains the factor levels (1 and 2), which determine the type of parameter values for each experiment. Once the experimental design has been determined and the trials have been carried out, the performance characteristic measurements from each trial can be used to analyze the relative effect of the various parameters.

Taguchi´s method is based on the use of the signal-to-noise ratio (SNR). The SNR is a measurement scale that has been used in the communications industry for nearly a century for determining the extent of the relationship between quality factors in a measurement model (Taguchi, Chowdhury et al. 2005). The SNR approach involves the analysis of data for variability in which an input-to-output relationship is studied in the measurement system. Thus, to determine the effect each parameter has on the output, the SNR is calculated by the follow formula:

$$SN_i = 10\log\frac{\bar{y}_i^2}{s_i^2} \tag{4}$$

where

$$\bar{y}_i = \frac{1}{N_i}\sum_{u=1}^{N_i} y_{i,u}$$

$$s_i^2 = \frac{1}{N_i - 1}\sum_{u=1}^{N_i}\left(y_{i,u} - \bar{y}_i\right)$$

i*=Experiment number*

u*=Trial number*

N_i*=Number of trials for experiment* ***i***

To minimize the performance characteristic (objective function), the following definition of the SNR should be calculated:

$$SN_i = -10\log\left(\sum_{u=1}^{Ni}\frac{y_u^2}{N_i}\right) \tag{5}$$

To maximize the performance characteristic (objective function), the following definition of the SNR should be calculated:

$$SN_i = -10\log\left[\frac{1}{N_i}\sum_{u=1}^{N_i}\frac{1}{y_u^2}\right] \tag{6}$$

Once the SNR values have been calculated for each factor and level, they are tabulated as shown in Table 3.5, and then the range R (R = high SN - low SN) of the SNR for each parameter is calculated and entered on Table 3.5

Table 3.5 Rank for SNR values

Level	P1	P2	P3	P4	P5	P6	P7	…	P11
1	$SN_{1,1}$	$SN_{2,1}$	$SN_{3,1}$	$SN_{4,1}$	$SN_{5,1}$	$SN_{6,1}$	$SN_{7,1}$		$SN_{11,1}$
2	$SN_{1,2}$	$SN_{2,2}$	$SN_{3,2}$	$SN_{4,2}$	$SN_{5,2}$	$SN_{6,2}$	$SN_{7,2}$	…	$SN_{11,2}$
3	$SN_{1,3}$	$SN_{2,3}$	$SN_{3,3}$	$SN_{4,3}$	$SN_{5,3}$	$SN_{6,3}$	$SN_{7,3}$	…	$SN_{11,3}$
4	$SN_{1,4}$	$SN_{2,4}$	$SN_{3,4}$	$SN_{4,4}$	$SN_{5,4}$	$SN_{6,4}$	$SN_{7,4}$	…	$SN_{11,4}$
Range	R_{P1}	R_{P2}	R_{P3}	R_{P4}	R_{P5}	R_{P6}	R_{P7}	…	R_{P11}
Rank	**$Rank_{P1}$**	**$Rank_{P2}$**	**$Rank_{P3}$**	**$Rank_{P4}$**	**$Rank_{P5}$**	**$Rank_{P6}$**	**$Rank_{P7}$**	**…**	**$Rank_{P11}$**

According to Taguchi's method, the larger the R value for a parameter, the greater its effect on the process.

3.3 Experiment

3.3.1 Experiment setup

The experiment was conducted on a DELL Studio Workstation XPS 9100 with Intel Core i7 12-core X980 processor at 3.3 GHz, 24 GB DDR3 RAM, Seagate 1.5 TB 7200 RPM SATA 3Gb/s disk, and 1 Gbps network connection. We used a Linux CentOS 6.4 64-bit distribution and Xen 4.2 as the hypervisor. This physical machine hosts five virtual machines (VM), each with a dual-core Intel i7 configuration, 4 GB RAM, 20 GB virtual storage, and a virtual network interface type. In addition, each VM executes the Apache Hadoop distribution version 1.0.4, which includes the Hadoop Distributed File System (HDFS) and MapReduce framework libraries, Apache Chukwa 0.5.0 as performance measures collector and Apache HBase 0.94.1 as performance measures repository. One of these VM is the master node, which executes NameNode (HDFS) and JobTracker (MapReduce), and the rest of the VM are slave nodes running DataNodes (HDFS) and JobTrackers (MapReduce). Figure 3.4 presents the cluster configuration for the set of experiments.

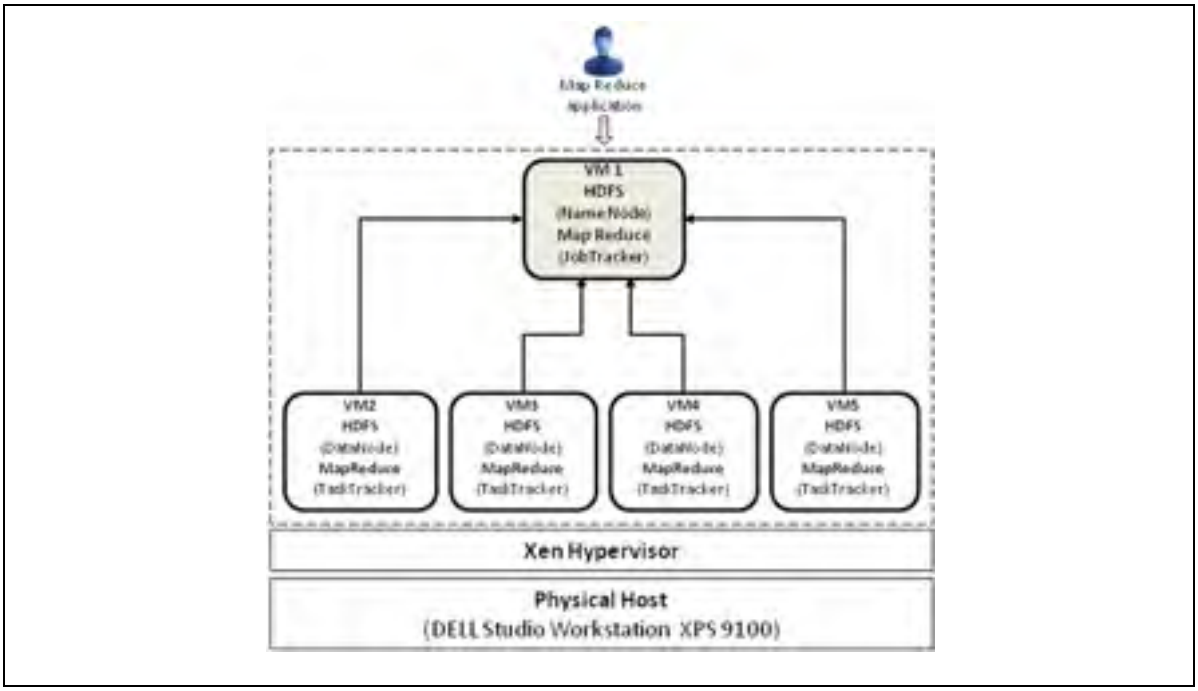


Figure 3.4 Cluster configuration for the experiment

3.3.2 Mapping of performance measures onto PMFCC concepts

A total of 103 MapReduce Jobs (CCA) were executed in the virtual Hadoop cluster and a set of performance measures were obtained from MapReduce Jobs logs and monitoring tools. One of the main problems that arose after the performance measures repository ingestion process was the cleanliness of data. Cleanliness calls for the quality of the data to be verified prior to performing data analysis. Among the most important data quality issues to consider during data cleaning in the model were corrupted records, inaccurate content, missing values, and formatting inconsistencies, to name a few. Consequently, one of the main challenges at the preprocessing stage was how to structure data in standard formats so that they can be analyzed more efficiently. For this, a data normalization process was carried out over the data set by means of the standard score technique (see formula 7).

$$Xnorm_i = \frac{X_i - \mu_i}{S_i} \tag{7}$$

where

$\boldsymbol{X_i}$=*Feature* ***i***

$\boldsymbol{\mu_i}$=*Average value of* ***Xi*** *in data set*

$\boldsymbol{S_i}$=*Range of feature* ***i*** *(Max*$\boldsymbol{X_i}$*-Min*$\boldsymbol{X_i}$*)*

The normalization process scaled the values between the range of [-1, 1] according to the different collected performance measures which are expressed in different units and dimensions. For example the measure processing time is expressed in minutes while the measure memory utilization is expressed in Mbytes. Table 3.6 presents an extract from the different collected performance measures after the process of normalization.

Table 3.6 Extract of collected performance measures after normalization process

Performance measure	138367812000-job_201311051347_00021	1384366260-job_201311131253_00019	1384801260-job_201311181318_000419
Time of CC System Up	-0.4534012681	-0.4158208360	0.1921547093
Load map tasks capacity	-0.0860196415	-0.0770106325	-0.0860196415
Load reduce tasks capacity	-0.0334295334	-0.0334295334	-0.0334295334
Network Rx bytes	-0.0647059274	0.4808087278	-0.0055927073
Network Tx bytes	-0.0779191010	0.3139488890	-0.0613171507
Network Rx dropped	0.0	0.0	0.0
Network Tx collisions	0.0	0.0	0.0
Rx network errors	0.0	0.0	0.0
Tx network errors	0.0	0.0	0.0
CPU utilization	-0.0950811052	0.5669416548	-0.0869983066
Hard disk bytes read	-0.0055644728	0.0196859057	-0.0076297598
Hard disk bytes written	-0.0386960610	0.2328110281	-0.0253053155
Memory utilization	0.1956635952	0.4244033618	-0.0341498692
Processing time	-0.1838906682	0.8143236713	0.0156797304
Response time	0.0791592524	0.1221040377	-0.1846444285
Turnaround time	-0.1838786629	0.8143213555	0.0156595689
Task MTBF	0.0	0.0	0.0
Mean recovery time	0.0	0.0	0.0
Job Status	1.0	0.0	1.0

Note: Table 3.6 shows that values related to network measures are equal to zero because the experiment is performed in a Hadoop virtual cluster. This means that real transmission over a physical network does not exist leaving out the possibility of errors. In addition, other measures such as mean time between failure and mean recovery time are also equal to zero because during the experiment duration Hadoop virtual cluster never failed.

3.3.3 Selection of key measures to represent the performance of CCA

One of the challenges in the design of the PMMoCCA is how to determine a set of key sub concepts which have more relevance in the performance compared to others. For this, the application of feature selection is used during the process for knowledge discovery. As previously mentioned, two techniques used for feature selection are: means and variances,

and the Relief algorithm. The means and variances approach assumes that the given features are independent of others. In the experiment a total of 103 Hadoop MapReduce Jobs were executed storing their performance measures. A MapReduce Job may belong to one of two classes according to its status; failed or successful (0 or 1) (see Table 3.6).

Thus, applying means and variances technique to the data set (see 3.2.2.1 section); the feature Job Status classifies each Job records into two classes 0 and 1. First, it is necessary to compute a mean value and variance for both classes and for each feature (PMFCC sub concept measure). It is important to note that test values will be compared with the highest set of values obtained after the ranking process (9.0) because this distinguished them from the rest of results. Results are shown in Table 3.7.

Table 3.7 Results of means and variances

Performance measures	Test values
MapReduceJob_ProcessingTime	**9.214837**
MapReduceJob_TurnAround	**9.214828**
SystemHDWriteBytes_Utilization	**8.176328**
SystemUpTime	7.923577
SystemLoadMapCapacity	6.613519
SystemNetworkTxBytes	6.165150
SystemNetworkRxBytes	5.930647
SystemCPU_Utilization	5.200704
SystemLoadReduceCapacity	5.163010
MapReduceJob_ResponseTime	5.129339
SystemMemory_Utilization	3.965617
SystemHDReadBytes_Utilization	0.075003
NetworkRxDropped	0.00
NetworkTxCollisions	0.00
NetworkRxErrors	0.00
NetworkTxErrors	0.00

The analysis shows that measures *job processing time and job turnaround* have the potential to be distinguishing features between the two classes because their means are far apart and interest in such measures increases, this means their test values are greater than 9.0. In

addition, it is important to mention that although between the second and third result (hard disk bytes written) there is a considerable difference; the latter is also selected in order to analyze its relationship with the rest of measures because it also has the potential, in terms of their use, to stand out from the rest of the measures and give more certainty to the analysis of relationships. Thus, the measures *job processing time, job turnaround and hard disk bytes written* are selected as candidates to represent the performance of the CCA in the Hadoop system.

In order to give more certainty to the above results, the Relief algorithm technique (see 3.2.2.2 section) was applied to the same data set. As previously mentioned, the core of Relief algorithm estimates the quality of features according to how well their values distinguish between samples (performance measures of MapReduce Job records) close to each other. Thus, after applying the Relief algorithm to the data set, results are presented in table 3.8 where the algorithm detects those features that are statistically relevant to the target classification which are measures with highest quality score (see section 3.2.2.2).

