A MODEL OF THE RELATIONSHIPS BETWEEN AVAILABILITY MECHANISMS AND OUTAGE SOURCES IN CLOUD COMPUTING

BY

MBOGHOLI JOHN SAUL MSAGHA

PHD/CI/00116/2013

A THESIS SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE OF DOCTOR OF PHILOSOPHY IN INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING AND INFORMATICS

MASENO UNIVERSITY

DECLARATION

Student's Declaration

This thesis entitled **"A Model Of The Relationships Between Availability Mechanisms And Outage Sources In Cloud Computing"** is my original work and has not been presented to any other University for examination.

Signature:_____ Date _____

Mbogholi John Saul Msagha

REG. NO. PHD/CI/00116/2013

Supervisors' Declaration

The undersigned certify that we have read and hereby recommend for acceptance of Maseno University a thesis entitled **"A Model Of The Relationships Between Availability Mechanisms And Outage Sources In Cloud Computing".**

Signature:_____ Date _____

Dr. Henry Okora Okoyo

Department of Computer Science

Maseno University

Signature:_____ Date _____

Dr. Okoth Sylvester J. McOyowo

Department of Computer Science

Maseno University

ACKNOWLEDGEMENT

Without the Grace of God this thesis would never have been completed; I thank Him and acknowledge Him first as the Light and guide of this study. I especially acknowledge the input, guidance and mentorship of my supervisors Dr Henry .O. Okoyo and Dr Sylvester J. O. McOyowo. Their input and words truly humbled me, and working with them is an experience I'll carry throughout my life. May God's blessings be upon both of you.

I would like to thank Raysa Oliveira of the University of Beira Interior (Portugal). She spent invaluable time helping me to understand the workings of the simulator used in this study from that long distance away. I thank Dr Geoffrey Serede Sikolia, who provided insight into my thesis, provided encouragement throughout and always made time to listen to my ideas; a true friend. My colleagues and many more I cannot mention here. I also thank my classmates whom we have encouraged each other during this journey: Eric Oteyo, Samuel Ndichu and James Obuhama.

A special thank you to the following for their immeasurable contributions and support: my parents, Mrs M Gitau, my beloveds Eva, Tiffany, Tamia and Ted. My sister Joyce, and Esther, Tony, Minnie and Irene. I also thank my friend Simon K Chibole, and many friends who supported me in one way or the other by encouragement and/or support.

Since it is impossible to exhaust this list, I am truly grateful to all those who have supported me in any way, be it spiritual, moral, or financial. May God bless you all.

DEDICATION

Every journey begins with a single step. I would like first to dedicate this thesis to the two people who started me on this journey by holding my hands and taking me to school. They have been a pillar of encouragement and source of inspiration. They have made personal sacrifices, some of which I am all too aware, and others I might never know about, to help me reach this point in my life. They have encouraged me and urged me on especially in those moments that I truly need it. I love you and salute you. Thank you to my parents Mr Gabriel Mwaingo Mbogholi (who went to be with God on Christmas day 2019) and Mrs Anastasia Chanya Mbogholi. On a special note I acknowledge and thank Mrs Mary Gitau for her encouragement and support throughout this program.

I dedicate this to my cherished wife Eva. She stood by me throughout this journey, encouraged me in those moments when I did not feel particularly encouraged, and made personal sacrifices to help see me through this program. Thank you for the sacrifices, patience and encouragement. To Tiffany, Tamia and Ted, I also dedicate this work to you.

To the loving memory of Teddy Mbogholi Mwaingo, friend and brother, who went to be with God in 2008. You were a source of encouragement, and though you did not live to see this day I am sure you're smiling in happiness for me from Heaven. I miss you.

And to the loving memory of Faustine Mwaighacho Mwadilo, dear friend and akofe, who went to be with God in 2017. A true confidant that I also wish was here to see this day. May you celebrate with us from Heaven. I miss you.

ABSTRACT

The use of cloud computing has been growing exponentially since its inception. Availability of the cloud, however, has been a problem for users and Cloud Service Providers (CSPs) alike; outages have been on the rise. This problem could be attributed to the fact that engineers building Availability Mechanisms (AMs) and those studying outage causes do not work collectively. The general objective of the study was to develop and evaluate an availability mechanism model for service outages in cloud computing environments. The study specifically sought to: identify the causes of outages in cloud computing environments, identify availability mechanisms in use in cloud computing environments, formulate a model that establishes correspondences between AMs and outages, and evaluate the performance of the model by measuring its service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters. A model was developed called the Ferris Wheel of Availability (FWA) model. The model was developed by relating AMs to outage causes, with AMs being conjugate in nature in relation to the respective outage causes. There were seven categories of AMs and seven categories of outage causes; AMs were categorized as cluster management, component redundancy, limit detection policy, checkpointing, node management, Active-X variant and fault tolerance. Outage causes were categorized as configuration issues, hardware issues, resource exhaustion, security issues, node failures, network issues and natural disasters. Testing of the model was done using CloudSim, a discrete, deterministic simulator that allows users to set up their customized configurations and run them in it. The simulator was configured to run each outage cause individually and the applicable AMs were then injected simultaneously and output recorded. Each outage cause had two AMs, and the findings confirmed the effectiveness of the proposed model structure in increasing service availability at infrastructure level. Key findings were that checkpointing is not effective as an AM against resource exhaustion, and that effective management of a cluster results in effective management of the nodes in it. It was not conclusive as to whether limit detection policy was effective as an AM against security issues. The study also suggested that limit detection policy be renamed limit prevention policy. The study introduced a new availability parameter called execution availability and recommended its use together with service availability in predicting overall availability at infrastructure level. The key contributions of the study were: development of the FWA model that establishes correspondences between AMs and outage causes since a model that establishes these correspondences had not been developed before; discovery of the relationships between AMs and outage causes based on simulation tests and consequent analysis; and introduction of an availability parameter called execution availability that measures the ratio of tasks allocated versus tasks executed. It is recommended to study the feasibility of merging two or more simulators to achieve results which were inconclusive using one simulator; an extension to the simulator in use may also be investigated. The use of the FWA model at CSP level is also recommended as it assists analysts and developers to build for availability from the very foundation as opposed to adapting a wait-and-see attitude in countering outages as they occur. The outcome of the study points to suggest that the application of the FWA model in a cloud computing infrastructure has the potential to increase availability in the cloud.

TABLE OF CONTENTS

CONTENT **PAGE**

ABBREVIATIONS

AM	Availability Mechanism
API	Application Programming Interface
AWS	Amazon Web Service
BYOC	Bring Your Own Computer
BYOD	Bring Your Own Device
CAP	Consistency, high Availability and resilience to network Partitions
CIS	Cloud Information Service
CLOUDS	Cloud Computing and Distributed Systems
CPU	Central Processing Unit
CSA	Cloud Security Alliance
CSP	Cloud Service Provider
CTMC	Continuous Time Markov Chain
DC	Datacenter
FT	Fault Tolerance
FWA	Ferris Wheel of Availability
HA	High Availability
HA-OSCAR	High Availability Open Source Cluster Application Resource
IaaS	Infrastructure-as-a-Service
IBM	International Business Machines
ISP	Internet Service Provider

IWGCR	International Working Group on Cloud Computing Resiliency
LBVFT	Load Balancing for Virtualization and Fault Tolerance in cloud computing
MIPS	Millions of Instructions Per Second
NIST	National Institute of Standards and Technology
OSCAR	Open Source Cluster Application Resource
PaaS	Platform-as-a-Service
PE	Processing Element
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
SaaS	Software-as-a-Service
SDA	Software Defined Availability
SLA	Service Level Agreement
TTC	Time To Complete
TTF	Time To Fail
VFT	Virtualization and Fault Tolerance
VM	Virtual Machine

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

Cloud computing is a technology which enables users to access various services online without having to have them resident on their devices. For individuals this may include services related to social networking or access to media such as games and music; for businesses it includes services such as infrastructure, storage or application development platforms. These services are offered in different layers of the cloud and are referred to as IaaS (infrastructure as a service), PaaS (platform as a service) and SaaS (software as a service). It saves costs mostly for businesses as it means a business does not invest in infrastructure as it would ordinarily have to; thin clients and an internet connection is all that is needed. This has given birth to concepts such as BYOD (Bring Your Own Device) within the business world where staff use their own devices at work such as smartphones and tablets courtesy of availability of services via the cloud, and even BYOC (Bring Your Own Computer) implementing the same concept. The services via cloud are provided by Cloud Service Providers (CSPs) who act like the traditional Internet Service Providers (ISPs) the difference being that CSPs offer access to cloud services while ISPs (Internet Service Provider) provide access to the Internet. Cloud computing is a technology that has been grasped worldwide and usage has been growing steadily over the years. Table 1 shows the growth forecast for cloud computing services globally:

Table 1 Worldwide Public Cloud Services Revenue Forecast (Billions of U.S. Dollars)

	2016	2017	2018	2019	2020
Cloud Business Process Services (BPaaS)	39.6	42.2	45.8	49.5	53.6
Cloud Application Infrastructure Services (PaaS)	9.0	11.4	14.2	17.3	20.8
Cloud Application Services (SaaS)	48.2	58.6	71.2	84.8	99.7
Cloud Management and Security Services	7.1	8.7	10.3	12.0	13.9
Cloud System Infrastructure Services (IaaS)	25.4	34.7	45.8	58.4	72.4
Cloud Advertising	90.3	104.5	118.5	133.6	151.1
Total Market	**219.6**	**260.2**	**305.8**	**355.6**	**411.4**

Source: Gartner (October 2017)

Table 1 show the number of users migrating to the cloud has been growing at an exponential rate over the years. In the 2013 survey on the future of cloud computing conducted by GigaOM Research, North Bridge Venture Partners and 57 collaborating organizations, it was noted that even though security is still the top inhibitor to the adoption of cloud computing its significance as an inhibitor reduced by 9% from the previous year. However, reliability was among the top inhibitors as respondents showed concern and the need to have an "always-on" services infrastructure (businesswire.com, 2014). Other concerns include data availability, data management, resource allocation and load balancing (Ahuja & Mani, 2012). Recent outages by top vendors such as Google (2013, 2014) have not helped in increasing confidence on the reliability of this technology despite the continued uptake of the service; a paradox in itself. Further statistics show that outages incidents have been on the increase for the past three years (Gagnaire, Diaz, Coti, & Cerin, 2012; Christophe et al., 2013, 2014). Figure 1 shows the extent to which outages have been on the increase between 2010 and 2016.