Table 3.8 Relief algorithm results

Performance measure	Quality score (W)
MapReduceJob_ProcessingTime	**0.74903**
MapReduceJob_TurnAround	**0.74802**
SystemHDWriteBytes_Utilization	**0.26229**
SystemUpTime	0.25861
SystemCPU_Utilization	0.08189
SystemLoadMapCapacity	0.07878
SystemMemory_Utilization	0.06528
SystemNetworkTxBytes	0.05916
MapReduceJob_ResponseTime	0.03573
SystemLoadReduceCapacity	0.03051
SystemNetworkRxBytes	0.02674
SystemHDReadBytes_Utilization	0.00187
NetworkRxDropped	0.00
NetworkTxCollisions	0.00
NetworkRxErrors	0.00
NetworkTxErrors	0.00

The Relief results show that the performance measures *job processing time and job turnaround*, have the highest quality scores (***W***) and also have the potential to be distinguishing features between the two classes. In this case the performance measure 'hard disk bytes written' is also selected by means of the same approach as in the means and variance analysis: in other words, this has in terms of their use to stand out from the rest of the measures and give more certainty to the analysis of relationships. Thus, the measures *job processing time, job turnaround and hard disk bytes written* are also selected as candidates to represent the performance of CCA in the Hadoop system.

The results show that Time behavior and Resource utilization (see Table 3.3) are the PMFCC concepts that best represent the performance of the CCA. The next step is to determine how the rest of performance measures are related and to what degree. Studying these relationships enables to assess the influence each of them has on the concepts that best represent the CCA performance in the experiment. For this, Taguchi's experimental design method is applied in order to determine how different performance measures are related.

3.3.4 Analysis of relationship between selected performance measures

Once that a set of performance measures are selected to represent the performance of CCA, it is necessary to determine the relationships that exist between them and the rest of the performance measures. These key measures are defined as quality objectives (objective functions) according to Taguchi´s terminology. According to Taguchi (Taguchi, Chowdhury et al. 2005), quality is often referred to as conformance to the operating specifications of a system. To him, the quality objective (or dependent variable) determines the ideal function of the output that the system should show. In our experiment, the observed dependent variables are the following:

- Job processing time,
- Job Turnaround and
- Hard disk bytes written

Each MapReduce Job record (Table 3.6) is selected as an experiment in which different values for each performance measure is recorded. In addition, different levels of each factor (see Table 3.4) are established as:

- Values less than zero, level 1.
- Values greater or equal to zero, level 2.

Table 3.9 presents a summary of the factors, levels, and values for this experiment.

Table 3.9 Experiment factors and levels

Factor number	Factor name	Level 1	Level 2
1	Time of CC system up	< 0.0	≥ 0.0
2	Load map tasks capacity	< 0.0	≥ 0.0
3	Load reduce tasks capacity	< 0.0	≥ 0.0
4	Network Rx bytes	< 0.0	≥ 0.0
5	Network Tx bytes	< 0.0	≥ 0.0
6	CPU utilization	< 0.0	≥ 0.0
7	Hard disk bytes read	< 0.0	≥ 0.0
8	Memory utilization	< 0.0	≥ 0.0
9	Response time	< 0.0	≥ 0.0

Note. The factor set consisting of the rest of performance measures after the key selection process. In addition, it is important to mention that it is feasible to have values less than 0.0; this means negative values because the experiment is performed after the normalization process.

Using Taguchi's experimental design method, selection of the appropriate OA is determined by the number of factors and levels to be examined. The resulting OA array for this case study is L12 (presented in Table 3.4). The assignment of the various factors and values of this OA array is shown in Table 3.10

Table 3.10 Matrix of experiments

Exper-iment	Time of system up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	Network Tx bytes	CPU utiliza-tion	HD bytes read	Memory utiliza-tion	Respon-se time
1	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0
2	< 0	< 0	< 0	< 0	< 0	≥ 0	≥ 0	≥ 0	≥ 0
3	< 0	< 0	≥ 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0
4	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0
5	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0
6	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0
7	< 0	≥ 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	≥ 0
8	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	≥ 0	< 0	< 0
9	≥ 0	< 0	< 0	≥ 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0
10	≥ 0	≥ 0	≥ 0	< 0	< 0	< 0	< 0	≥ 0	≥ 0
11	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	< 0	< 0
12	≥ 0	≥ 0	< 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0

Table 10 shows the set of experiments to be carried out with different values for each parameter selected. For example, experiment 3 involves values of time of system up fewer than 0, map task capacity fewer than 0, reduce task capacity greater than or equal to 0, network rx bytes greater than or equal to 0, and so on.

A total of approximately 1000 performance measures were extracted by selecting those that met the different combination of parameter values after the normalization process for each experiment. Only a set of 40 measures met the experiment requirements presented in Table 3.10. This set of 12 experiments was divided into three groups of twelve experiments each (called trials). An extract of the values and results of each experiment for the *processing time* output objective is presented in Table 3.11 (ANNEXES II, III and IV present complete tables for the output objectives of *job processing time, job turnaround and hard disk bytes written* respectively).

Table 3.11 Trials, experiments, and resulting values for *job processing time* output objective

Trial	Experi-ment	Time of System Up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	Network Tx bytes	CPU utiliza-tion	…	Job processing time
1	1	-0.44091	-0.08601	-0.03342	-0.04170	-0.08030	-0.00762	…	**-0.183902878**
1	2	-0.34488	-0.07100	-0.03342	-0.02022	-0.18002	0.16864	…	**-0.170883497**
1	3	-0.49721	-0.08601	0.79990	0.01329	0.02184	-0.03221	…	**-0.171468597**
1	4	-0.39277	0.01307	-0.03342	0.02418	0.08115	-0.02227	…	**-0.13252447**
…	…	…	…	…	…	…	…	…	**…**
2	1	-0.03195	-0.08601	-0.03342	-0.06311	-0.09345	-0.17198	…	**0.015597229**
2	2	-0.01590	-0.19624	-0.03342	-0.06880	-0.01529	0.06993	…	**0.730455521**
2	3	-0.11551	-0.07701	0.79990	0.05635	0.09014	-0.02999	…	**-0.269538778**
2	4	-0.04868	0.80375	-0.20009	0.00585	0.01980	-0.07713	…	**-0.13252447**
…	…	…	…	…	…	…	…	…	**…**
3	1	-0.06458	-0.08601	-0.03342	-0.06053	-0.08483	-0.14726	…	**0.015597229**
3	2	-0.04868	-0.19624	-0.03342	-0.07017	-0.01789	0.07074	…	**0.730455521**
3	3	-0.29027	-0.07100	0.79990	0.049182	0.06387	-0.07363	…	**-0.264375632**
3	4	-0.06473	0.91398	-0.03342	0.00892	0.02461	-0.05465	…	**-0.13252447**
…	…	…	…	…	…	…	…	…	**…**

Taguchi's method defined the SNR used to measure robustness, which is the transformed form of the performance quality characteristic (output value) used to analyze the results. Since the objective of this experiment is to minimize the quality characteristic of the output (amount of processing time used per a map reduce Job), the SNR for the quality characteristic "the smaller the better" is given by formula 2, that is:

$$SN_i = -10\log\left(\sum_{u=1}^{Ni}\frac{y_u^2}{N_i}\right) \quad (2)$$

The SNR result for each experiment is shown in Table 3.12. Complete SNR result tables for the *job processing time, job turnaround and hard disk bytes written* experiments are presented in ANNEX V, ANNEX VI and ANNEX VII respectively.

Table 3.12 Processing time SNR results

Experi-ment	Time of system up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	…	Processing time Trial 1	Processing time Trial 2	Processing Time Trial 3	SNR
1	< 0	< 0	< 0	< 0	…	**-0.1839028**	**0.5155972**	**0.4155972**	**-0.999026**
2	< 0	< 0	< 0	< 0	…	**-0.1708835**	**0.7304555**	**0.7304555**	**-0.45658085**
3	< 0	< 0	≥ 0	≥ 0	…	**-0.1714686**	**-0.269538**	**0.2643756**	**1.25082414**
4	< 0	≥ 0	< 0	≥ 0	…	**-0.1325244**	**-0.132524**	**-0.132524**	**15.7043319**
5	< 0	≥ 0	≥ 0	< 0	…	**-0.1856763**	**-0.267772**	**-0.269537**	**1.39727504**
6	< 0	≥ 0	≥ 0	< 0	…	**-0.2677778**	**-0.269537**	**-0.185676**	**1.39727504**
7	< 0	≥ 0	≥ 0	≥ 0	…	**-0.1714686**	**-0.174542**	**-0.174542**	**3.98029432**
8	≥ 0	< 0	≥ 0	< 0	…	**-0.2688839**	**-0.267712**	**-0.268355**	**5.32068168**
9	≥ 0	< 0	< 0	≥ 0	…	**0.81432367**	**0.8143236**	**0.8143236**	**15.7761839**
10	≥ 0	≥ 0	≥ 0	< 0	…	**-0.1325244**	**-0.132524**	**-0.132524**	**15.7043319**
11	≥ 0	≥ 0	< 0	≥ 0	…	**-0.1837929**	**-0.182090**	**-0.269544**	**1.24567693**
12	≥ 0	≥ 0	< 0	< 0	…	**-0.1714686**	**-0.269538**	**-0.269538**	**1.23463636**

According to Taguchi's method, the factor effect is equal to the difference between the highest average SNR and the lowest average SNR for each factor (see Table 3.5). This means that the larger the factor effect for a parameter, the larger the effect the variable has on the process, or, in other words, the more significant the effect of the factor. Table 3.13 shows the factor effect for each variable studied in the experiment. Factor effect tables for *job turnaround time and hard disk bytes written* output values are presented in ANNEX VIII and ANNEX IX.

Table 3.13 Factor effect rank on the job processing time output objective

	Time of System Up	Map tasks capacity	Reduce tasks capacity	Net. Rx bytes	Net. Tx bytes	CPU utiliza-tion	HD bytes read	Memory utiliza-tion	Respon-se time
Average SNR at Level 1	3.18205	4.1784165	5.4175370	3.3712	3.8949	6.57901	5.11036	2.005514	4.011035
Average SNR at Level 2	7.85630	5.8091173	4.8417803	7.5914	6.0116	3.58260	5.15667	8.253802	6.248281
Factor Effect (difference)	4.67424	1.6307007	0.5757566	4.2202	2.1166	2.99641	0.04630	6.248288	2.237245
Rank	**2**	**7**	**8**	**3**	**6**	**4**	**9**	**1**	**5**

3.4 Results

3.4.1 Analysis and interpretation of results

Based on the results presented in Table 3.13, it can be observed that:

- *Memory utilization* is the factor that has the most influence on the quality objective (*processing time* used per a MapReduce Job) of the output observed, at 6.248288, and
- *Hard disk bytes read* is the least influential factor in this experiment, at 0.046390.

Figure 3.5 presents a graphical representation of the factor results and their levels. ANNEX X and ANNEX XI present the graphical representations of *job turnaround time and hard disk bytes written* output objectives.

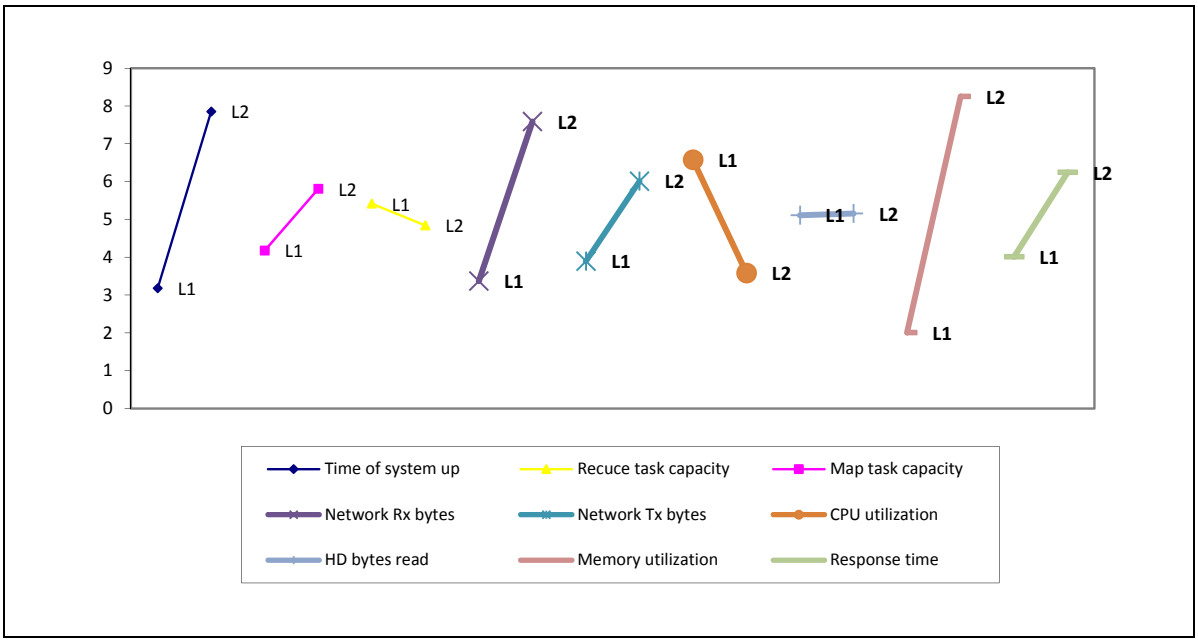


Figure 3.5 Graphical representations of factors and their SNR levels

To represent the optimal condition of the levels, also called the *optimal solution of the levels*, an analysis of SNR values is necessary in this experiment. Whether the aim is to minimize or maximize the quality characteristic (*job processing time* used per a MapReduce Job), it is always necessary to maximize the SNR parameter values. Consequently, the optimum level

of a specific factor will be the highest value of its SNR. It can be seen that the optimum level for each factor is represented by the highest point in the graph (as presented in Figure 3.5); that is, L2 for time of system up, L2 for map task capacity, L1 for reduce task capacity, etc.

Using the findings presented in Tables 3.12 and 3.13 and in Figure 3.5, it can be concluded that the optimum levels for the nine (9) factors in this experiment based on our experimental configuration cluster are presented in Table 3.14. ANNEX XII and ANNEX XIII present tables of the optimum levels of *job turnaround time and hard disk bytes written* factor output objectives.

Table 3.14 Optimum levels for factors of the processing time output

Factor number	Performance measure	Optimum level
1	Time of CC System Up	≥ 0 (L2)
2	Load map tasks capacity	≥ 0 (L2)
3	Load reduce tasks capacity	< 0 (L1)
4	Network Rx bytes	≥ 0 (L2)
5	Network Tx bytes	≥ 0 (L2)
6	CPU utilization	< 0 (L1)
7	Hard disk bytes read	≥ 0 (L2)
8	Memory utilization	≥ 0 (L2)
9	Response time	≥ 0 (L2)

3.4.2 Statistical data analysis of job processing time

The analysis of variance (ANOVA) is a statistical technique typically used in the design and analysis of experiments. According to Trivedi (Trivedi 2002), the purpose of applying the ANOVA technique to an experimental situation is to compare the effect of several factors applied simultaneously to the response variable (quality characteristic). It allows the effects of the controllable factors to be separated from those of uncontrolled variations. Table 3.15 presents the results of this ANOVA analysis of the experimental factors.

Table 3.15 Analysis of variance of job processing time output objective (ANOVA)

Factors	Degrees of Freedom	Sum of Squares (SS)	Variance (MS)	Contri-bution (%)	Variance ration (F)
Time of CC system up	1	21.84857	21.84857	21.814	101.87
Load map tasks capacity	1	2.659185	2.659185	2.655	12.39
Load reduce tasks capacity	1	0.331495	0.331495	0.330	1.54
Network Rx bytes	1	17.81038	17.81038	17.782	83.04
Network Tx bytes	1	4.480257	4.480257	4.473	20.89
CPU utilization	1	8.978526	8.978526	8.964	41.86
Hard disk bytes read	1	0.002144	0.002144	0.002	0.001
Memory utilization	1	39.04110	39.04110	38.979	182.04
Response time	1	5.005269	5.005269	4.997	23.33
Error	0	0.0000	0.0000		
Total	9	100.15		100	
Error estimate	1	0.0021445			

As can be seen in the contribution column of Table 3.15, these results can be interpreted as follows (represented graphically in Figure 3.6):

- *Memory utilization* is the factor that has the most influence (almost 39% of the contribution) on the processing time in this experiment.
- *Time of CC system up* is the factor that has the second greatest influence (21.814% of the contribution) on the processing time.
- *Network Rx bytes* is the factor that has the third greatest influence (17.782% of the contribution) on the processing time.
- *Hard disk bytes read* is the factor with the least influence (0.002% of the contribution) on the processing time in the cluster.

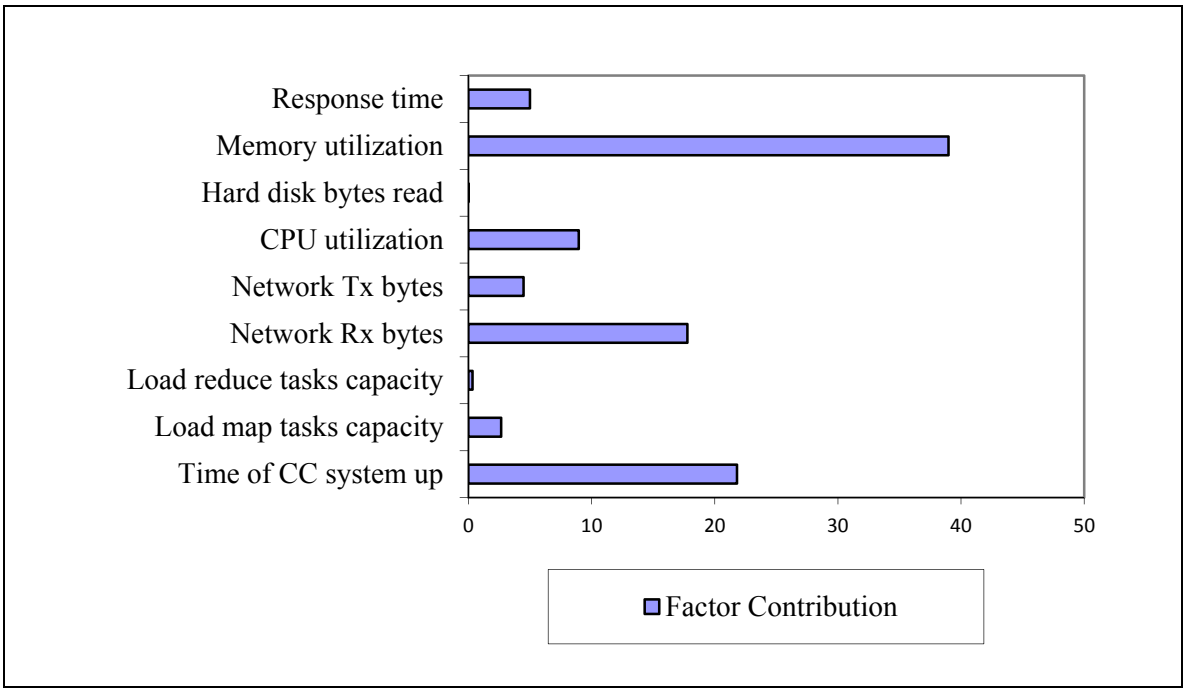


Figure 3.6 Percentage contribution of factors

In addition, based on the column related to the variance ratio F shown in Table 3.15, it can be concluded that:

- The factor *Memory utilization* has the most dominant effect on the output variable.
- According to Taguchi's method, the factor with the smallest contribution is taken as the error estimate. So, the factor *Hard disk bytes read* is taken as the error estimate, since it corresponds to the smallest sum of squares.

The results of this case study show, based on both the graphical and statistical data analyses of the SNR, that the *Memory utilization* required to process a MapReduce application in our cluster has the most influence, followed by the Time of CC system up and, finally, Network Rx bytes.

3.4.3 Statistical data analysis of job turnaround

The statistical data analysis of job turnaround output objective is presented in Table 3.16.

Table 3.16 Analysis of variance of job turnaround output objective (ANOVA)

Factors	Degrees of Freedom	Sum of Squares (SS)	Variance (MS)	Contri-bution (%)	Variance ration (F)
Time of CC system up	1	1.6065797	1.6065797	11.002	174.7780
Load map tasks capacity	1	3.0528346	3.0528346	20.906	0.020906
Load reduce tasks capacity	1	7.2990585	7.2990585	49.984	0.049984
Network Rx bytes	1	0.0176696	0.0176697	0.121	0.000121
Network Tx bytes	1	0.1677504	0.1677504	1.148	0.001148
CPU utilization	1	0.0009192	0.0009192	0.006	0.62E-05
Hard disk bytes read	1	2.3993583	2.3993583	16.431	0.064308
Memory utilization	1	0.0521259	0.0521259	0.357	0.000356
Response time	1	0.0064437	0.0064437	0.044	0.000044
Error	0	0.0000	0.0000		
Total	9	14.602740		100	
Error estimate	1	0.0009192			

As can be seen in the contribution column of Table 3.16, these results can be interpreted as follows (represented graphically in Figure 3.7):

- *Load reduce task capacity* is the factor that has the most influence (almost 50% of the contribution) on the job turnaround in this experiment.
- *Load map task capacity* is the factor that has the second greatest influence (almost 21% of the contribution) on the job turnaround.
- *Hard disk bytes read* is the factor that has the third greatest influence (16.431% of the contribution) on the job turnaround.
- *CPU utilization* is the factor with the least influence (0.006% of the contribution) on the job turnaround in the cluster system.

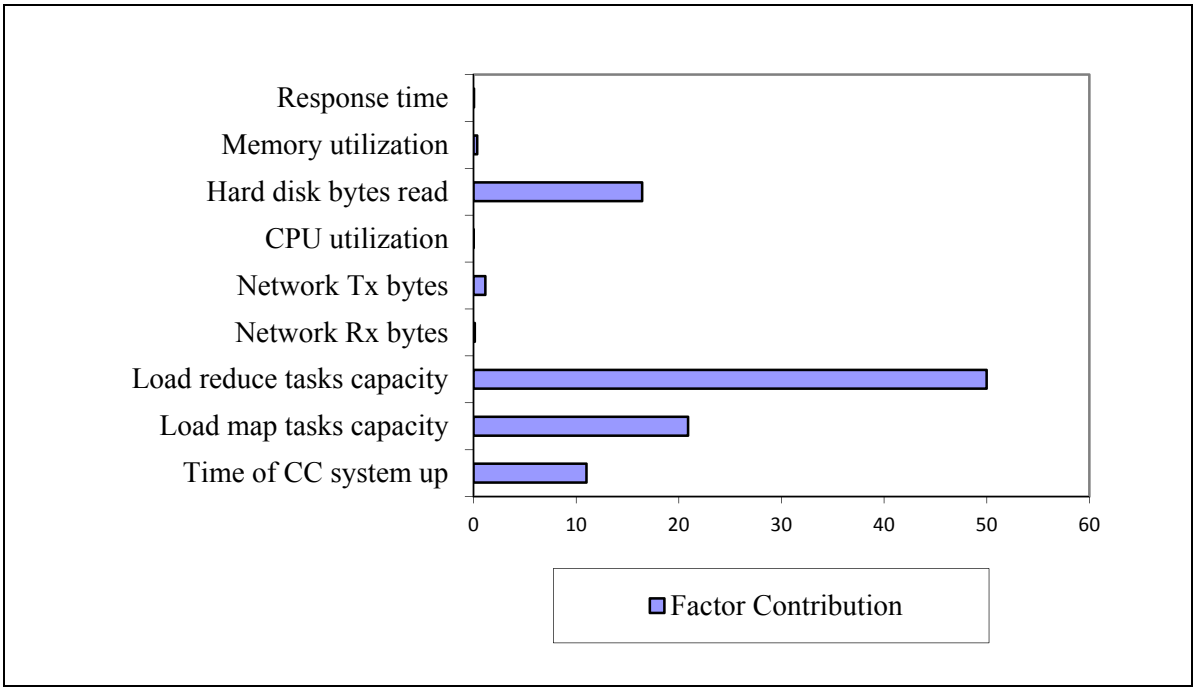


Figure 3.7 Percentage contribution of factors for job turnaround output objective

In addition, based on the column related to the variance ratio F shown in Table 3.16, it can be concluded that:

- The factor *Time of CC system up* has the most dominant effect on the output variable.
- According to Taguchi's method, the factor with the smallest contribution is taken as the error estimate. So, the factor *CPU utilization* is taken as the error estimate, since it corresponds to the smallest sum of squares.

The results of this case study show, based on both the graphical and statistical data analysis of the SNR, that the *Load reduce task capacity* into which is used by the Job in a MapReduce application in our cluster has the most influence in its job turnaround measure.

3.4.4 Statistical data analysis of hard disk bytes written

The statistical data analysis of hard disk bytes written output objective is presented in Table 3.17.

Table 3.17 Analysis of variance of hard disk bytes written output objective (ANOVA)

Factors	Degrees of Freedom	Sum of Squares (SS)	Variance (MS)	Contri-bution (%)	Variance ration (F)
Time of CC system up	1	2.6796517	2.6796517	37.650	69.14399
Load map tasks capacity	1	0.0661859	0.0661859	0.923	0.009299
Load reduce tasks capacity	1	0.0512883	0.0512883	0.720	0.007206
Network Rx bytes	1	0.1847394	0.1847394	2.595	0.025956
Network Tx bytes	1	0.4032297	0.4032297	5.665	0.056655
CPU utilization	1	1.3316970	1.3316970	18.711	0.187108
Hard disk bytes read	1	2.3011542	2.3011542	32.332	0.323321
Memory utilization	1	0.0387546	0.0387546	0.544	0.005445
Response time	1	0.0605369	0.0605369	0.850	0.008505
Error	0	0.0000	0.0000		
Total	9	7.1172380		100	
Error estimate	1	0.0387546			

As can be seen in the contribution column of Table 3.17, these results can be interpreted as follows (represented graphically in Figure 3.8):

- *Time of CC system up* is the factor that has the most influence (37.650% of the contribution) on the hard disk bytes written output objective in this experiment.
- *Hard disk bytes read* is the factor that has the second greatest influence (32.332% of the contribution) on the hard disk bytes written.
- *CPU utilization* is the factor that has the third greatest influence (18.711% of the contribution) on the hard disk bytes written.
- *Memory utilization* is the factor with the least influence (0.544% of the contribution) on the hard disk bytes written in the cluster system.