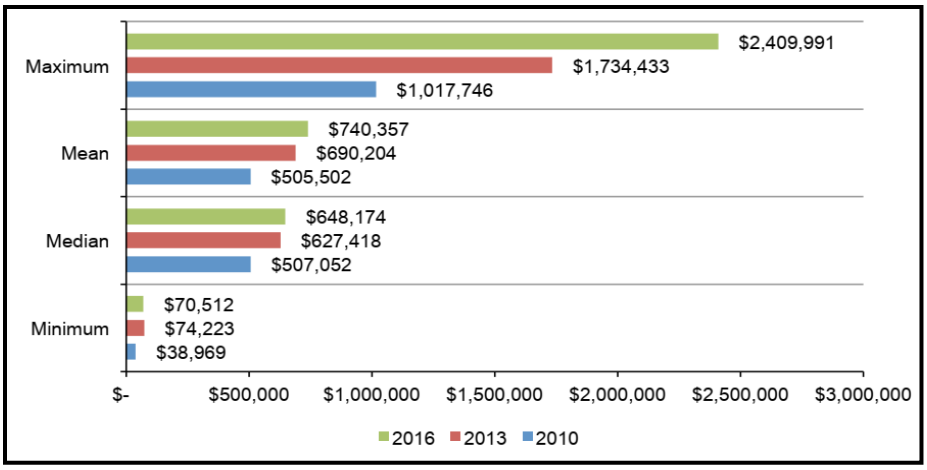

Figure 1 Data center outage costs (Ponemon, 2018)

Unfortunately most available literature points out major concerns of cloud service providers and users, and offers a solution via a concept known as recovery oriented computing (Nasuni, 2010; Satria, Park, & Jo, 2017). Recovery oriented computing focuses on fault tolerance in the cloud infrastructure without addressing the causes of the outages in the first place, implying a reactive rather than a proactive approach.

The current availability mechanisms that have been developed by researchers address the different ways in which to keep the cloud up and running through a redundant variant, or through some virtualization or other software implementation technique; these are discussed in chapter two of this thesis. However, none of these techniques have been designed to deal specifically with the different identified causes of outages. The study aimed to investigate whether it can be presumed that this generic approach to development of availability mechanisms by industry (researchers as well as service providers), rather than the lack of addressing specific outage causes may be the reason downtime has gone from bad to worse in the past three years (IWGCR, 2013, 2014; Cloudharmony, 2015, 2016, 2017). A series of descriptive studies was undertaken to better understand the relationship between these factors resulting in a proactive model that correlates outage causes to different classes of availability mechanisms and thus provide the genesis of building for proactivity rather than reactivity.

1.2 Statement of the Problem

Downtime and outages have been on the increase over the past six years (2010 to 2016), negatively impacting on productivity (Ponemon, 2018). The overall research problem that was addressed in this study is that current identified availability mechanisms have not helped in stemming this upward trend, and little has been done to address the real causes of these outages.

1.3 Research Objectives

The general objective of the study was to develop and evaluate an availability mechanism model for service outages in cloud computing environments. In so doing the study sought to fulfill the following specific objectives:

To:

1. Identify the causes of outages in cloud computing infrastructures.
2. Identify availability mechanisms in use in cloud computing infrastructures.
3. Formulate a model that establishes correspondences between AMs and outage causes.
4. Evaluate the performance of the model by measuring its service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters.

1.4 Research Question

The study aimed to answer the following research question:

What are common traits of cloud infrastructures not experiencing five 9's (99.999%) availability, and how can these commonalities be used to aid the cloud infrastructure providers in prevention of these outages?

In order to address the main research question, the study sought to answer the following sub-questions:

1. What are the current causes of outages in cloud computing environments?
2. What constitutes availability at the cloud infrastructure level, and which mechanisms are in place currently to achieve availability?
3. Which models map availability to outage causes?
4. How can the proposed model be used to enhance overall availability in a cloud environment?

1.5 Significance of the Study

The findings of the research added to the knowledge and understanding of the subject of availability in cloud computing. This study was significant in the sense that it:

1. Identified the factors that lead to downtime at infrastructure level in cloud environments.
2. Identified the current availability mechanisms in place in cloud computing infrastructures.
3. Developed a new model for to inform enhancing availability at cloud infrastructure level.
4. Demonstration of viability of the proposed model in increasing availability at cloud infrastructure level via simulations.
5. Generated greater awareness among cloud infrastructure providers and the larger IT community on what constitutes availability in cloud environments by suggesting the use of a combination of availability parameters, namely execution availability and service availability.

1.6 Scope of the study

The scope of the study was limited to infrastructure level at the CSP. This is the one concerned with providing the services required by the higher layers. The users have no access to this level; only the CSP does. The configuration of a cloud may be heterogeneous or homogenous. This

study was focused on heterogeneous infrastructures as opposed to homogeneous infrastructure. Crago et al. (2011) described heterogeneous computing in further detail. The reason for this is that as cloud computing grows and is embraced by more CSPs and users alike, the issue of vendor lock-in becomes a contentious issue. Homogeneity in cloud computing means a CSP uses everything in its infrastructure from a single vendor, including the software. This does not give room for the CSP to develop customized configurations. In a heterogeneous environment the CSP is free to use hardware from different vendors (and with different hardware configurations), and is now free to use open source software or implement software from a vendor of their choice. Indeed Cao, Simonin, Cooperman, and Morin (2014) argued for the adoption of cloud-agnostic architectures.

1.7 Limitations of the study

The study was carried out using simulations done in CloudSim toolkit. It was not possible to use a real environment as it was neither practical nor cost effective.

1. The simulation tool in itself had the limitation of not providing a GUI meaning the user had to define the scenario and write the relevant Java code to enact the scenario.
2. The second limitation of the simulator was in the fact that it is a discrete, deterministic simulator. This meant that observations were made when changes occurred in the system state. When tasks were sent to the simulator it was not possible to observe each task independently as it was run; rather the output alerted the user as to the identity of the executed tasks and the overall time it took to execute each task. It would have been desirable to be able to view singular events and states from the beginning of the simulation to the end rather than knowing when a task begins and when it ends.
3. The simulator was not able to simulate resource exhaustion and a denial of service (DoS) attack; the existing load balancing policies to prevent resource exhaustion were found to be valid for evaluation purposes, while no suitable means was found to investigate the DoS attack.
4. Another limitation of the simulator was that it was not possible to observe actual VM migration taking place due to its discrete nature; the inference can only be made due to

the fact that in the given scenarios the hosts were failed and when VM migration was configured the tasks were completed. The use of FTCloudSim, an extension of CloudSim enabled checkpointing to be enacted and observed, since CloudSim on its own could not simulate checkpointing.

1.8 Thesis Outline

This thesis is divided into five chapters:

1. The first chapter has introduced the importance of cloud computing in terms of usage globally. The chapter has gone on to demonstrate how the usage of cloud computing has grown steadily over the years. However, cloud outages have been a thorn in the flesh for CSPs and users like. The statement of the problem is that despite having all the outages little has been done to study the actual causes of these outages. It is proposed that when AMs address specific outage causes then availability at infrastructure level will be increased. The study proposed to develop a model that will relate AMs to outage causes at infrastructure level; the general objective of the study was to develop and evaluate an availability mechanism model for service outages in cloud computing environments.
2. Chapter two describes the different cloud architectures, benefits provided by using the cloud, cloud deployment models, and the current literature on outage causes and AMs. It also examines availability as defined by different authors. Further it does a critique of the current AMs and relates this to the gap identified in the statement of the problem in chapter one. The chapter then introduces the proposed model to fill the gap, and describes its characteristics, merits and limitations. It also provides the logic behind the development of relationships between AMs and outage causes. This chapter fulfills the first three specific objectives of the study.
3. Chapter three describes the methodology that was used to test the model; several scenarios were configured and run in the simulator to test the effect of the different AMs on the outage causes in a proactive manner. Part of the fourth specific objective is fulfilled in this chapter.
4. Chapter four documents the results of the tests performed in chapter three and discuss them, together with the relevance of the findings in relation to the literature and the

objectives of the study. The remainder of the fourth specific objective is completed in this chapter.

5. Chapter five summarizes the findings of the study, shows how the objectives of the study were met, concludes, and gives recommendations and suggestions for future research.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter examines the current literature on outage causes and AMs in cloud computing infrastructures, and goes further to posit the need to relate the AMs to different outage causes. It begins by defining the different cloud computing benefits, architectures and deployment models to enable better understanding of cloud computing services as a whole. The current literature on outage causes is described by various authors, and the current AMs. A section on how different authors have defined availability is then discussed. The chapter offers a critique of the AMs, showing that the gap that exists currently is based on the fact that these AMs do not address specific outage causes. It finally opines that when AMs can address specific outage causes then availability in cloud infrastructures can be improved, hence the need for a model that will specifically relate the outage causes to the AMs.

2.2 Cloud computing benefits, architecture and deployment

Pallis observed that cloud computing is an evolving paradigm (as cited in Mohapatra, Smruti & Mohanty, 2013) that has changed business and social Internet paradigms as much as the advent of Web 2.0 did (Msagha, 2012). Users benefit when using cloud computing technology due to its global availability (via Internet), scalable usage (use only what you need), convenience, and the seemingly unlimited resources offered by providers. Further the fact that the users can access these resources directly without direct intervention of the service provider, adds to the convenience. Ideally this means that this service be available on-demand, always and in a convenient manner to all its users and of course with minimal service provider interaction (NIST, 2018). Consequently, therefore, no organization using cloud services ought to have to worry about availability of the service or the probability of the infrastructure crashing.

Usage of the cloud in today's digital age is of a stupendous nature; there are cloud communities all over the world in digisphere and statistically more and more organizations are abandoning traditional physical server environments for virtualized environments (Grispos, Storer & Glisson,

2012). Arising from this one can only imagine the far reaching implications of service failure at the infrastructure level. Consider an organization called NgomaPicha that uses the cloud for storing socially-in-demand products like music and movies. The organization provides downloads from users and communities at a fee. They have received a 24 hour head start in rights for the latest album by John Henry a teenage superstar idol; everyone wants the album first and the downloads begin. Twelve hours later the service is down for a tragic 5 hours due to failure at the CSP level. Picture the domino effect from the CSP to Michelle, one of JH's (as he is known to his fans) biggest fans, who had just begun downloading the album. NgomaPicha lose their head start and countless dollars in revenue, and Michelle is not likely to download from their site over a while as she pouts and smarts due to this inconvenience. Further, the CSP might face a legal suit from NgomaPicha and hundreds, possibly thousands of their clients, *ad infinitum*.

2.2.1 Cloud computing benefits

Cloud computing is the direction that the industry is shifting to due to the paradigm shift away from physical resources to virtualized services (Grispos, Storer & Glisson, 2012). Of all its advantages and benefits the strength of scalability is arguably the cloud's biggest strength; users get to use only the resources they need and they can scale up and down as the need arises, paying only for what they use. Other benefits offered by the cloud include (Caldarelli, Ferri, & Maffei, 2017):

- Lower total cost of ownership for organizations as they do not need to invest in infrastructure; the cloud provides this for them
- Always on, always available services
- Scalability implies no wasted capacity
- Revenue models which benefit the consumer such as pay-as-you-go and try-before-you-buy
- Economic disaster recovery solutions

- No additional software required by users
- Online deployment and development tools

The cloud computing paradigm has steadily gained momentum globally with more users enjoying the benefits of the different “IT”-as-a-service models.