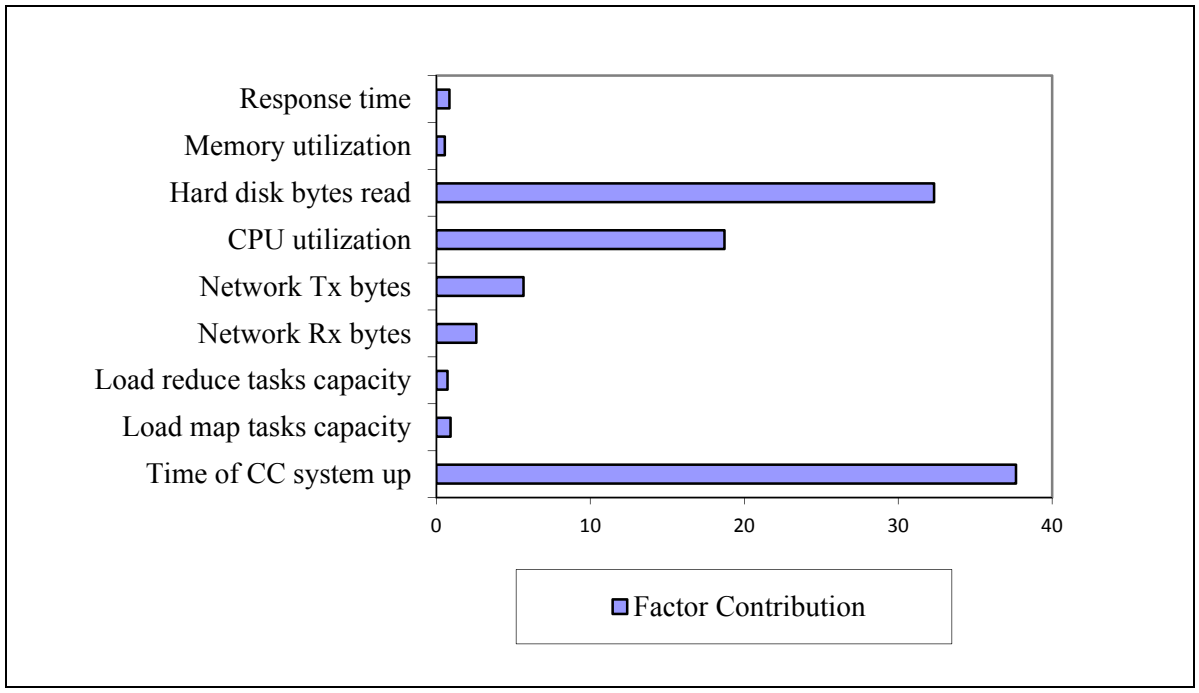


Figure 3.8 Percentage contribution of factors for hard disk bytes written output objective

In addition, based on the column related to the variance ratio F shown in Table 3.17, it can be concluded that the following:

- The factor *Time of CC system up* has the most dominant effect on the output variable.
- According to Taguchi's method, the factor with the smallest contribution is taken as the error estimate. So, the factor *Memory utilization* is taken as the error estimate, since it corresponds to the smallest sum of squares.

The results of this experiment show, based on both the graphical and statistical data analysis of the SNR, that the *Time of CC system up* while a Job MapReduce application is executed in our cluster has the most influence in the hard disk written.

3.5 Summary of performance measurement analysis

To summarize, when an application is developed by means of MapReduce framework and is executed in the experimental cluster, the factors *job processing time, job turn around,* and *hard disk bytes written*, must be taken into account in order to improve the performance of

the CCA. Moreover, the summary of performance concepts and measures which are affected by the contribution performance measures is shown in Figure 3.9.

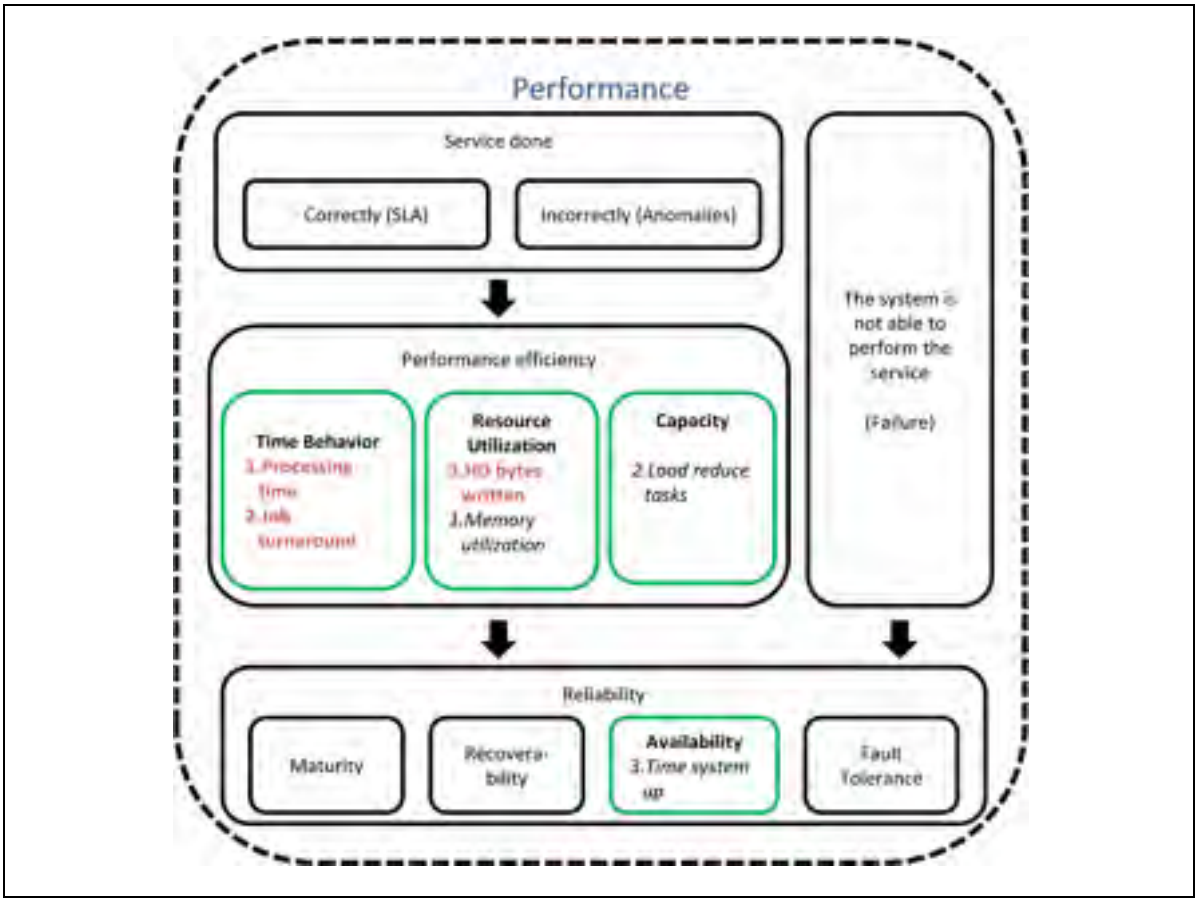


Figure 3.9 Summary of performance measurement analysis

Figure 3.9 shows that the performance on this experiment is determined by two sub concepts; *Time behavior and Resource utilization*. The results of the performance analysis show that the main performance measures involved in these sub concepts are: *Processing time, Job turnaround and Hard disk bytes written*. In addition, there are two sub concepts which have greater influence in the performance sub concepts; *Capacity and Availability*. These concepts contribute with the performance by means of their specific performance measures which have contribution in the behavior of the performance measures, they are respectively: *Memory utilization, Load reduce task, and Time system up*.

CONCLUSION

This chapter presents the conclusions of this thesis which proposes a performance measurement model for cloud computing applications - PMMoCCA. This performance measurement model is based on a measurement framework for cloud computing which has been validated by researchers and practitioners. Such framework defines the elements necessary to measure the performance of a cloud computing system using software quality concepts. The design of the framework is based on the concepts of metrology, along with aspects of software quality directly related to the performance concept, which are addressed in the ISO 25010 international standard.

It was found through the literature review that the performance efficiency and reliability concepts are closely associated with the performance measurement. As a result, the performance measurement model for CCA which is proposed in this thesis, integrates ISO 25010 concepts into a perspective of measurement for CCA in which terminology and vocabulary associated are aligned with the ISO 25010 international standard.

In addition, this thesis proposes a methodology as part of the performance measurement model for determining the relationships among the CCA performance measures. One of the challenges that addresses this methodology is how to determine the extent to which the performance measures are related, and to their influence in the analysis of CCA performance. This means, the key design problem is to establish which performance measures are interrelated and how much they contribute to each of performance concepts defined in the PMFCCA. To address this challenge, we proposed the use of a methodology based on Taguchi's method of experimental design combined with traditional statistical methods.

Experiments were carried out to analyze the relationships between the performance measures of several MapReduce applications and performance concepts that best represent the performance of CCA, as for example CPU processing time and time behavior. We found that when an application is developed in the MapReduce programming model to be executed in

the experimental cloud computing system, the performance on the experiment is determined by two main performance concepts; Time behavior and Resource utilization. The results of performance analysis show that the main performance measures involved in these concepts are: *Processing time, Job turnaround and Hard disk bytes written*. Thus, these measures must be taken into account in order to improve the performance of the application.

1. Answers to the research questions

This thesis focused on the development of a Performance Measurement Model for Cloud Computing Applications (PMMoCCA). This PMMoCCA defines how to measure CC performance characteristics related to CCA that affect to the whole CCS in order to improve the performance.

The research goals must also address the following research question:

- How can the performance of Cloud Computing Applications be improved?

and, more specifically:

1. What is the measurement process to analyze the performance of CCA?
2. Which CCS characteristics are more related with the performance of CCA?
3. Is there an existing method able to measure the above characteristics from the perspective of maintainers, developers and users?
4. How can the PMMoCCA be used in practice to analyze the performance in order to improve CCA in an organization?

1.1 Research sub-question 1 – discussion

The literature review shows that there are different approaches to the performance measurement of CCA. Standards like ISO 14756 states that performance measurement of

computer based software systems (CBSS) consists of a specified configuration of its hardware, its system software and application software. This means, all the hardware components and all software components shall be specified in detail and none of them shall have any change or special modification during the measurement process to getting better results in the measurement. Others measurement processes are focused on building predictive models using regression analysis which enables to model the behavior of the system and try to predict the performance of a system application. Moreover, they are focused on modeling the reliability of large, high-performance, computer systems to try to measure the system performance.

Others measurement approaches are based on the use of automated anomaly detection frameworks that can process massive volume of diverse health-related data by means of pattern recognition technologies. This type of performance measurement examines the performance of a set of benchmarks designed to represent a typical High Performance Computing (HPC) workload running on public clouds. Studies in performance measurement and analysis of application are focused on network I/O (network-intensive applications) in virtualized Clouds. The objective of this type of studies is to understand the performance impact of co-locating applications in a virtualized Cloud, in terms of throughput performance and resource sharing effectiveness.

1.2 Research sub-question 2 – discussion

This thesis identifies different CCS characteristics which are present when a performance measurement is carried out. These characteristics, also called performance concepts, arise when a CCS performs a service correctly and are; 1) responsiveness, 2) productivity, and 3) utilization. Moreover, these CCS performance concepts have a strong relationship which software engineering quality concepts as proposed by ISO 25010. The ISO 25010 concepts of performance efficiency and reliability are involved in the measurement of CCS and as a consequence in CCA. Sub concepts derived from these ISO 25010 concepts such as *time behavior, resource utilization, capacity, maturity, recoverability, availability and fault*

tolerance, define the performance measures which should be used in order to carry out a performance measurement of a CCA (see section 3.2.1).

1.3 Research sub-question 3 – discussion

Although there are in the literature measurement methods for the performance measurement of CCA, their approach is from an infrastructure standpoint and does not consider CCA performance factors from a software engineering application perspective. In addition, these methods only include a maintainer viewpoint and do not take in account others viewpoints like developers and users. This thesis is based on the performance evaluation of CCA by means of a proposed performance measurement model which can be adapted to the needs of different stakeholders in an organization. Moreover, the proposed measurement model can be used with different statistical analysis models in order to obtain valuable information which help to improve the organization performance.

1.4 Research sub-question 4 – discussion

One of the main challenges of organizations is how to measure and represent the performance of software systems so that they can be used in improving themselves. The lack of information which helps to understand and define concepts of assurances of availability, reliability and liability in Cloud Computing Applications (CCA), is a main issue in organization. Other concepts such as price, performance, time to completion (availability), probability of failure and liability are key to being able to produce a comparison service, in order to establish Service Level Agreements (SLA) or design better mechanisms to improve the performance in CCA and as consequence in the organizations. The proposed performance measurement model for CCA defines a measurement framework, a measurement process, their performance measures and a performance analysis method in order to measure and represent the performance of CCA. In addition, this model matches performance analysis results with performance concepts of software engineering which can be used by organizations at the time of the design and development software systems in cloud

environments, software systems maintenance plans, product comparison services or establishing Service Level Agreements (SLA).

1.5 Contributions

The main contributions of this thesis are:

- Findings in literature review show that there is not a unique procedure to measure the performance of cloud computing systems and, more specifically, cloud computing applications. Furthermore, it was found that there are different methods to analyze the performance of CCA but none of them align their results to quality concepts such as those used in organizations for evaluating performance.
- A detailed inventory of performance measurement activities and processes and their references are provided in order to carry out a performance measurement of CCA.
- As part of this thesis a Performance Measurement Framework for CC is proposed which can be used along with different statistical methods by aligning extracted performance measures from CCS and CCA with different performance concepts of software engineering.
- The proposed performance measurement model for CCA includes an experiment as a case study which uses of a methodology to establish relationships between extracted performance measures and performance concepts of software engineering. Furthermore, this methodology allows to represent these performance concepts from a quantitative point of view.
- This performance measurement model for CCA, can be used in any cloud computing environment by the alignment of their performance measures with the performance concepts defined in 3.2.1 by means of their formulae.
- In addition, this model includes a novel perspective for the performance measurement including software engineering concepts in traditional performance measures used in the performance measurement of CCA.

- The results of this research have been progressively made public from 2011 to 2014 at cloud computing conferences, software measurement conferences, book chapters and in software engineering journals.
    1. Luis Bautista, Alain April (2011) "Sustainability of Hadoop Clusters", 1st International Conference on Cloud Computing and Services Sciences (CLOSER 2011), Noordwijkerhout, The Netherlands, 7-9 May, ISBN: 978-989-8425-52-2, pp. 587-590
    2. Luis Bautista, Alain Abran and Alain April (2012), "Design of a Performance Measurement Framework for Cloud Computing," Journal of Software Engineering and Applications, Vol. 5 No. 2, pp. 69-75.
    3. Luis Bautista, Alain April and Alain Abran (2013), "A Methodology for Identifying the Relationships between Performance Factors for Cloud Computing Applications", 04/2013; Chapter: 15 in book: Software Engineering Frameworks for the Cloud Computing Paradigm, Edition: 2013, XVIII, 365 p. 122 illus., Publisher: Springer, Editors: Zaigham Mahmood, Saqib Saeed. ISBN: 978-1-4471-5030-5
    4. Luis Bautista, Alain April and Alain Abran (2014), "DIPAR: A Framework for Implementing Big Data Science in Organizations", Chapter: 8 in Book: Continued Rise of the Cloud: Advances and Trends in Cloud Computing, Edition: 2014, Publisher: Springer, Editor: Zaigham Mahmood. ISBN 978-1-4471-6451-7
    5. Luis Bautista, Alain April (2014) "Methodology to Determine Relationships between Performance Factors in Hadoop Cloud Computing Applications ", 4th International Conference on Cloud Computing and Services Sciences (CLOSER 2014), Barcelona, Spain, 3-5 April, ISBN: 978-989-758-019-2, pp. 375-386
    6. Anderson Ravanello, Jean-Marc Desharnais, Luis Bautista, Alain April and Abdelouahed Gherbi (2014), “Performance measurement for cloud computing applications using ISO 25010 standard characteristics”, To appear - Joint Conference of the 24rd International Workshop on Software Measurement &

9th International Conference on Software Process and Product Measurement - IWSM-MENSURA 2014, Rotterdam (Netherlands), Oct. 6-8, 2014

1.6 Future works

Further research is needed to design new performance measurement methods and mechanisms to analyze the performance of Cloud Computing applications. This future research could contribute to validating and improve the proposed PMMoCCA and to include new performance measures as well as their definition and description.

In order to design new performance measurements methods it is necessary to design a repository of performance measures which provides information and tools to facilitate the design, validation, and comparison of performance analysis models and algorithms for CCS and CCA. The purpose of this repository is to help to establish *attribute*–performance relationships relating to specific applications with relatively well-known demands on systems to be able to determine how comparison services may be formulated. As result of this repository, new performance analysis techniques could be developed and tested in order to extend the performance measurement model proposed in this thesis.

We therefore expect that future research models will be proposed to analyze the “normal node behavior” of CCS and CCA by means of advanced analysis methods such as machine learning and big data analysis.

ANNEX I

COLLECTED PERFORMANCE MEASURES EXTRACTED FROM A HADOOP SYSTEM APPLICATION

Measure	Source	Description
jobs:clusterMapCapacity	Jobs of CCA	Maximum number of available maps to be created by a job
jobs:clusterReduceCapacity	Jobs of CCA	Maximum number of available reduces to be created by a job
jobs:finishTime	Jobs of CCA	Time at which a job was completed
jobs:firstJobCleanupTaskLaunchTime	Jobs of CCA	Time at which a job is retired from the cluster after completed
jobs:JobSetupTaskLaunchTime	Jobs of CCA	Time at which a job is setup in the cluster for processing
jobs:firstMapTaskLaunchTime	Jobs of CCA	Time at which a first map task of a specific job is launched
jobs:firstReduceTaskLaunchTime	Jobs of CCA	Time at which a first reduce task of a specific job is launched
jobs:jobId	Jobs of CCA	Job ID
jobs:launchTime	Jobs of CCA	Time at which a job is launched for processing
jobs:mapSlotSeconds	Jobs of CCA	Number of map slots used per seconds by a job
jobs:numMaps	Jobs of CCA	Number of maps created by a job
jobs:numReduces	Jobs of CCA	Number of reduces created by a jobs
jobs:numSlotsPerMap	Jobs of CCA	Number of slots used per a map task
jobs:reduceSlotsSeconds	Jobs of CCA	Number of reduce slots used per seconds by a job
jobs:Status	Jobs of CCA	Job status after processing (Successful or Failed)
jobs:submitTime	Jobs of CCA	Time at which a job was submitted for processing
cpu:idle	CC System	Time of CPU not doing any work
cpu:sys	CC System	Percentage of CPU used by the operating system itself

cpu:user	CC System	Percentage of CPU used by user applications
disk:ReadBytes	CC System	Amount of HD bytes read by a job
disk:Reads	CC System	Amount of HD reads done by a job
disk:WriteBytes	CC System	Amount of HD bytes written by a job
disk:Writes	CC System	Amount of HD writes done by a job
memory:ActualFree	CC System	Amount of free memory on a specific time
memory:ActualUsed	CC System	Amount of used memory on a specific time
memory:FreePercent	CC System	Percentage of free memory on a specific time
memory:Total	CC System	Total of RAM in the system on a specific time
memory:Used	CC System	Amount of average memory used on a specific time
memory:UsedPercent	CC System	Percentage of average memory used on a specific time
network:RxBytes	CC System	Amount of network bytes received on a specific time
network:RxDropped	CC System	Amount of network bytes dropped on a specific time
network:RxErrors	CC System	Amount of network errors during received transmission on a specific time
network:RxPackets	CC System	Amount of network packets received on a specific time
network:TxBytes	CC System	Amount of network bytes transmitted on a specific time
network:TxCollisions	CC System	Amount of network collisions in transmitted packets on a specific time
network:TxErrors	CC System	Amount of network errors during transmission on a specific time
network:TxPackets	CC System	Amount of network packets transmitted on a specific time
system:Uptime	CC System	Amount of time that the system has been up

ANNEX II
TRIALS, EXPERIMENTS, AND RESULTING VALUES FOR JOB PROCESSING TIME OUTPUT OBJECTIVE

Trial	Experi-ment	Time of system Up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	Network Tx bytes	CPU utiliza-tion	HD bytes read	Memory utiliza-tion	Respon-se time	**Processing time**
1	1	-0.44091	-0.08601	-0.03342	-0.04170	-0.08030	-0.00762	-0.00762	-0.20375	-0.08801	**-0.18390288**
1	2	-0.34488	-0.07100	-0.03342	-0.02022	-0.18002	0.16864	0.01302	0.13602	0.06995	**-0.1708835**
1	3	-0.49721	-0.08601	0.79990	0.01329	0.02184	-0.03221	-0.00760	-0.31021	0.20492	**-0.1714686**
1	4	-0.39277	0.01307	-0.03342	0.02418	0.08115	-0.02227	0.05008	0.15678	0.21719	**-0.13252447**
1	5	-0.39302	0.91398	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**-0.18567633**
1	6	-0.04868	0.91398	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**-0.26777782**
1	7	-0.49594	0.01307	0.79990	0.00215	-0.03908	0.02385	0.03924	-0.26190	0.20492	**-0.1714686**
1	8	0.15702	-0.19624	0.79990	-0.00881	0.02324	0.10820	0.05056	-0.00827	-0.27370	**-0.26888392**
1	9	0.19227	-0.07701	-0.03342	0.35088	0.74423	0.33852	-0.00625	0.14872	0.12210	**0.81432367**
1	10	0.41680	0.91398	0.79990	-0.06419	-0.08596	-0.06604	-0.00679	0.14274	0.21719	**-0.13252447**
1	11	0.19227	0.80375	-0.03342	0.09299	-0.07310	0.00610	-0.00762	-0.26175	-0.07881	**-0.18379299**
1	12	0.19227	0.013079	-0.03342	-0.04344	0.02184	-0.03221	0.00205	-0.31021	0.20492	**-0.1714686**
2	1	-0.03195	-0.08601	-0.03342	-0.06311	-0.09345	-0.17198	-0.00762	-0.20232	-0.15703	**0.51559723**
2	2	-0.01590	-0.19624	-0.03342	-0.06880	-0.01529	0.06993	0.00242	0.58463	0.08629	**0.73045552**
2	3	-0.11551	-0.07701	0.79990	0.05635	0.09014	-0.02999	-0.06897	-0.24807	0.14629	**-0.26953878**
2	4	-0.04868	0.80375	-0.20009	0.00585	0.01980	-0.07713	0.70895	0.03568	0.21719	**-0.13252447**
2	5	-0.02393	0.01307	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**-0.26777782**

2	6	-0.05602	0.35930	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**-0.26953796**
2	7	-0.48037	0.01307	0.79990	0.01621	-0.06085	0.00246	0.05515	-0.25045	0.08682	**-0.17454293**
2	8	0.20190	-0.19624	0.79990	-0.05146	0.05303	0.18861	0.81390	-0.00413	-0.27703	**-0.26771266**
2	9	0.42482	-0.07701	-0.03342	0.78087	0.77419	0.61072	-0.00716	0.46148	0.12210	**0.81432367**
2	10	0.42482	0.01307	0.79990	-0.06717	-0.08702	-0.13193	-0.00762	0.14087	0.21719	**-0.13252447**
2	11	0.42482	0.01307	-0.03342	0.00618	-0.02499	0.04906	-0.00762	-0.07440	-0.13403	**-0.18209049**
2	12	0.42482	0.80375	-0.20009	-0.03741	0.09014	-0.02999	0.03389	-0.24807	0.14629	**-0.26953878**
3	1	-0.06458	-0.08601	-0.03342	-0.06053	-0.08483	-0.14726	-0.00762	-0.06376	-0.15703	**0.41559723**
3	2	-0.04868	-0.19624	-0.03342	-0.07017	-0.01789	0.07074	0.08132	0.60821	0.08629	**0.73045552**
3	3	-0.29027	-0.07100	0.79990	0.049182	0.06387	-0.07363	-0.07240	-0.01116	0.12296	**-0.26437563**
3	4	-0.06473	0.91398	-0.03342	0.00892	0.02461	-0.05465	0.06548	0.04622	0.21719	**-0.13252447**
3	5	-0.04868	0.80375	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**-0.26953796**
3	6	-0.39302	0.01307	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**-0.18567633**
3	7	-0.48791	0.01307	0.79990	0.04494	-0.01795	0.19131	0.05276	-0.25062	0.08682	**-0.17454293**
3	8	0.21687	-0.19624	0.79990	-0.01194	0.04457	0.20503	0.81390	-0.00264	-0.30370	**-0.2683553**
3	9	0.20723	-0.07701	-0.03342	0.42740	0.53122	0.54756	-0.00644	0.47173	0.12210	**0.81432367**
3	10	0.43284	0.35930	0.79990	-0.06687	-0.08656	-0.14263	-0.00762	0.01657	0.21719	**-0.13252447**
3	11	0.09804	0.35930	-0.03342	0.29237	-0.04205	0.03557	-0.07220	-0.09377	-0.11703	**-0.26954448**
3	12	0.20723	0.80375	-0.03342	-0.00421	0.02110	-0.05948	0.04931	-0.11880	0.14629	**-0.26953878**