2.2.2 Cloud Architecture

The cloud services architecture provides different services to the users. The main cloud services are shown in figure 2.1

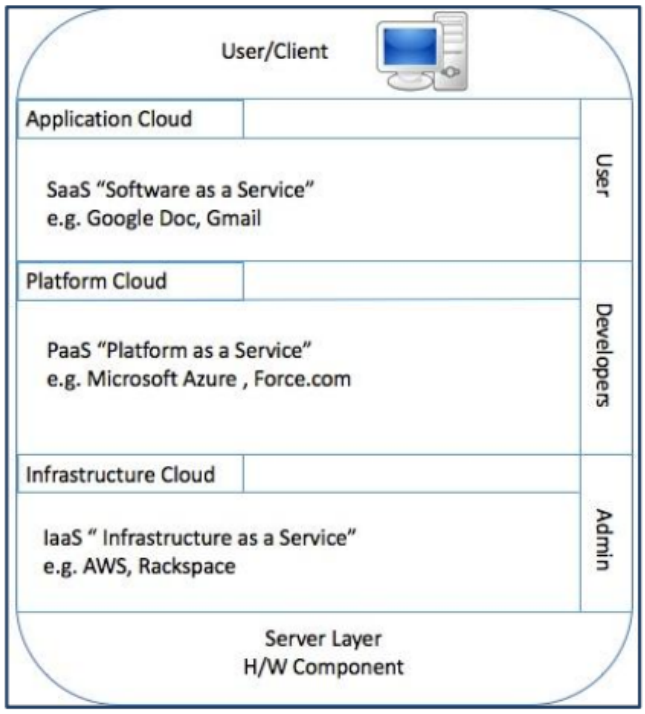

Figure 2.1 Cloud infrastructure services (Bajaber, AlQulaity, & Alotaibi (2017))

- Infrastructure-as-a-service (IaaS): this is the infrastructure at the lowest level of the model that offers a comprehensive infrastructure consisting of servers, storage, software and networks which are placed at the disposal of the user, for example, Amazon S3

- Platform-as-a-service (PaaS): a mid-level service targeted at developers that provides them with a platform for development and deployment of applications, for example, Microsoft Azure

- Software-as-a-service (SaaS): at the top of the stack in the model and it provides users with different software applications over the Internet that they can use, for example, SalesForce CRM

2.2.3 Cloud deployment models

Additionally these services are deployed using three cloud deployment models as shown in figure 2.2

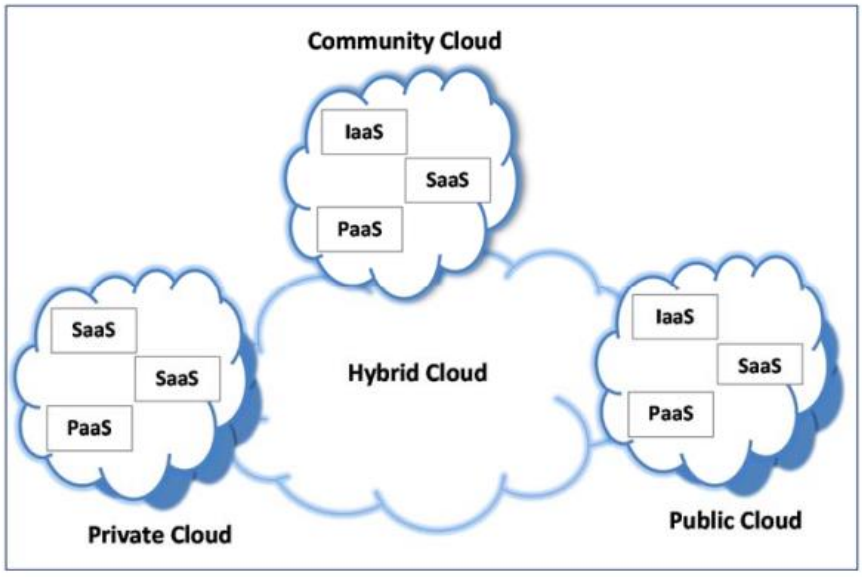

Figure 2.2 Cloud deployment models (Khalil, Khreishah, & Azeem, (2014))

- Public cloud: these are clouds provided by third parties which are available to all users who access them usually for a fee.

- Private cloud: these are clouds built by organizations in their private capacity and hosted and managed by their respective ICT departments.

- Hybrid cloud: this is a mix or hybrid of a public and private cloud model. It is the integration of on-premises IT infrastructures and internal cloud applications with applications and information deployed to a service provider (a.k.a. cloud bursting) either on a temporary or permanent basis (IBM, 2010)

The choice of deployment model is dependent on the needs of the enterprise.

Figure 2.3 demonstrates how the cloud relates to the datacenter and different types of users, namely internal and external users. A datacenter provides both public and private clouds. External users can only access the public cloud, while internal users can access the private cloud directly and the public one through a process called cloud bursting. Cloud bursting occurs when the internal users need services from the public cloud, and thus they "burst" through the private cloud and on into the public cloud. At this point both the private and public cloud are available to the internal users.

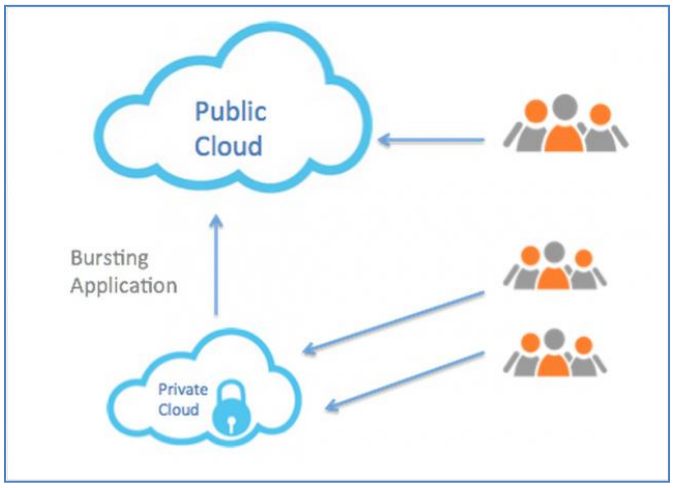

Figure 2.3 Relationship between cloud and its users and components (ctl.io, accessed Nov 2019)

2.3 Cloud outages

A cloud outage, just like a power outage, is a period of time (short or long) during which cloud services are unavailable. As time has gone by the number of cloud outages at service provider level has gone up. The Cloud Security Alliance (CSA) Cloud Vulnerabilities Working group (2012) pointed out that the number of cloud vulnerability incidents doubled between 2009 and 2011. More recent data collection by The International Working Group on Cloud Computing Resiliency (IWGCR) (Ko, Lee & Rajan, 2013) shows an increase of 70% in reported downtime across the service providers sampled from 240 hours to 410 hours. Further, in their latest report covering 2013, IWGCR (Christophe et al., 2014) statistics from the same service providers show a whooping increase of 218% in downtime from 410 hours in 2012 to 1305.21 hours in 2013. This is definitely a worrying trend all factors notwithstanding. It is exacerbated further by the fact that most service providers do not disclose the cause of these outages; in fact, the CSA Cloud Vulnerabilities group observed that 25% of reported outages do not disclose the cause of the outages. However, the same group opined that most service providers had begun reporting outages since 2010 nevertheless.

To give an idea what the cost of outages is to business consider the following facts:

Amazon's outage of January 2013 which lasted 49 minutes costed them upwards of US $ 4 million in sales. In August of the same year a 30 minute outage cost them an estimated US $ 66240 per minute based on their 2012 sales. (forbes.com, accessed 11/08/14)

Google's 5 minute outage in 2013 cost them an estimated US $ 500000, again based on their quarter 2 2013 revenue. This issue alone led to a 40% drop in Internet traffic worldwide (forbes.com, venturebeat.com, cnet.com, theregister.co.uk, accessed 11/08/14)

Small and Medium Businesses are also affected significantly by downtime; their losses can run from $ 2000 to as much as $ 30000. According to Patterson (2002) the average cost of one hour of downtime can be calculated as follows:

Let Employee costs per hour = E_c;
Let fraction of employees affected by outage = E_f;
Let Average Income per hour = I_a;
Let Fraction income affected by outage = I_o;

Let average cost of downtime per hour = C;

$$C = E_C \times E_f + I_a \times I_o$$

Other factors that may increase this cost include legal issues and customer migration.

2.3.1 Causes of cloud outages

There are various causes of cloud outages and numerous inherent factors that cause risk of the cloud infrastructure failing. Though it has been noted in section 2.3 that the CSA Cloud Vulnerabilities group had observed more opening up by CSPs on causes of outages since 2010 nevertheless there is still reason to have more disclosure from these players. However, some of the risks and causes of cloud outages that have been investigated are discussed in this section.

In her IBM white paper on mitigating risks of cloud resource exhaustion outages, Myerson (2013) noted the following as the types of failures that trigger outages: leap year failure, numerically unstable algorithm, resource optimization failure, threshold policy implementation failure, hypervisor failure, and virtual desktop failure.

Between 2009 and 2012 the Cloud Security Alliance proposed different threats as the highest category of threats to cloud computing. Table 2.1 shows the top 7 cloud computing threats identified by the CSA.

Table 2.1 Top CSA Cloud Computing threats (CSA, 2010)

No.	CSA Top Threat
1	Abuse and Nefarious Use of Cloud Computing
2	Insecure Interfaces and APIs
3	Malicious Insiders
4	Shared Technology Issues
5	Data Loss or Leakage
6	Account or Service Hijacking
7	Unknown Risk Profile

Table 2.2 shows additional identified cloud computing threats.

Table 2.2 Additional cloud computing threats (Cloud Vulnerabilities Working Group, 2013)

New Threat	Cause of Vulnerability	Severity of Disruption
8	Hardware Failure	Hardware, from switches to servers in data centers, may fail making cloud data inaccessible.
9	Natural Disasters	Based on the geographical location and the climate, data centers may be exposed to natural disasters such as lightning, storms, and earthquakes, which can affect the cloud services.
10	Closure of Cloud Service	Disputes with the cloud provider or non-profitability of the cloud service may result in the termination of the cloud service, leading to data loss unless end-users are legally protected.
11	Cloud-related Malware	Attackers can use cloud-specific malware, such as bugs and Trojans, to either infiltrate or corrupt the network.

Bigelow (2011) interviewed several leading datacenter acknowledged experts and these were the causes of datacenter downtime expressed:

- Human errors
- Network protocols and network hardware (or a combination of both)
- Pilot errors
- Network servers

Li, Liang, Brien, and Zhang (2013) also performed a survey on cloud outages and classified causes of outages shown in figure 2.4

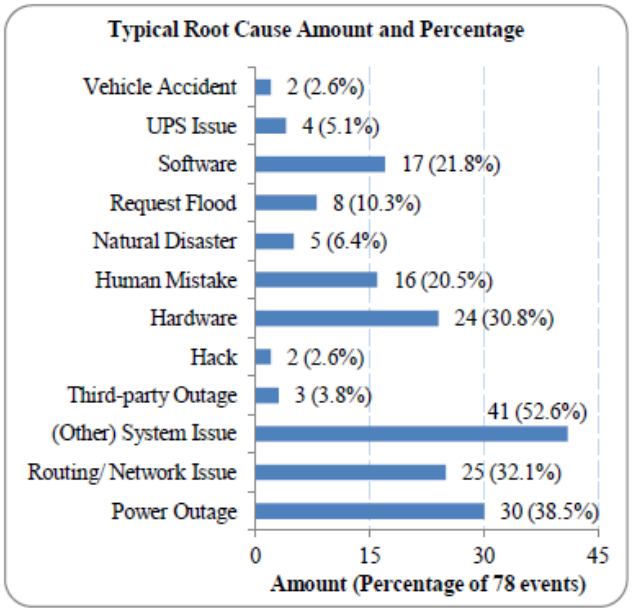

Figure 2.4 Causes of cloud outages (Li et al. (2013))

In their empirical study on network failures and their impact on services Potharaju and Jain (2013) studied network failures at both inter-data center (DC) and intra-data centre (DC). Though their study was the first of its kind in terms of the geographical size and was focused more on the networking aspects within and across datacenters they found the cause of network failures in the respective DC environments as shown in figure 2.5

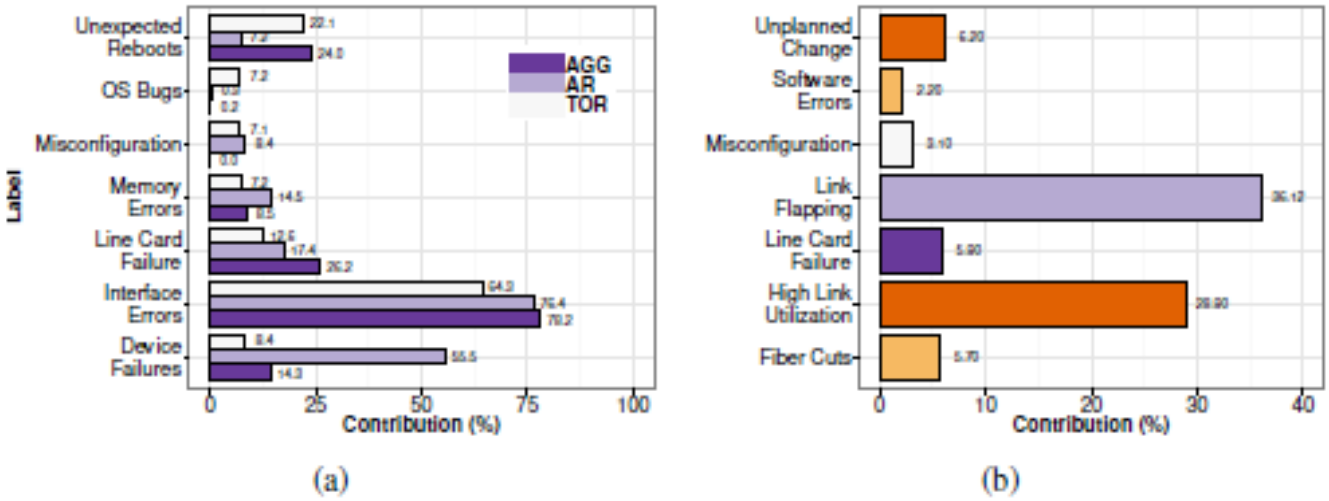

Figure 2.5 Causes of network failures at (a) Inter-DC level and (b) Intra-DC level (Potharaju and Jain (2013))

A summary of the impact of these failures on service provision is shown in figure 2.6

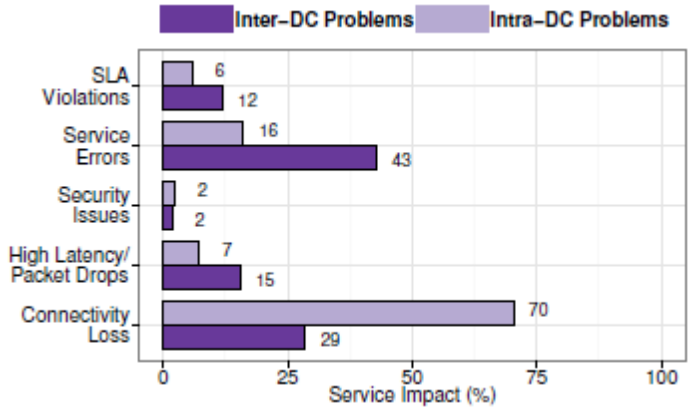

Figure 2.6 Impact of network failures on service (Potharaju and Jain (2013))

As can be observed at intra-DC level connectivity loss contributed to the highest percentage of failures at 70%; further, at inter-DC level service errors yielded the highest percentage failure at 43%. This shows that connectivity loss and service errors were the failure factors causing most outages.

In their survey on systems approaches to tackling configuration errors Xu and Zhou (2015) pointed out that configuration errors are some of the most dominant causes of outages and downtime. Further Barroso and H¨olzle report that configuration errors were the second major cause of the service-level failures at one of Google's main services (as cited in Xu & Zhou, 2015). Rabkin and Katz (2013) report that configuration errors were the dominant cause of Hadoop cluster failures, in terms of both the number of customer cases and the supporting time.

With all these outages at the different levels in the cloud how have service providers been building for availability?

2.4 Availability defined

Availability refers to the ability of a system to provide services efficiently and on time (Keus and Ullman, 1994). ("Availability", n.d.) goes further to define availability, *inter alia*, as "readily obtainable, accessible, suitable or ready for use, at hand". "Highly available characterizes a system that is designed to avoid the loss of service by reducing or managing failures as well as minimizing planned downtime for the system. We expect a service to be highly available when life, health, and well-being, including the economic well-being of a company, depend on it" (Weygant, 2001).

Availability in cloud computing is generally measured using the 9's measurement with the count of 9s being a percentage, for example three 9's implies (99.9%) uptime while four 9's implies (99.99%) uptime and so on. This is normally specified in the service provider's SLA (Service Level Agreement) with the ideal uptime environment being the five 9's (99.999%) uptime. The uptime percentages are based on annual availability minus any time it takes for maintenance and scheduled outages. The amount of downtime is calculated by multiplying the percentage downtime by the total hours in a calendar year (Endo et al., 2016):

Let percentage downtime = D;

$Downtime\ per\ annum\ (A) = D\ \times 24\ \times 365$;

As an indicator three 9's means 99.9% uptime which implies nine hours of downtime per year (outside the scheduled outages and maintenance outages)

$A = 0.1\% \times 24 \times 365 = 8.76$ hours

While 4 9's means 53 minutes of downtime per year:

$A = 0.01\% \times 24 \times 365 = 0.876 \text{ hours} = 52.56$ minutes

The foregoing interpretations of the term availability imply that an available system (or network for that matter) is one that is accessible, ready to use at any given time and information or resources on it can be accessed and used in the correct format. Suffice to say this is what cloud service providers promise to users of their services and in return users expect no less. Cloud service providers have used different techniques to ensure availability to their users and it is worthwhile noting that availability must be addressed at both datacenter and at infrastructure level.

Rohani and Roosta (2014) define three different types of availability, namely inherent, achieved and operational.

Jose (2013) proposed a measure of availability called service availability, which is defined as:

Service Availability (η) = R_A / R_R

Where R_A is the resources allocated

and R_R is the resources requested

2.5 Current Availability Mechanisms (AMs)

Fault tolerance is generally defined as the ability of a system to remain in operation even if some of the components used to build the system fail (Barr & Narin, 2010). However, there are other AMs that service providers have designed to increase availability. These techniques include replication, use of proxy network, checkpointing, redundancy, passive-active mechanisms, virtualization, multi-master architectures, collaborative fault tolerance and configuration free systems. Other techniques in use but which do not form part of this study include using

specialized middleware (Kanso & Lemiuex, 2013) and software defined availability (Teich, 2014).

2.5.1 Use of replication

This AM involves replicating servers and storage across the network (Hauck, Huber, & Klems, 2010). However, replicating data across networked servers provokes a trade-off challenge known as the strong CAP principle (Fox & Brewer, 1999; Gilbert & Lynch, 2002). CAP states that only two of three properties, transactional consistency (C), high availability (A), and resiliency to network partitions (P) can be achieved at the same time. In widely distributed systems – for the most part cloud environments – partitions are considered inevitable, leaving the trade-off between consistency and availability, for example, high availability can be achieved by optimistic replication at the cost of consistency (Hauck et al., 2010). Summarily, when one server goes down another takes over; the same applies to storage. The disadvantage is the tradeoff brought about by the CAP principle, meaning that availability can only be achieved at the expense of either consistency or partition tolerance.

2.5.2 Use of proxy

Another way of increasing availability and pursuing the fault avoidance theorem include trying to eliminate bottlenecks in the network. Bearing in mind that data communication in most clouds uses the client-server paradigm a cloud proxy network can be deployed that allows optimized data-centric operations to be performed at strategic network locations (Weissman & Ramakrishan, 2009). This is demonstrated in figure 2.7

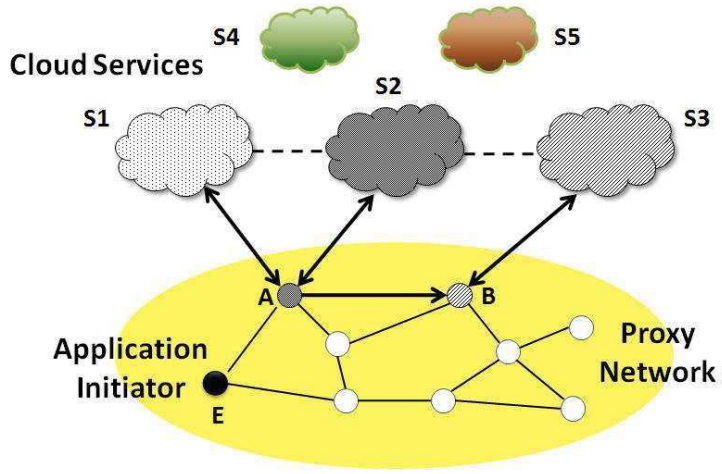

Figure 2.7 System Model (Weissman & Ramakrishan (2009))

Si are the cloud services, nodes A and B are proxies, and E is an initiator for an application that uses clouds S1, S2 and S3. Solid arcs represent actual proxy-to-proxy and proxy-cloud interactions, and dotted lines represent logical cloud-to-cloud interactions**.** The power behind this proxy model lies in the combination of the following set of roles which boost performance and reliability of distributed data-intensive applications:

- Cloud service interaction – a proxy may act as a client to a cloud service; thus a proxy with better network connectivity can access one or more cloud services.