ANNEX III
TRIALS, EXPERIMENTS, AND RESULTING VALUES FOR JOB TURNAROUND OUTPUT OBJECTIVE

Trial	Experi-ment	Time of system Up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	Network Tx bytes	CPU utiliza-tion	HD bytes read	Memory utiliza-tion	Respon-se time	**Mapreduce Job turn around**
1	1	-0.44091	-0.08601	-0.03342	-0.04170	-0.08030	-0.00762	-0.00762	-0.20375	-0.08801	**-0.18390886**
1	2	-0.34488	-0.07100	-0.03342	-0.02022	-0.18002	0.16864	0.01302	0.13602	0.06995	**-0.17087273**
1	3	-0.49721	-0.08601	0.79990	0.01329	0.02184	-0.03221	-0.00760	-0.31021	0.20492	**-0.1714433**
1	4	-0.39277	0.01307	-0.03342	0.02418	0.08115	-0.02227	0.05008	0.15678	0.21719	**-0.13249859**
1	5	-0.39302	0.91398	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**-0.18567864**
1	6	-0.04868	0.91398	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**-0.26783498**
1	7	-0.49594	0.01307	0.79990	0.00215	-0.03908	0.02385	0.03924	-0.26190	0.20492	**-0.1714433**
1	8	0.15702	-0.19624	0.79990	-0.00881	0.02324	0.10820	0.05056	-0.00827	-0.27370	**-0.26892714**
1	9	0.19227	-0.07701	-0.03342	0.35088	0.74423	0.33852	-0.00625	0.14872	0.12210	**-0.21432136**
1	10	0.41680	0.91398	0.79990	-0.06419	-0.08596	-0.06604	-0.00679	0.14274	0.21719	**-0.13249859**
1	11	0.19227	0.80375	-0.03342	0.09299	-0.07310	0.00610	-0.00762	-0.26175	-0.07881	**-0.18379798**
1	12	0.19227	0.013079	-0.03342	-0.04344	0.02184	-0.03221	0.00205	-0.31021	0.20492	**-0.1714433**
2	1	-0.03195	-0.08601	-0.03342	-0.06311	-0.09345	-0.17198	-0.00762	-0.20232	-0.15703	**0.41558004**
2	2	-0.01590	-0.19624	-0.03342	-0.06880	-0.01529	0.06993	0.00242	0.58463	0.08629	**0.73041236**
2	3	-0.11551	-0.07701	0.79990	0.05635	0.09014	-0.02999	-0.06897	-0.24807	0.14629	**-0.26947932**
2	4	-0.04868	0.80375	-0.20009	0.00585	0.01980	-0.07713	0.70895	0.03568	0.21719	**-0.13249859**
2	5	-0.02393	0.01307	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**-0.26783498**

2	6	-0.05602	0.35930	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**-0.26958764**
2	7	-0.48037	0.01307	0.79990	0.01621	-0.06085	0.00246	0.05515	-0.25045	0.08682	**-0.17453028**
2	8	0.20190	-0.19624	0.79990	-0.05146	0.05303	0.18861	0.81390	-0.00413	-0.27703	**-0.2677568**
2	9	0.42482	-0.07701	-0.03342	0.78087	0.77419	0.61072	-0.00716	0.46148	0.12210	**0.81432136**
2	10	0.42482	0.01307	0.79990	-0.06717	-0.08702	-0.13193	-0.00762	0.14087	0.21719	**-0.13349859**
2	11	0.42482	0.01307	-0.03342	0.00618	-0.02499	0.04906	-0.00762	-0.07440	-0.13403	**-0.18210145**
2	12	0.42482	0.80375	-0.20009	-0.03741	0.09014	-0.02999	0.03389	-0.24807	0.14629	**-0.26947932**
3	1	-0.06458	-0.08601	-0.03342	-0.06053	-0.08483	-0.14726	-0.00762	-0.06376	-0.15703	**0.33558004**
3	2	-0.04868	-0.19624	-0.03342	-0.07017	-0.01789	0.07074	0.08132	0.60821	0.08629	**0.73041236**
3	3	-0.29027	-0.07100	0.79990	0.049182	0.06387	-0.07363	-0.07240	-0.01116	0.12296	**-0.26432233**
3	4	-0.06473	0.91398	-0.03342	0.00892	0.02461	-0.05465	0.06548	0.04622	0.21719	**-0.14249859**
3	5	-0.04868	0.80375	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**-0.26958764**
3	6	-0.39302	0.01307	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**-0.18567864**
3	7	-0.48791	0.01307	0.79990	0.04494	-0.01795	0.19131	0.05276	-0.25062	0.08682	**-0.17453028**
3	8	0.21687	-0.19624	0.79990	-0.01194	0.04457	0.20503	0.81390	-0.00264	-0.30370	**-0.2684059**
3	9	0.20723	-0.07701	-0.03342	0.42740	0.53122	0.54756	-0.00644	0.47173	0.12210	**0.81432136**
3	10	0.43284	0.35930	0.79990	-0.06687	-0.08656	-0.14263	-0.00762	0.01657	0.21719	**-0.13249859**
3	11	0.09804	0.35930	-0.03342	0.29237	-0.04205	0.03557	-0.07220	-0.09377	-0.11703	**-0.26954937**
3	12	0.20723	0.80375	-0.03342	-0.00421	0.02110	-0.05948	0.04931	-0.11880	0.14629	**-0.26947932**

ANNEX IV
TRIALS, EXPERIMENTS, AND RESULTING VALUES FOR HARD DISK BYTES WRITTEN OUTPUT OBJECTIVE

Trial	Experi-ment	Time of system Up	Map tasks capacity	Reduce tasks capacity	Network Rx bytes	Network Tx bytes	CPU utiliza-tion	HD bytes read	Memory utiliza-tion	Respon-se time	**HD written bytes utilization**
1	1	-0.44091	-0.08601	-0.03342	-0.04170	-0.08030	-0.00762	-0.00762	-0.20375	-0.08801	**-0.03206421**
1	2	-0.34488	-0.07100	-0.03342	-0.02022	-0.18002	0.16864	0.01302	0.13602	0.06995	**0.26373356**
1	3	-0.49721	-0.08601	0.79990	0.01329	0.02184	-0.03221	-0.00760	-0.31021	0.20492	**0.10575238**
1	4	-0.39277	0.01307	-0.03342	0.02418	0.08115	-0.02227	0.05008	0.15678	0.21719	**-0.00858399**
1	5	-0.39302	0.91398	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**0.00568632**
1	6	-0.04868	0.91398	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**0.000911**
1	7	-0.49594	0.01307	0.79990	0.00215	-0.03908	0.02385	0.03924	-0.26190	0.20492	**0.48626143**
1	8	0.15702	-0.19624	0.79990	-0.00881	0.02324	0.10820	0.05056	-0.00827	-0.27370	**-0.02614525**
1	9	0.19227	-0.07701	-0.03342	0.35088	0.74423	0.33852	-0.00625	0.14872	0.12210	**0.31637467**
1	10	0.41680	0.91398	0.79990	-0.06419	-0.08596	-0.06604	-0.00679	0.14274	0.21719	**-0.0417276**
1	11	0.19227	0.80375	-0.03342	0.09299	-0.07310	0.00610	-0.00762	-0.26175	-0.07881	**-0.02610864**
1	12	0.19227	0.013079	-0.03342	-0.04344	0.02184	-0.03221	0.00205	-0.31021	0.20492	**0.10575238**
2	1	-0.03195	-0.08601	-0.03342	-0.06311	-0.09345	-0.17198	-0.00762	-0.20232	-0.15703	**-0.04002126**
2	2	-0.01590	-0.19624	-0.03342	-0.06880	-0.01529	0.06993	0.00242	0.58463	0.08629	**-0.15830157**
2	3	-0.11551	-0.07701	0.79990	0.05635	0.09014	-0.02999	-0.06897	-0.24807	0.14629	**0.03263584**
2	4	-0.04868	0.80375	-0.20009	0.00585	0.01980	-0.07713	0.70895	0.03568	0.21719	**-0.00216128**
2	5	-0.02393	0.01307	0.79990	-0.05962	0.03435	0.10635	-0.07240	0.58698	-0.33036	**0.000911**

2	6	-0.05602	0.35930	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**0.000075**
2	7	-0.48037	0.01307	0.79990	0.01621	-0.06085	0.00246	0.05515	-0.25045	0.08682	**0.00786428**
2	8	0.20190	-0.19624	0.79990	-0.05146	0.05303	0.18861	0.81390	-0.00413	-0.27703	**-0.00845735**
2	9	0.42482	-0.07701	-0.03342	0.78087	0.77419	0.61072	-0.00716	0.46148	0.12210	**0.43795467**
2	10	0.42482	0.01307	0.79990	-0.06717	-0.08702	-0.13193	-0.00762	0.14087	0.21719	**-0.04071904**
2	11	0.42482	0.01307	-0.03342	0.00618	-0.02499	0.04906	-0.00762	-0.07440	-0.13403	**-0.02604023**
2	12	0.42482	0.80375	-0.20009	-0.03741	0.09014	-0.02999	0.03389	-0.24807	0.14629	**0.03263584**
3	1	-0.06458	-0.08601	-0.03342	-0.06053	-0.08483	-0.14726	-0.00762	-0.06376	-0.15703	**-0.03582089**
3	2	-0.04868	-0.19624	-0.03342	-0.07017	-0.01789	0.07074	0.08132	0.60821	0.08629	**0.46256752**
3	3	-0.29027	-0.07100	0.79990	0.049182	0.06387	-0.07363	-0.07240	-0.01116	0.12296	**-0.00199111**
3	4	-0.06473	0.91398	-0.03342	0.00892	0.02461	-0.05465	0.06548	0.04622	0.21719	**-0.02305365**
3	5	-0.04868	0.80375	0.79990	-0.07491	0.05004	0.09205	-0.07178	0.56174	-0.30036	**0.000075**
3	6	-0.39302	0.01307	0.79990	-0.01796	0.06881	0.03948	-0.00762	0.22850	-0.05427	**0.00568632**
3	7	-0.48791	0.01307	0.79990	0.04494	-0.01795	0.19131	0.05276	-0.25062	0.08682	**-0.00130791**
3	8	0.21687	-0.19624	0.79990	-0.01194	0.04457	0.20503	0.81390	-0.00264	-0.30370	**-0.02531442**
3	9	0.20723	-0.07701	-0.03342	0.42740	0.53122	0.54756	-0.00644	0.47173	0.12210	**0.28498627**
3	10	0.43284	0.35930	0.79990	-0.06687	-0.08656	-0.14263	-0.00762	0.01657	0.21719	**-0.04212046**
3	11	0.09804	0.35930	-0.03342	0.29237	-0.04205	0.03557	-0.07220	-0.09377	-0.11703	**-0.04820045**
3	12	0.20723	0.80375	-0.03342	-0.00421	0.02110	-0.05948	0.04931	-0.11880	0.14629	**-0.00319362**

ANNEX V
FACTOR EFFECT ON JOB PROCESSING TIME OUTPUT OBJECTIVE

Experi-ment	P1	P2	P3	P4	P5	P6	P7	P8	P9	Processing time Trial 1	Processing time Trial 2	Processing Time Trial 3	SNR
1	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	**-0.1839028**	**0.5155972**	**0.4155972**	**-0.999026**
2	< 0	< 0	< 0	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	**-0.1708835**	**0.7304555**	**0.7304555**	**-0.4565808**
3	< 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	**-0.1714686**	**-0.269538**	**0.2643756**	**1.25082414**
4	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	**-0.1325244**	**-0.132524**	**-0.132524**	**15.7043319**
5	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**-0.1856763**	**-0.267772**	**-0.269537**	**1.39727504**
6	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**-0.2677778**	**-0.269537**	**-0.185676**	**1.39727504**
7	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	**-0.1714686**	**-0.174542**	**-0.174542**	**3.98029432**
8	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	≥ 0	**-0.2688839**	**-0.267712**	**-0.268355**	**5.32068168**
9	≥ 0	< 0	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	≥ 0	**0.81432367**	**0.8143236**	**0.8143236**	**15.7761839**
10	≥ 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	≥ 0	< 0	**-0.1325244**	**-0.132524**	**-0.132524**	**15.7043319**
11	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	**-0.1837929**	**-0.182090**	**-0.269544**	**1.24567693**
12	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	≥ 0	< 0	**-0.1714686**	**-0.269538**	**-0.269538**	**1.23463636**

P1 = Time of system up
P2 = Map tasks capacity
P3 = Reduce tasks capacity
P4 = Network Rx bytes
P5 = CPU utilization
P6 = HD bytes read
P7 = Memory utilization
P8 = Response time
P9 = CPU utilization