- Computing – a proxy may carry out computations on data via a set of data operators. This role allows a proxy to filter, compress, merge, mine, and transform data.

- Caching – a proxy may efficiently store and serve data to other nearby proxies that may consume the data later on. Proxies can also cache intermediate results from a cloud interaction that may be reused again.

- Routing – a proxy may route data to another proxy as part of an application workflow. This role is particularly important if the application is interacting with multiple clouds which are all widely distributed, and there may be no single proxy that can efficiently orchestrate all of these interactions.

Lastly the application initiator is a node (E in figure 2.7) in the proxy network which acts on behalf of an application, for example, an end-user machine. The initiator acts as an application control point and the place where resource allocation decisions are ultimately made.

The advantage of using the proxy network is that there is more efficient handling of tasks in the proxy nodes as they are logically connected. However, if a cloud becomes unavailable then only the data that had been sent to the proxy can be used; further, if a proxy also becomes unavailable there is no mechanism to bring it up and any data that it had not already shared with the other proxies becomes unavailable as well. This implies the need to have a mechanism in place that can detect the failing proxy and migrate information on it to another proxy when it fails.

2.5.3 Use of checkpointing

Availability can further be enhanced by examining the checkpoints in the cloud infrastructure. The cloud can be viewed as a 3-tier architecture with a presentation manager at the top of the hierarchy that receives all user requests. This is also the central cloud and users cannot access nodes beneath it. The request manager allocates these requests as jobs/threads to sub-clouds (service manager) which in turn allocate them to service nodes which process the individual threads. Singh, Singh, and Chhabra (2012) proposed that availability can be increased by improving checkpoint efficiency and preventing check pointing from being the bottleneck of cloud data centers. A checkpoint is a local state of a job saved on stable storage. Checkpoints work like restore points for an operating system such that the status of a process can be saved at consistent intervals; if there is failure computation can be resumed from the earlier checkpoints, thereby avoiding restarting execution of the job from the beginning again. When a node fails at the service manager or service node level, the threads can be re-allocated to other nodes which will take up the execution since in cloud computing environments nodes in the data centre do not share memory. They went on further to examine the check pointing scheme using two main metrics: checkpoint overhead (increase in the execution time of the job because of a checkpoint implementation) and checkpoint latency (duration of time required to save the checkpoint). By performing a multilevel checkpoint analysis in a simulated environment they observed the shortcomings of Young's model and Daly's extension of Young's model (cited in Sing et al.,

2012). Essentially by varying the checkpoint rerun time Singh et al (2012) proposed two load balancing algorithms to cater for the multilevel proposition so that execution time for a job could be minimized. In conclusion they stated that the check pointing interval plays a critical role in determining availability of the cloud.

The advantage of this AM is that as checkpoints act like restore points of an OS, they reduce job execution total time in event of node failure at manager or node level by re-allocating threads in the shared-nothing environment (uses checkpoint overhead and checkpoint latency as metrics). However, it is based on rollback recovery which is reactive not proactive; this does not address a particular outage cause since some outages affect the nodes themselves making this mechanism ineffective in such instances.

2.5.4 Use of redundancy

Thanakornworakij et al. (2012) also proposed a high availability solution in HA-OSCAR (High Availability Open Source Cluster Application Resource). OSCAR is a cluster software stack that provides a high performance computing runtime stack and tools for cluster computing (Brim, Mattson & Scott, 2001). Cluster computing is whereby more than one computer is connected together to act and appear as one computer. The main goal of the HA-OSCAR project was to leverage existing OSCAR technology, so the HA-OSCAR project was formed to provide high-availability capabilities in OSCAR clusters. HA-OSCAR then introduces several enhancements and new features to OSCAR mainly in areas of availability, scalability and security. The proposed system (HA-OSCAR 2.0) would use the concept of component redundancy to eliminate single-point-of-failures. It would utilize HATCI (High Availability Tools Configuration and Installation). HATCI is composed of three components: Node Redundancy, Service Redundancy and Data Replication Services. As seen in figure 2.8 there is redundancy from the head node down to the switches and to the client nodes thus ensuring if any primary device fails then a secondary device can take over its place. The head node provides service requests from users and routes appropriate tasks to the compute nodes (essentially similar to the request manager described by Singh et al. (2012) in section 2.6.3). An evaluation was then

performed to demonstrate improved availability in an OSCAR-V and HA-OSCAR integrated environment.

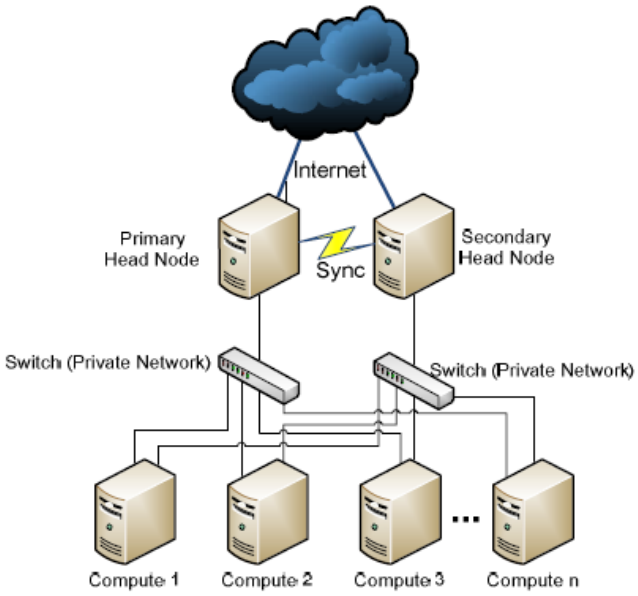

Figure 2.8 Typical HA-OSCAR V Cluster System (Thanakornworakij *et al.*(2012))

Availability can be shown by means of the Continuous Time Markov Chain (CTMC) model as shown in figure 2.9.

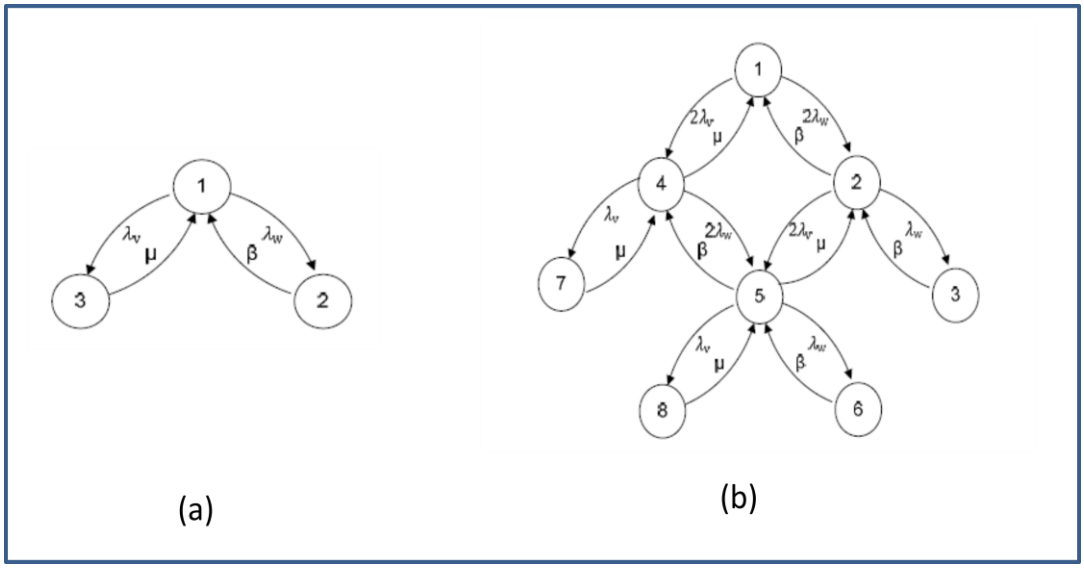

Figure 2.9 CTMC diagram for OSCAR-V Cluster system in (a) single state and (b) multiple states (Thanakornworakij et al.(2012))

In figure 2.9 (a), in state 1 both server nodes and switches are functioning well. In state 2 a switch node has had a failure while in state 3 a server has had a failure. The system will only be available when all components are working, which is state 1. The system goes from state 1 to state 2 when switch failure occurs at rate λ_w and from state 1 to state 3 when server failure occurs at rate λ_v. Switch recovery occurs at rate β while server recovery occurs at rate μ, returning the system to state 1 from state 2 and state 3 respectively. For maximum availability the system must stay in state 1 as long as possible. Figure 2.9 (b) shows the system being available in multiple states namely 1, 2, 4 and 5. By performing an availability and cluster system analysis Thanakornworakij et al. (2011) reached the conclusion that availability for OSCAR-V cluster system was 0.996 while it was 0.99999 for the HA-OSCAR V cluster system. This translates to a downtime of 39.2 hours and 4.25 minutes annually respectively.

The advantage of this AM is efficient failover mechanisms at all levels and it works well in cluster environments. However, it is designed for cluster environments and does not address a particular outage cause, for example, could failure of primary device cause failure of failover device?

2.5.5 Use of passive-active mechanisms

The Linux-HA (High-Availability Linux) project (linux-ha.org, accessed 12/08/17) provides a high-availability solution for Linux, FreeBSD, OpenBSD, Solaris and Mac OS X which promotes reliability, availability, and serviceability. This solution has been used in a number of experiments such as Solissa and Abdurohman (2018), Leppinen et al. (2019), and Dang et al. (2019). The project's main software product is Heartbeat, a GPL (General Public License)-licensed portable cluster management program for high availability. Heartbeat can detect node failures reliably in less than half a second. With a low latency communication infrastructure, such as Infiniband or Myrinet, this time could be lowered significantly. The architecture is based on an active-passive high availability solution (where one server is active while the other one remains passive until the active one fails, in which case it takes over). Each service under high availability needs at least two identical servers: a primary host, in which the service run, one or more secondary hosts, able to recover the application in less than one second. As a result of

failure detection, the active-passive roles are switched. The same procedure can be done manually, for planned or unplanned down time, i.e. in case of maintenance needs. A heartbeat keep-alive system is used to monitor the health of the nodes in the cluster. Heartbeat monitors node health through communication media, usually serial and Ethernet links. It is a good solution to have multiple redundant connection links. Each node runs a heartbeat daemon process. When a node death is detected, Heartbeat runs a script to start or stop services on the secondary node. A local disaster recovery solution is typically composed of two homogeneous nodes, one active and one passive. The active node is usually called master or production node, and the passive node is called secondary or standby node. During normal operation, the only working node is the master node; in the event of a node failover or switchover, the standby node takes over the production role, by taking its IP number, and completely replacing the master one. To maintain the standby node for failover, the standby node contains homogeneous installations and applications: data and configurations must also be constantly synchronized with the master node. Figure 2.10 shows a typical dual server heartbeat system.