ANNEX VI
FACTOR EFFECT ON MAP REDUCE JOB TURNAROUND OUTPUT OBJECTIVE

Experi-ment	P1	P2	P3	P4	P5	P6	P7	P8	P9	Job turn around Trial 1	Job turn around Trial 2	Job turn around Trial 3	SNR
1	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	**-0.1839088**	**0.41558004**	**0.33558004**	**-2.3831427**
2	< 0	< 0	< 0	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	**-0.1708727**	**0.73041236**	**0.73041236**	**-0.4565787**
3	< 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	**-0.1714433**	**-0.2694793**	**-0.2643223**	**1.25094064**
4	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	**-0.1324985**	**-0.1324985**	**-0.1424985**	**2.74286333**
5	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**-0.1856786**	**-0.2678349**	**-0.2695876**	**1.39686744**
6	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**-0.2678349**	**-0.2695876**	**-0.1856786**	**1.39686744**
7	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	**-0.1714433**	**-0.1745302**	**-0.1745302**	**3.97664403**
8	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	≥ 0	**-0.2689271**	**-0.2677568**	**-0.2684059**	**5.32115657**
9	≥ 0	< 0	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	≥ 0	**-0.2143213**	**0.81432136**	**0.81432136**	**-0.5275076**
10	≥ 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	≥ 0	< 0	**-0.1324985**	**-0.1334985**	**-0.1324985**	**4.72372344**
11	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	**-0.1837979**	**-0.1821014**	**-0.2695493**	**1.24573844**
12	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	≥ 0	< 0	**-0.1714433**	**-0.2694793**	**-0.2694793**	**1.23476506**

P1 = Time of system up
P2 = Map tasks capacity
P3 = Reduce tasks capacity
P4 = Network Rx bytes
P5 = CPU utilization
P6 = HD bytes read
P7 = Memory utilization
P8 = Response time
P9 = CPU utilization

ANNEX VII
FACTOR EFFECT ON HARD DISK BYTES WRITTEN UTILIZATION OUTPUT OBJECTIVE

Experi-ment	P1	P2	P3	P4	P5	P6	P7	P8	P9	HD written bytes Trial 1	HD written bytes Trial 2	HD written bytes Trial 3	SNR
1	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	< 0	**-0.0320642**	**-0.0400212**	**-0.0358208**	**1.91018131**
2	< 0	< 0	< 0	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	**0.26373356**	**-0.1583015**	**0.46256752**	**-1.6330432**
3	< 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	**0.10575238**	**0.03263584**	**-0.0019911**	**-0.4560860**
4	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	**-0.0085839**	**-0.0021612**	**-0.0230536**	**-0.1106766**
5	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**0.00568632**	**0.000911**	**0.000075**	**-0.6850961**
6	< 0	≥ 0	≥ 0	< 0	≥ 0	< 0	≥ 0	< 0	≥ 0	**0.000911**	**0.000075**	**0.00568632**	**-0.6850961**
7	< 0	≥ 0	≥ 0	≥ 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	**0.48626143**	**0.00786428**	**-0.0013079**	**-1.8658518**
8	≥ 0	< 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	≥ 0	**-0.0261452**	**-0.0084573**	**-0.0253144**	**0.56476973**
9	≥ 0	< 0	< 0	≥ 0	≥ 0	< 0	≥ 0	≥ 0	≥ 0	**0.31637467**	**0.43795467**	**0.28498627**	**1.25654573**
10	≥ 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	≥ 0	< 0	**-0.0417276**	**-0.0407190**	**-0.0421204**	**3.51836121**
11	≥ 0	≥ 0	< 0	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	**-0.0261086**	**-0.0260402**	**-0.0482004**	**0.81445397**
12	≥ 0	≥ 0	< 0	< 0	< 0	≥ 0	< 0	≥ 0	< 0	**0.10575238**	**0.03263584**	**-0.0031936**	**-0.4876445**

P1 = Time of system up
P2 = Map tasks capacity
P3 = Reduce tasks capacity
P4 = Network Rx bytes
P5 = CPU utilization
P6 = HD bytes read
P7 = Memory utilization
P8 = Response time
P9 = CPU utilization

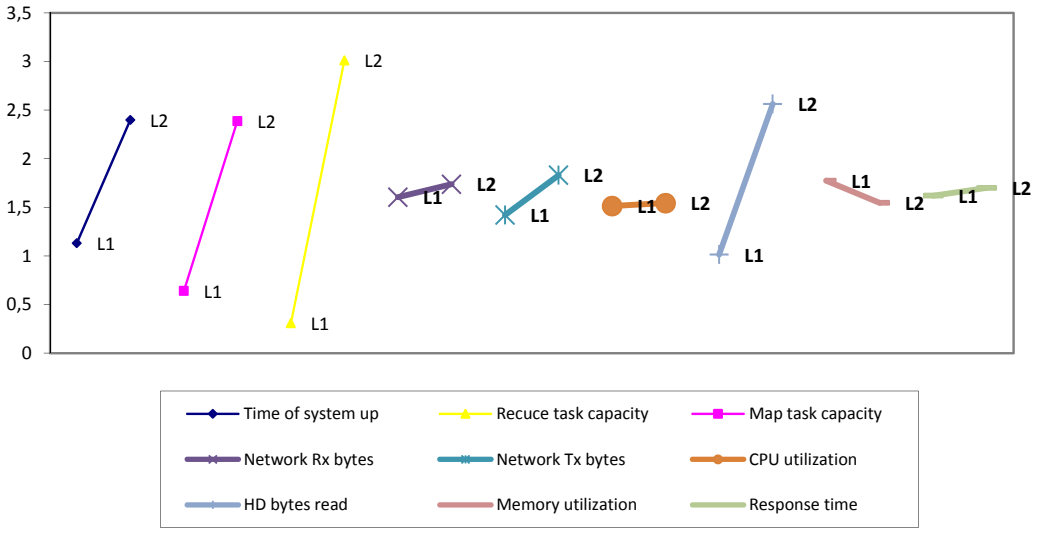
ANNEX VIII
FACTOR EFFECT RANK ON JOB TURNAROUND OUTPUT OBJECTIVE

	Time of system Up	**Map tasks capacity**	**Reduce tasks capacity**	**Network Rx bytes**	**Network Tx bytes**	**CPU utiliza-tion**	**HD bytes read**	**Memory utiliza-tion**	**Response time**
Average SNR at Level 1	1.1320659	0.6409736	0.3093562	1.6048083	1.4212768	1.5138299	1.0147838	1.7743503	1.6200584
Average SNR at Level 2	2.3995751	2.3882098	3.0110332	1.7377357	1.8308503	1.5441484	2.5637700	1.5460392	1.7003311
Factor Effect (difference)	**1.2675092**	**1.7472362**	**2.7016769**	**0.1329274**	**0.4095735**	**0.0303185**	**1.5489862**	**0.2283111**	**0.0802727**
Rank	**4**	**2**	**1**	**7**	**5**	**9**	**3**	**6**	**8**

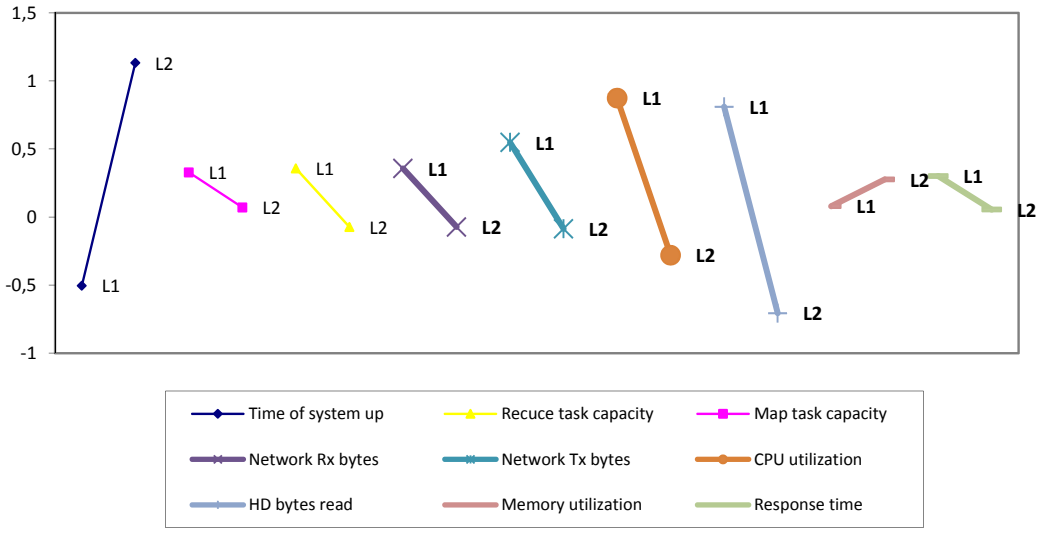
ANNEX IX
FACTOR EFFECT RANK ON HARD DISK BYTES WRITTEN OUTPUT OBJECTIVE

	Time of system Up	**Map tasks capacity**	**Reduce tasks capacity**	**Network Rx bytes**	**Network Tx bytes**	**CPU utiliza-tion**	**HD bytes read**	**Memory utiliza-tion**	**Response time**
Average SNR at Level 1	-0.503669	0.3284735	0.2916361	0.3574903	0.5488203	0.8748270	0.8104662	0.0799704	0.3014226
Average SNR at Level 2	1.1332972	0.0712071	0.0651668	-0.072323	-0.086183	-0.279164	-0.706489	0.2768324	0.0553802
Factor Effect (difference)	**1.6369641**	**0.2572663**	**0.2264692**	**0.4298132**	**0.6350037**	**1.1539917**	**1.5169555**	**0.1968620**	**0.2460424**
Rank	**1**	**6**	**8**	**5**	**4**	**3**	**2**	**9**	**7**

ANNEX X
GRAPHICAL REPRESENTATION OF JOB TURNAROUND TIME OUTPUT OBJECTIVE



ANNEX XI
GRAPHICAL REPRESENTATION OF HARD DISK BYTES WRITTEN OUTPUT OBJECTIVE



ANNEX XII
OPTIMUM LEVELS OF JOB TURNAROUND TIME FACTOR

Factor number	Performance measure	Optimum level
1	Time of CC System Up	≥ 0 (L2)
2	Load map tasks capacity	≥ 0 (L2)
3	Load reduce tasks capacity	≥ 0 (L2)
4	Network Rx bytes	≥ 0 (L2)
5	Network Tx bytes	≥ 0 (L2)
6	CPU utilization	≥ 0 (L2)
7	Hard disk bytes read	≥ 0 (L2)
8	Memory utilization	< 0 (L1)
9	Response time	≥ 0 (L2)

ANNEX XIII
OPTIMUM LEVELS OF THE HARD DISK BYTES WRITTEN FACTOR

Factor number	Performance measure	Optimum level
1	Time of CC System Up	≥ 0 (L2)
2	Load map tasks capacity	< 0 (L1)
3	Load reduce tasks capacity	< 0 (L1)
4	Network Rx bytes	< 0 (L1)
5	Network Tx bytes	< 0 (L1)
6	CPU utilization	< 0 (L1)
7	Hard disk bytes read	< 0 (L1)
8	Memory utilization	≥ 0 (L2)
9	Response time	< 0 (L1)

BIBLIOGRAPHY

Abran, A. (2010). *"Software Metrics and Software Metrology"*. Hoboken, New Jersey, John Wiley & Sons Interscience and IEEE-CS Press.

Abran, A. and Buglione, L. (2003) *"A multidimensional performance model for consolidating Balanced Scorecards"* Advances in Engineering Software 34, p. 339-349, Amsterdam Elsevier.

Alexandru, I. (2011). "Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing." IEEE Transactions on Parallel and Distributed Systems **22**(6): 931-945.

Alonso, F., Fuertes, J., Montes, C. and Navajo, R.J. (1998) *"A Quality Model: How to Improve the Object-Oriented Software Process"* International Conference On systems, Man, and Cybernetics. San Diego, CA, USA (IEEE), (11-14 Oct), p.5884-9.

Amazon (2010) *"Amazon EC2"*, http://aws.amazon.com/ec2/#details

Amazon (2010) *"Amazon Web Services"*, http://aws.amazon.com/ec2/#os

Anand, A. (2008) *"Scaling Hadoop to 4000 nodes at Yahoo!"* Yahoo Developer Network, http://developer.yahoo.com/blogs/hadoop/posts/2008/09/scaling_hadoop_to_4000_nodes_a/

Apache Software Foundation (2008) *"Streaming Edits to a Backup Node"*, https://issues.apache.org/jira/browse/HADOOP-4539

Apache Software Foundation (2008) *"ZooKeeper Overview"*, http://hadoop.apache.org/zookeeper/docs/current/zookeeperOver.html

Apache Hadoop (2010) *"BookKeeper Getting Started Guide"*, http://hadoop.apache.org/zookeeper/docs/r3.2.1/bookkeeperStarted.html

Bairavasundaram, L. N., G. R. Goodson, et al. (2008). *"An Analysis of Data Corruption in the Storage Stack"* ACM Transactions on Storage (TOS) **4**(3).

Bass, L., Clements, P., Kazman, R. (2003) *"Software Architecture in Practice"*, Reading, MA: Addison-Wesley, 2nd ed., cited on pp. 34-36.