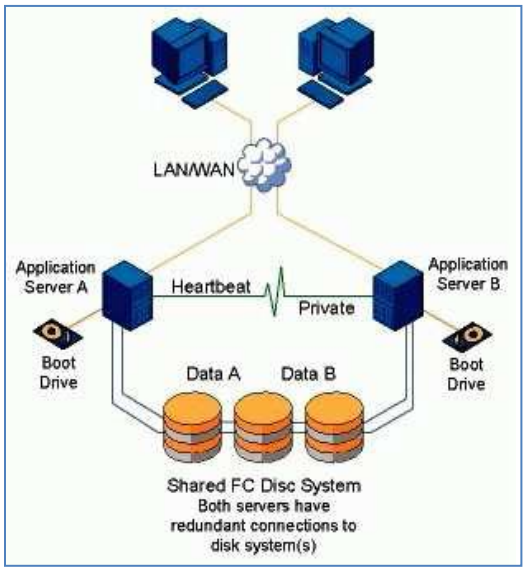

Figure 2.10 Typical dual server heartbeat system (linux-ha.org)

The advantage of this AM is that it's very versatile and relies on Heartbeat for detection. However, it is ideally designed for specific environments, namely Linux, FreeBSD, OpenBSD,

Solaris and Mac OS X. If the mechanism can be used or customized outside of these specific environments then it would be optimal.

2.5.6 Use of virtualization

Vmware Inc is a leading IT solutions provider. The organization has sold its solutions using the concept of virtualization; in this instance the use of virtual machines. The virtual machines give the illusion of a physical complete machine to the user, but it is only a software generated device that offers the full functionality of a working PC. One of their products is the Vmware Infrastructure 3 virtualization suite that seeks to offer HA to clients. It is based on the concept of clusters and resource pools. The latter simplify control over the resources of clusters or hosts (Vmware, 2007). Figure 2.11 shows how the solution is configured.

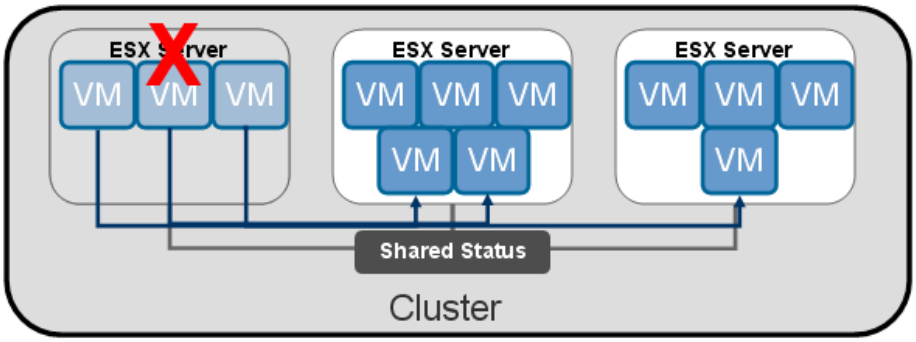

Figure 2.11 The Vmware HA solution (Vmware. (2007))

Figure 2.11 shows the solution lies in combining the server hosts into one cluster so that resources can be shared. The cluster is seen as one operational unit even though there are several physical hosts in it. When one server fails (marked X in figure 2.11) the other servers immediately have the virtual machines from the failed server transferred to them by Vmware HA. This setup can be aptly summed up with the famous 'one for all and all for one' adage. There are many advantages to such a setup particularly in terms of saving hardware costs (no

need to purchase physical redundant servers) and the fact that all applications can access the cluster regardless of the platforms they utilize.

The virtual machine approach has also been explored by Nagpal, Shivar and Kumar (2013) in real time cloud computing environment. Their approach in building for fault tolerance is an adaptive one that relies on measuring the reliability of the nodes during task execution.

This solution is particularly apt for the cloud computing environment. This is so since cloud computing technology is based on the concept of virtualization. If the solution could be customized to address particular outage causes then it would be an ideal AM.

2.5.7 Use of multi-master architectures

Availability can also be examined by breaking down the cloud architecture and offering different solutions for availability at the different levels of the cloud. Stanik, Hoger and Kao (2013) did exactly this with their CloudDisco solution. This solution offers availability at the cloud middleware level using a self-healing mechanism in case of failure at any of the three points depicted in figure 2.12

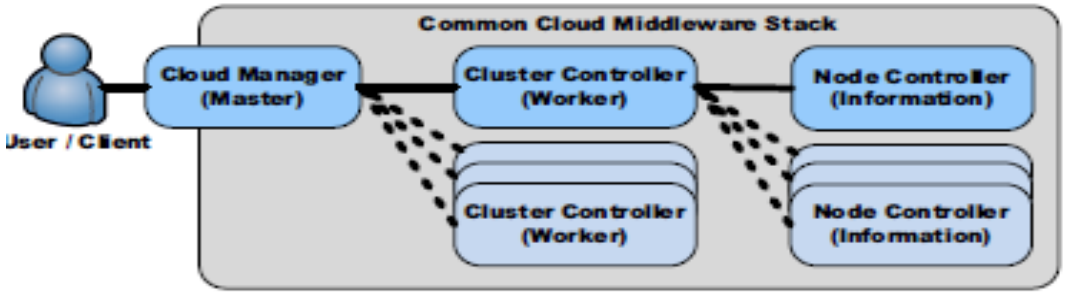

Figure 2.12 Middleware components in 3-tiered master-worker architecture (Stanik, Hoger and Kao, 2013)

In the architecture above the cloud manager acts as a master for the cloud environment receiving requests directly from the user for infrastructure (hardware). The cloud manager in turn passes

this request to the cluster controllers (workers). The controller workers process the request and pass it back to the cloud manager who in turn also notifies the user. It can be observed in this architecture if the cloud manager fails then the entire cloud fails even if both the cluster controllers and node controllers are available. CloudDisco offers a multi-master architecture which the authors claim not only prevents failure but also offers a self-healing mechanism in the event of failure in any of the levels of the architecture.

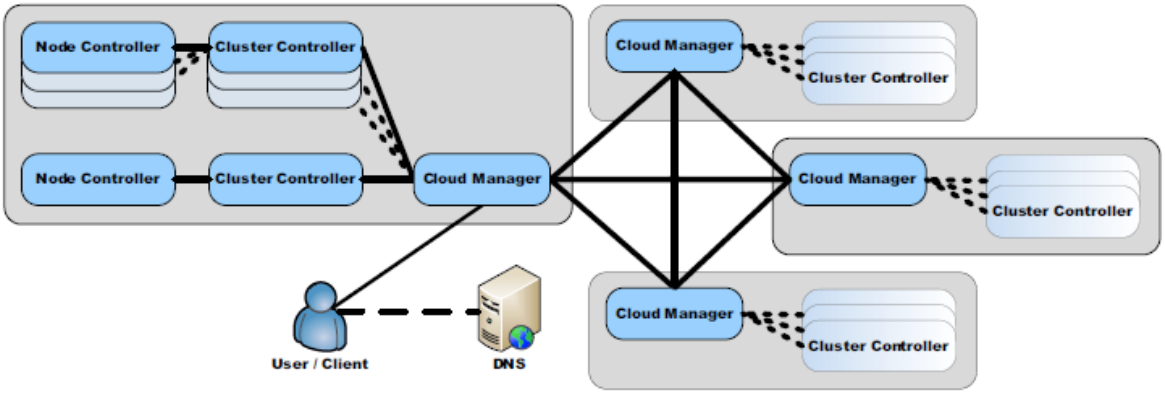

Figure 2.13 CloudDisco multi-master architecture layout (Stanik, Hoger and Kao (2013))

In the multi-master architecture shown in figure 2.13 all cloud managers are peers and act as a unit; thus they are not replicas but active peers which collectively make up the cloud, i.e. each cloud manager owns a fraction of the cloud. All cluster controllers are connected to exactly one cloud manager at any given time and since all cloud managers are peers, users and cluster controllers alike, can connect to any cloud manager. The cluster controllers do not have any knowledge of each other (operating like a shared-nothing environment) but are connected to at least one node controller (resource provider). In the above architecture each cloud manager must be connected to at least one other cloud manager and the collective collection (of cloud managers) results in a mesh topology between cloud managers and a tree topology with each master architecture. In the event of failure of a cloud manager the cluster controller within it can connect to another cloud manager by means of a mechanism thus ensuring no total cloud blackout and a better failover mechanism. This is illustrated in figure 2.14

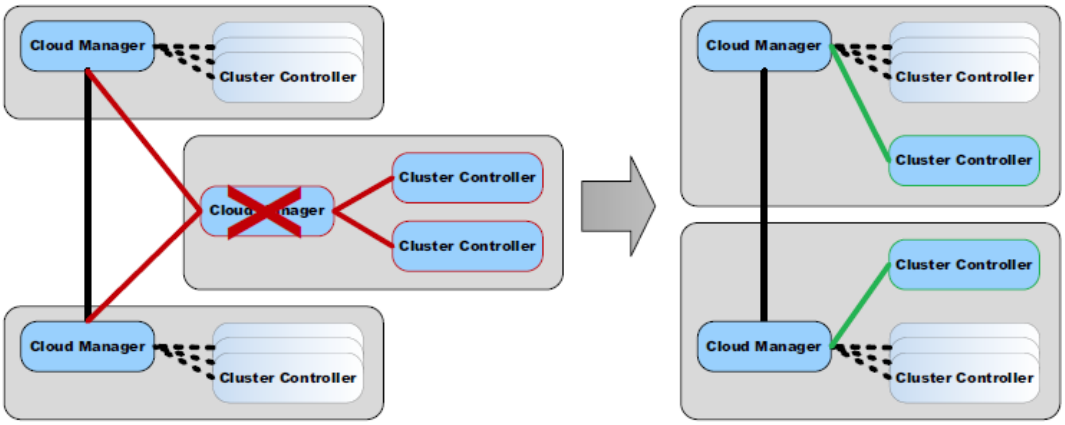

Figure 2.14 Self-healing mechanism in failover pattern. (CloudDisco) (Stanik, Hoger and Kao, 2013)

In figure 2.14, the cloud manager in the middle master architecture fails and the two individual cluster controllers then move and connect to the two nearest cloud managers thus reducing the hop distance. It is also worth mentioning that the authors bore scalability in mind when developing this architecture, for the most part through the cloud managers. There are many master nodes available in this setup and these ensure scalability since user requests can be distributed across all available master nodes (cloud managers).

The self healing mechanism presents a versatile AM in this case. However, due to the multiple cloud managers a multi-attack can result in bottlenecks in the whole system and thus reduce availability of the mechanism as a whole. Further this solution does not target any particular outage cause and only focuses on node failure and not what caused the failure in the first place.

2.5.8 Use of fault tolerance

Tchana, Broto and Hagimont (2012) opined that most fault tolerance strategies face challenges due to the disjoint between the two main users of the cloud, i.e., the service provider and the end user. This is due to the fact that each one has control only on a certain part of the cloud but no ubiquitous control by either party. Customers are limited to only detecting faults of virtual

machines and their applications, while the provider can only manage real resources (physical machines) and virtual machines faults. Therefore, possible Fault Tolerance (FT) solutions vary according to the involved participants and according to the implementation level. The two parties (service providers and end users) access to the architecture is illustrated in figure 2.15

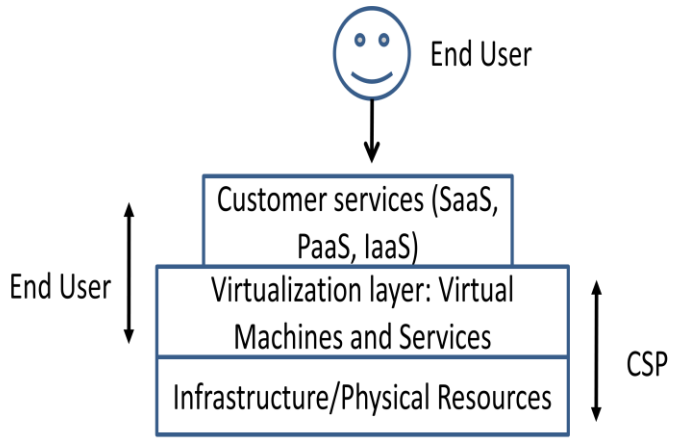

Figure 2.15 Cloud computing architecture (Tchana, Broto and Hagimont, 2012)

As an example of collaborative fault tolerance at the virtual machine (VM) level of the architecture the authors state: "If the VM fault detection is done by the customer, we will have the same problem as described in the previous section (where at the application level the customer cannot tell where the fault is). At the customer level, a VM failure detected by a sensor can be due to a hardware failure (the machine, which hosts the VM), which is out of scope for the customer. VM fault detection at the cloud level allows getting a more accurate decision. If the VM fault detection is implemented at the cloud level, collaboration with the customer level can provide a better solution than the checkpoint-based one... Concretely, once the fault is detected by the cloud, it starts a new VM with the same features (networking, memory, CPU, image) as the failed VM and then calls the customer to redeploy, restart and synchronize the new VM. This solution would probably perform better than the checkpointing one regarding the cost of save/restore operations". The authors went further to perform an analysis of exclusive fault tolerance (FT) versus collaborative FT using a simulated environment consisting of a prototype called CloudEngine. CloudEngine is based on an adaptable autonomic management system

called TUNeEngine which has management capabilities at both cloud and customer level. Collaborative FT performed using CloudEngine and the administrative capabilities of TUNEeEngine (for the customer level) show a relatively longer repair time (5 minutes and 30 seconds) but with no overhead on execution (as is the case with other exclusive FT efforts like checkpointing). The service is kept available during repair using a mirror server.

Fault tolerance as an AM has been used in many implementations, for example Das and Khilar (2013) who proposed a technique called Load Balancing for Virtualization and Fault Tolerance in Cloud Computing (LBVFT). This technique attempts to combine virtualization and fault tolerance on one platform to increase availability. The attempts to use different variations of fault tolerance all require testing the technique against different types of attacks, since each attack is aimed at crippling a certain part of the infrastructure; thus, fault tolerance techniques would provide better results when targeted at particular types of attacks.

2.5.9 Building configuration free systems

In their survey on systems approaches to tackling configuration errors Xu and Zhou (2015) opined that *inter alia*, hardening systems against configuration errors and building configuration free systems would help ameliorate configuration errors. This approach is one that is targeted specifically at configuration errors, unlike most of the described AMs in this section. The challenge would be the technique to use to implement this; for example, configuration occurs at different levels in the cloud, from the infrastructure level down to the SaaS layer. However, as the focus of this study was at the infrastructure level then building for configuration free systems at this level would involve either pre-configured scripts or some form of restriction in input. This is discussed further in chapter four of the thesis.

2.6 Critique of the identified works

The literature has examined the different outage causes that have been identified by various authors. Further, mechanisms and techniques that have been designed to increase availability in the cloud have been described in section 2.5. An examination of these AMs indicates a

similarity in terms of not addressing particular outage causes. The purpose of this literature review was to show that the shortcomings in the AMs is that, whereas they give general solutions to increasing availability, they do not address specific outage causes. It was posited in chapter one of this thesis that when AMs can be designed to address particular outage causes then will availability be increased in the cloud. It was thus important that the outage causes be identified first in the literature, then each AM examined to determine its shortcomings, and further which particular outage cause it addresses.

2.6.1 Identification of outage causes: Specific Objective One

The first specific objective of this study was to identify the real causes of outages so that they form the basis for relating them to the AMs. Different authors have devised different taxonomies on outage causes, such as Nabi, Toeroe, and Khendek (2016), Pham, Phuong Cao, Kalbarczyk, & Iyer (2012) and Kalyan & Kumar (2015). These authors described different outages as related to their respective studies.

The outage causes used in this study can be identified from the literature review done describing the works of the CSA, Bigelow (2011), Myerson (2013), Li et al. (2013), Potharaju and Jain (2013), and Xu and Zhou (2015).

As at the time of this writing the Cloud Security Alliance had identified twelve major threats to cloud computing, most of which pose the potential to bring the entire cloud down (table 2.1 and table 2.2).

Li et al. (2013) survey on cloud outages and resultant classification of outages was described. The authors' result of the survey was that (1) none of the Cloud vendors can avoid suffering from service outages; (2) Cloud service outages could happen at any time, and each Cloud vendor has experienced violation of its Service Level Agreements during the past years; (3) Climate and Age are two influential factors related to the outage locations; and (4) Power Outage and Routing/Network Issue are two common classes of Cloud service outage causes. However, this study only sampled the top five vendors in North America and performed an analysis based

on accepted research methods. The study also opined that the framework revealed could be helpful to other cloud vendors. The main aim of their research was to show cloud providers what type of outages they are likely to experience in their environment and thus learn lessons therefrom. However, the sampling of only the top five vendors in North America does raise the question regarding the choice of sample as well as the fact that it is debatable that that the network size in itself does play a role in determining the kind of outages to expect (this can be deduced from the work of Potharaju and Jain (2013)). Further the study did not go a step further to test these parameters in a simulated environment and observe the results, thus, while the research methodology is acceptable no actual testing was done.

Bigelow (2011) interviews with the industry experts revealed five major causes of downtime, namely human errors, network protocols, pilot errors and network servers. These causes are based on the experiences of these experts, and perhaps due to information sharing within the industry. Thus it is not conclusive from a scientific viewpoint as the causes are broadly general and not specific to any particular documented environment.

Potharaju and Jain's (2013) analysis was a first of its kind as it covered outages both within and between data centers. The focus in the investigation, however, was based on the networking aspect of the data centers. The study did not examine other aspects of the cloud such as the infrastructure services, or even the virtual machines which may play a role in bringing the network and by extension the cloud, down.

Each of the above works focused on a different aspect of the cloud and the experiences of different users. The outage causes identified above can be categorized and grouped for the purposes of this study as:

1. Hardware related issues
2. Configuration issues
3. Security issues
4. Resource exhaustion
5. Node failures
6. Network related issues

7. Natural disasters.

2.6.2 Availability by use of failover policies

The research by Myerson (2013) on causes of cloud outages is based on outages due to resource exhaustion in the cloud. The study identified seven factors namely leap year failure, numerically unstable algorithms, resource optimization failure, threshold policy implementation failure, hypervisor failure and virtual desktop failure. It went to examine mitigation of risk from the user, developer and technical container perspectives. In all three areas it suggests failover mechanisms to manage the different user level expectations. Myerson's research then points out that users should look to the SLA to determine what the policy on cloud outages is.

However, the research was focused on dealing with cloud outages or at least mitigating risk based on policies that specify failover mechanisms to deal with the user expectations. The use of failover mechanisms would be desirable if the mechanisms would be proactive rather than reactive; that is, policies designed to prevent the outage from occurring in the first place. This represents a shortcoming as far as the objectives of this study are concerned, as the policies suggested by Myerson come into effect after the outage occurs, rather than preventing the outage from occurring in the first place. There is therefore, a need for limit detection policy that can achieve this. This AM, therefore, is designed to address resource exhaustion as an outage cause.

2.6.3 Availability by replication

When designing for high availability it is worth remembering the strong CAP principle which affects widely distributed environments (the cloud is a widely distributed environment). Hauck et al. (2010) concluded that high availability can be achieved by optimistic replication at the cost of consistency because of the CAP principle. However, most availability solutions are based on some form of replication mechanism at hardware or software level. Availability in this instance could be increased by pairing replication with another mechanism, say active redundancy perhaps at hardware level. This, however, could be research that can be done as it is not part of

this study. This AM does not address any specific outage, rather it anticipates inconsistency of data, should the other two CAP parameters be achieved.

2.6.4 Availability using proxies

The proxy network solution proposed by Weissman and Ramakrishan (2009) while theoretically viable would most likely pose resource challenges to service providers. This is since all nodes in the proxy network act as peers, and therefore as this is a distributed environment the question arises as to how much resource intensive it will be. Further even though the proxy network may increase overall availability of the clouds by its very design it is actually intended to accelerate execution of cloud applications across a distributed environment. The research did not examine or build for fault tolerance, for example, what happens if a node holding data for execution from a user goes down? What mechanism can recover this data and pass it on to other nodes or to the desired cloud for execution? A possible solution to this would be to introduce node management as an AM to manage the nodes themselves using scripts or policies at the server level (the level which controls the initiator(s)) that could proactively anticipate challenges such as node failures caused by resource exhaustion or other causes. This AM does not address any specific outage cause, rather, offers a solution to increasing availability in general terms.

2.6.5 Availability using checkpointing

The checkpointing algorithms proposed by Singh et al. (2012) in increasing availability were also examined. The study's algorithms propose to refine the checkpointing process by fine tuning the checkpoint overhead and checkpoint latency and they concluded that a checkpoint interval of 5 seconds is optimal. As earlier described checkpoints work like restore points of an operating system. This approach works well in an environment of shared-nothing computers and is the assumption that the datacenter will consist only of shared-nothing computing environment. Further it is based on the concept of rollback recovery which happens after failure has occurred, a reactive rather than proactive approach. Lastly the study is not specific as to what type of failure this strategy is meant to address as some of the outage causes described in section 2.6.1 will affect the nodes themselves thereby effectively neutralizing checkpointing efforts.