Bautista, L. and April, A. (2011) *"Sustainability of Hadoop Clusters"*, 1st International Conference on Cloud Computing and Services Sciences (CLOSER 2011), Noordwijkerhout, The Netherlands, 7-9 May, ISBN: 978-989-8425-52-2, pp. 587-590

Bautista, L., Abran, A. and April, A. (2012), *"Design of a Performance Measurement Framework for Cloud Computing"*, Journal of Software Engineering and Applications, Vol. 5 No. 2, pp. 69-75.

Bautista, L., Abran, A. and April, A. (2013), *"A Methodology for Identifying the Relationships between Performance Factors for Cloud Computing Applications"*, 04/2013; Chapter: 15 in book: Software Engineering Frameworks for the Cloud Computing Paradigm, Edition: 2013, XVIII, 365 p. 122 illus., Publisher: Springer, Editors: Zaigham Mahmood, Saqib Saeed. ISBN: 978-1-4471-5030-5

Bautista, L., April, A. and Abran, A. (2014), *"DIPAR: A Framework for Implementing Big Data Science in Organizations"*, Chapter: 8 in Book: Continued Rise of the Cloud: Advances and Trends in Cloud Computing, Edition: 2014, Publisher: Springer, Editor: Zaigham Mahmood. ISBN 978-1-4471-6451-7

Bautista, L., April, A. and Abran, A. (2014) *"Methodology to Determine Relationships between Performance Factors in Hadoop Cloud Computing Applications"*, 4th International Conference on Cloud Computing and Services Sciences (CLOSER 2014), Barcelona, Spain, 3-5 April, ISBN: 000-00-00, pp. 000-00

Bisciglia, C. (2009) *"Hadoop High Availability Configuration"* http://www.cloudera.com/blog/2009/07/hadoop-ha-configuration/

Boehm, B. (1978) *"Characteristics of Software Quality"*, Amsterdam, Holland Elsevier, p. 210.

Buglione, L., and A. Abran (1999) *"Geometrical and statistical foundations of a three-dimensional model of software performance"*, Advances in Engineering Software 30, p. 913-919, Amsterdam Elsevier.

Buglione, L., and A. Abran (2002) *"QEST nD: n-dimensional extension and generalization of a software performance measurement model"* Advances in Engineering Software 33, p. 1-7, Amsterdam Elsevier.

Burgess, M., H. Haugerud, et al. (2002). *"Measuring System Normality"*, ACM Transactions on Computer Systems **20**(2): 125-160.

Cao, J., Chen, Y., Zhang, K., He, Y. (2004) *"Checkpoint in Hybrid Distributed Systems"*, Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)

Carolan, Gaeden, (2005) *"Introduction to Cloud Computing Architecture"*, Sun MicroSystems.

Chen, K., Xin, J., Zheng W. (2008) *"Virtual Clusters: Customizing the Cluster Environment through Virtual Machines"*, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing.

Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riché, T. (2009) *"UpRight Cluster Services"*, SOSP'09, Big Sky, Montana, USA.

Côté, M.-A., W. Suryn, et al. (2006) *"Software Quality Model Requirements for Software Quality Engineering"*, INSPIRE Conference (BSI), London, Uk.

Coulouris, G., J. Dollimore, et al. (2011) *"Distributed Systems Concepts and Design"*, Edinburgh, Addison Wesley.

Cheikhi, L. and A. Abran (2012) *"Investigation of the Relationships between the Software Quality Models of ISO 9126 Standard: An Empirical Study using the Taguchi Method"*, Software Quality Professional Magazine.

Dean, J. and S. Ghemawat (2008) *"MapReduce: simplified data processing on large clusters"*, Communications of the ACM **51**(1): 107-113.

Dhruba, B. (2010, February 19) *"Hadoop Distributed File System Architecture"*, http://hadoop.apache.org/docs/r0.20.2/hdfs_design.html.

Dhruba, B. and A. Ryan (2010) *"A metric to detect persistently faulty machines in Hadoop"*, IFIP Workshoo on Dependable Computing and Fault Tolerance Chicago, Illinois, US.

Dohnert, J. (2012) *"Microsoft's cloud platform Azure suffers outage in Western Europe"*, http://www.v3.co.uk/v3-uk/news/2194727/microsofts-cloud-platform-azure-suffers-outage-in-western-europe.

Dormey, R.G. (1995) *"A model for software product quality"*, IEEE Transactions on Software Engineering 21, 146-162.

Dormey, R.G. (1996) *"Cornering the Chimera"* IEEE Software, vol. 13, no 1, p. 33-43

Foster, I., Zhao, Y., Raicu, I., Lu, S. (2008) *"Cloud Computing and Grid Computing 360-Degree Compared"*, In: Grid Computing Environments Workshop (GCE'08).

Gangadharan, G. R. and D. M. Parrilli (2011) *"Service Level Agreements in Cloud Computing: Perspectives of Private Consumers and Small-to-Medium Enterprise",* Cloud Computing for Enterprise Architectures, Computer Communications and Network, Z. Mahmood and R. Hill. London, Springer-Verlag. **0:** 207-225.

Gottfrid, D., (2007) *"Self-Service, Prorated Supercomputing"*, http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/?scp=1&sq=self%20service%20prorated&st=cse

Grimshaw, A., Morgan M., Merrill, M., et al. (2009) *"An Open Grid Services Architecture Primer"*, published by the IEEE Computer Society pp. 24-31

Hadoop, A. F. (2014). *"What Is Apache Hadoop?"*, from http://hadoop.apache.org/.

Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A., Katz, R., Shenker, S., Stoica, I. (2010) *"Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center"*, EECS, University of California at Berkeley, Technical Report No. UCB/EECS-2010-87

ISO/IEC (1999) *"ISO/IEC 14756: Measurement and rating of performance of computer-based software systems",* Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2001) *"ISO/IEC 9126-1: Software Engineering-Software product quality-Part 1: Quality model"*, Geneva, Switzerland: International Organization for Standardization.

ISO/IEC (2005) *"ISO/IEC 19759:Software Engineering — Guide to the Software Engineering Body of Knowledge (SWEBOK)",* Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2006) *"ISO/IEC CD 25030:Software engineering – Software product quality requirements and evaluation (SQuaRE) – Quality requirements"*, Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2007) *"ISO/IEC 25020: Software Engineering –Software quality requirements and evaluation (SQuaRE) - Quality measurement – Measurement reference model and guide",* Geneva, Switzerland, International Organization for Standardization**:** 18.

ISO/IEC (2008) *"ISO/IEC 15939:2007 Systems and software engineering — Measurement process"*, Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2008) *"ISO/IEC:Basic and General Concepts and Associated Terms, VIM"*, International Vocabulary of Metrology B. I. d. P. e. Mesures. Geneva, Switzerland, ISO/IEC Guide 99-12.

ISO/IEC (2011) *"ISO/IEC 25010: Systems and software engineering – Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models"*, Geneva, Switzerland, International Organization for Standardization**:** 43.

ISO/IEC (2012) *"ISO/IEC JTC 1 SC38:Cloud Computing Overview and Vocabulary"*, Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2013) *"ISO/IEC 25023: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality"*, Geneva, Switzerland, International Organization for Standardization.

ISO/IEC (2013) *"ISO/IEC DIS 17789: Information technology - Cloud Computing - Reference Architecture"*, Geneva, Switzerland, International Organization for Standardization.

Jackson, K. R., L. Ramakrishnan, et al. (2010) *"Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud"*, IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), Washington, DC, USA, IEEE Computer Society.

Jain, R. (1991) *"The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling"*, New York, United States, John Wiley & Sons - Interscience.

Jin, H., S. Ibrahim, et al. (2010) *"Tools and Technologies for Building Clouds",* Cloud Computing: Principles, Systems and Applications, Computer Communications and Networks. London, Springer-Verlag. **0:** 3-20.

Kantardzic, M. (2011) *"DATA MINING: Concepts, Models, Methods, and Algorithms"*, Hoboken, New Jersey, IEEE Press & John Wiley, Inc.

Kira, K. and L. A. Rendell (1992) *"The Feature Selection Problem: Traditional Methods and a New Algorithm"*, The Tenth National Conference on Artificial Intelligence (AAAI), San Jose, California.

Kolakowski, N. (2009) *"Microsoft's cloud azure service suffers outage"*, Retrieved from http://www.eweekeurope.co.uk/news/microsoft-s-cloud-azure-service-suffers-outage-396

Kourpas, E. (2006) *"Grid Computing: Past, Present and Future – An Innovation Perspective"*, IBM white paper

Kramer, W., J. Shalf, et al. (2005) *"The NERSC Sustained System Performance (SSP) Metric"*, California, USA, Lawrence Berkeley National Laboratory.

Li, B., L. Gillam, et al. (2010) *"Towards Application-Specific Service Level Agreements: Experiments in Clouds and Grids"*, Cloud Computing: Principles, Systems and

Applications, Computer Communications and Networks. London, Springer-Verlag. **0:** 361-372.

Lin, J. and C. Dyer (2010) *"Data-Intensive Text Processing with MapReduce"*, University of Maryland, College Park, Manuscript of a book in the Morgan & Claypool Synthesis Lectures on Human Language Technologies.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977) *"Factors in software quality"*, Griffiths Air Force Base, N.Y. : Rome Air Development Center Air Force Systems Command.

Mei, Y., L. Liu, et al. (2010) *"Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud"*, IEEE International Conference on Cloud Computing, CLOUD 2010, Miami, FL, USA, IEEE.

Pfister, G. (2009) *"How hardware virtualization works"*, http://www.isgtw.org/?pid=1002636 Proceedings of CloudViews, Cloud Computing Conference 2009

Prasad, R. B., E. Choi, et al. (2010) *"A Taxonomy, Survey, and Issues of Cloud Computing Ecosystems"*, Cloud Computing: Principles, Systems and Applications, Computer Communications and Networks. N. A. a. L. Gillam. London, Uk, Springer-Verlag**:** 21-46.

Pressman, R. S. (2001) *"Software Engineering: A practitioner's approach"*, Boston: McGraw-hill (5th ed.).

Rao, A., R. Upadhyay, et al. (2009) *"Cluster Performance Forecasting Using Predictive Modeling for Virtual Beowulf Clusters"*, Springer-Verlag Berlin Heidelberg 456-461.

Ravanello, A., Desharnais, JM., Bautista, L., April, A. and Gherbi, A. (2014) *"Performance measurement for cloud computing applications using ISO 25010 standard characteristics"*, To appear - Joint Conference of the 24rd International Workshop on Software Measurement & 9th International Conference on Software Process and Product Measurement - IWSM-MENSURA 2014, Rotterdam (Netherlands), Oct. 6-8, 2014

Schroeder, B., E. Pinheiro, et al. (2009) *"DRAM Errors in the Wild: A Large-Scale Field Study"*, SIGMETRICS/Performance, Seattle, WA, USA, ACM.

Smith, D., Q. Guan, et al. (2010) *"An Anomaly Detection Framework for Autonomic Management of Compute Cloud Systems"*, IEEE 34th Annual IEEE Computer Software and Applications Conference Workshops (COMPSACW)**:** 376-381.

Stanoevska-Slabeva, K., Wozniak., T. (2010) *"Grid and Cloud Computing"*, pp. 23-45 Springer.

Sterling, T. (2000) *"An Introduction to PC Cluster for High Performance Computing"*, California Institute of Technology and NASA Jet Propulsion Laboratory, USA, Cluster Computing White Papers

Red, B., Junqueira, F. (2008) *"A Simple Totally Ordered Broadcast Protocol"*, In proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware (LADIS), Yorktown Heights, New York, vol. 341:2008

Taguchi, G., S. Chowdhury, et al. (2005) *"Taguchi's Quality Engineering Handbook"*, John Wiley & Sons, New Jersey.

Tanenbaum, A., Steen V. (2002) *"Distributed Systems, Principles and Paradigms"*, Prentice Hall 2002, 2nd edition.

Trivedi, K. S. (2002) *"Probability and Statistics with Reliability, Queuing and Computer Science Applications"*, New York, U.S.A., John Wiley & Sons, Inc.

Venner, J. (2009) *"Pro Hadoop"*, New York, USA, Springer-Verlag.

Wang, F., Dong, B., Qiu, J., Li, X., Yang, J., Li, Y. (2009) *"Hadoop High Availability through Metadata Replication"*, CloudDB'09 ACM, Hong Kong, China.

White, T. (2012) *"Hadoop: The Definitive Guide"*, Gravenstein Highway North, Sebastopol, CA, O'Reilly Media.

Xen Org (2008) *"Xen Architecture Overview"*, http://wiki.xen.org/xenwiki/XenArchitecture?action=AttachFile&do=get&target=Xen+Architecture_Q1+2008.pdf

Xen Org (2009) *"Xen in the Cloud"*, http://www.xen.org/files/Marketing/XeninCloud.pdf

Xing, L. and A. Shrestha (2005) *"Distributed Computer Systems Reliability Considering Imperfect Coverage and Common-Cause Failures"*, 11th International Conference on Parallel and Distributed Systems, Fuduoka, Japan, IEEE Computer Society.

Yahoo!, I. (2012) *"Yahoo! Hadoop Tutorial"*, Retrieved January 2012, from http://developer.yahoo.com/hadoop/tutorial/module7.html#configs.