Checkpointing as an AM would thus be more effective if it were used to address specific outage causes.

2.6.6 Availability using active-passive approach

The Linux-HA project unfortunately provides HA solution only for particular platforms. However, the mechanism in itself is a very potent solution as it uses the concept of an active-passive approach, where if one part fails (active) another is there to immediately take over (passive). The Linux-HA project doesn't target any particular outage cause either, however, using the principle of active-passive it can be used to deal with particular outage causes.

2.6.7 Availability using cluster management

The CloudDisco multi-master architecture by Stanik, Hoger and Kao (2013) is also a very versatile solution in terms of design. It is also the first architecture observed where a self-healing mechanism is deployed. The study suggested an almost perfect way of ensuring availability for the end user. The only piece that may need addressing in this case is the scenario where a multi-attack occurs and there is need to make several failovers to other cloud managers. In the event that all failovers are successful, would it affect reliability as a bottleneck, increasing latency and reducing efficiency or even causing downtime due to resource exhaustion?

Similarly, the HA-OSCAR project by Thanakornworakij *et al.* (2011) was also described and it is a versatile solution for cluster environments. The use of secondary devices as failover mechanisms is a worthy approach for most cloud environments. However, failover mechanisms work well for purposes of fault tolerance without addressing what caused the failure in the first place. In the case of hardware failure can the same cause also make the failover device to fail as well?

CloudDisco, HA-OSCAR and the Vmware solution utilize the concept of cluster management in a virtual environment. The Vmware solution was identified as a suitable AM to use in the cloud environment as the cloud is for the most part operationally virtual. The design of CloudDisco, HA-OSCAR and Vmware are all designed to virtually manage clusters from the cluster

controllers downwards in the infrastructure. Nonetheless, none of these three were designed to deal with specific outage causes. Cluster management would therefore be a suitable AM to use in the cloud environment to deal with specific outage causes.

2.6.8 Availability using fault tolerance

The fault tolerance approaches offered by Tchana, Broto and Hagimont (2012) as well as Das and Khilar (2013) are based on virtualization. The former used collaborative FT approach with the downside that there was a longer repair time but no overhead on execution. The autonomic management system introduced in this system (TUNEeEngine) does provide good fault tolerance. The same can be said for Das and Khilar (2013) as their research approach involves load balancing (a traditional approach even out of virtualization circles) using virtual nodes to balance loads and also check the healthy state of the nodes. Fault tolerance as an AM is a suitable AM especially in environments where outage causes might not be of an anticipatory nature, for example natural disasters. However, the two mechanisms described in this section were also not designed to handle any specific outage cause.

2.6.9 Availability using configuration management

Xu and Zhou's (2015) solution to tackling configuration errors is a practical one in terms of building configuration free systems. The authors suggested that configuration errors can be dealt with by building configuration free systems. The paradox of using the latter approach would be that the key would be in building an underlying configuration that is totally error free in the first instance, which again lands back at the former. However, this is one AM that has been specifically suggested for tackling configuration errors as an outage cause.

2.6.10 Identification of Availability Mechanisms (AMs): Specific Objective two

The second objective of this study was to identify AMs in use in the cloud computing infrastructure. Different authors have used different taxonomies to classify availability mechanisms, such as, Bajaber, AlQulaity, and Alotaibi (2017) and Mesbahi, Rahmani, and

Hosseinzadeh (2018). For the purposes of this study, from the foregoing AMs identified from section 2.6.2 to 2.6.9 these can now be grouped into seven different categories. The identified AMs are:

1. Component redundancy
2. Cluster management
3. Checkpointing
4. Fault tolerance
5. Active-X variant
6. Node management
7. Limit detection policy.

2.7 Outstanding issues/addressing the gap

In the background to this study the thesis sought to better understand the causes of cloud outages and relate them to the availability mechanisms that have been developed by researchers and industry over the years so that these mechanisms can be improved. The current availability mechanisms that have been developed by researchers address the different ways in which to keep the cloud up and running without addressing specific outage causes. The exception to this, however, is twofold: the work of Myerson (2013) which specifically suggested failover policies for resource exhaustion, and Xu and Zhou (2015) who suggested configuration free systems as a means of eliminating configuration errors.

The reviewed literature has identified seven AMs and outage causes which are summarized in table 2.3

Table 2.3 Availability mechanisms and outage causes

Availability Mechanisms (AMs)	**Outage causes**
Component redundancy	Hardware issues
Cluster management	Configuration issues
Checkpointing	Security issues
Fault tolerance	Natural disasters
Active-X variant	Network issues
Node management	Node failures
Limit detection policy.	Resource exhaustion

From the study of the literature covering outage causes and that covering AMs it is evident that the two groups of authors when juxtaposed have been running research in parallel. Where do they meet? This is to state that there is a disconnect between causes of outages and proposed availability mechanisms; no literature is available and no research as of the time of this writing has been presented to correlate outages to the existing availability mechanisms.

As observed in the real world one cannot take broad spectrum antibiotics to treat each and every single infection of the body even though they are available. Science as a whole must take the approach to treat causes as opposed to symptoms.

The purpose of this study was to develop a model that will address the causes by correlating them to the different availability mechanisms in place. Only when this happens can more relevant availability mechanisms be designed and existing mechanisms be improved to increase uptime in the cloud. Improvement will be achieved since by examining the model designers will be able to see what they may have missed in the design of their solutions and thus improve on them or build different solutions. Cloud providers, current and in future, will also be able to

learn from the model in terms of getting to know how to build for better availability and consequentially better uptime. The model is introduced and discussed next.

2.8 Model development and design

The proposed solution was in the form of a model that could be used by stakeholders in building for optimal availability in their respective cloud computing environments called the Ferris Wheel of Availability (FWA) model. The model aimed to correlate various causes of outages to known availability mechanisms in order to proactively increase availability.

2.8.1 Relationship between outage causes and AMs

In determining the relationship between outage causes and AMS, it is argued logically that by putting an AM in place at infrastructure level proactively, i.e., before an outage occurs, then it can prevent the targeted outage cause from making the infrastructure unavailable in the first place. This is exactly what the FWA model purposes to do; to attempt to logically relate the AMs to specific outage causes so that when the AMs are in place the outage causes will not cause the infrastructure to be unavailable. As a result the infrastructure will continue being available to provide services. As of the time of this writing this is the first attempt to do so in a model.

It is expected that when CSPs are able to address specific causes of outages to known availability mechanisms then they can increase availability rather than simply adopting generic availability mechanisms without specific reference to their particular situations.

From the literature review it is clear that outages cause availability to go down to the extent that the infrastructure is not available. This is a concern to users and CSPs alike in that lack of availability results in lack of productivity since services are not available. An AM would counter an outage cause by ensuring that the infrastructure stays up even when an outage cause comes into effect. In effect this means that availability is dependent on an outage not happening. Thus an AM will ensure that despite having an occurrence of outage cause availability will remain unaffected at infrastructural level. This relationship is demonstrated in figure 2.16.

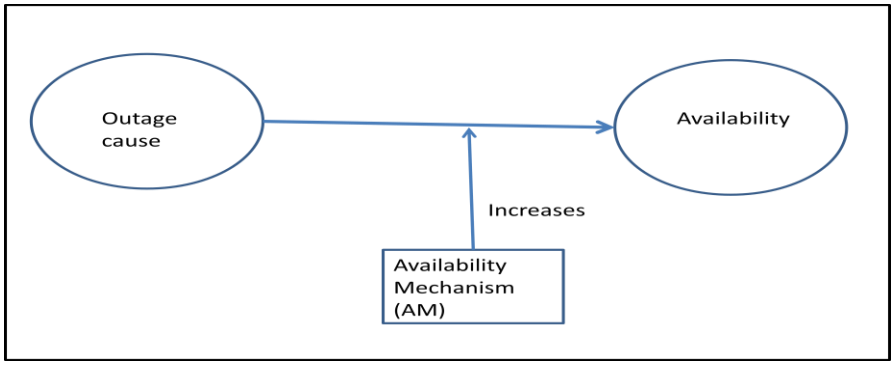

Figure 2.16 Relationship between outage cause and AM

2.8.2 Structure of the FWA model

The structure of the FWA model was based on the outage causes and AMs identified in the chapter two. As discussed in section 3.2.1 it is expected that when an AM can address a specific outage cause then can availability be increased at CSP level; this was done successfully by Xu and Zhou (2015) when they argued the case for building configuration free systems. Myerson (2013) also posited the case for failover policies to counter resource exhaustion.

The relationship between AMs, outage causes and availability was discussed in section 2.8.1. Figure 2.16 went further to show this relationship in a visual model. The relationship between outage causes, AMs and Availability can be explained as follows:

$Let\ O_n\ represent\ an\ outage\ cause\ such\ that\ O_1\ \ldots.O_n\ are\ all\ outage\ causes.$

$Let\ O'\ be\ \{O_1, O_2, O_3, \ldots O_n\}\ the\ set\ of\ all\ outage\ causes.$

$Let\ A_n\ represent\ an\ AM\ such\ that\ A_1\ \ldots.A_n\ are\ all\ AMs.$

$Let\ \alpha\ be\ \{A_1, A_2, A_3, \ldots A_n\}\ the\ set\ of\ all\ AMs.$

$From\ fig\ 2.16\ it\ was\ found\ that\ an\ outage\ is\ the\ absence\ of\ availability\ (A), thus:$

$Let\ O\ represent\ an\ outage => O = -A, thus:$

$Outage\ cause\ without\ AMs => Outage = -A;$

$and, mathematically\ expressed$:

$(Outage\ cause) - \ AM = \ -A$

$\therefore$

$$k\sum_{t=0}^{N} \text{Ơ} \ - \ p\sum_{t=0}^{N} \alpha \ \ = -A\ ,$$

$where$

$A = Availability\ of\ the\ infrastructure,$

$k = factor\ that\ relates\ outage\ causes,$

$p = factor\ that\ relates\ AMs,$

$t = time\ when\ availability\ is\ being\ measured.$

$Further,$

$$p\sum_{t=0}^{N} \alpha \ + \ (-k)\sum_{t=0}^{N} \text{Ơ} \ = A \ldots\ldots\ldots\ldots (1)$$

Equation (1) attests that for maximum availability there should be at least one AM corresponding to at least one outage cause, i.e., when k = 1 and p = 1.

The Ferris Wheel of Availability (FWA) model presents classification of availability mechanisms and outage causes in the following manner:

- Availability mechanisms are grouped into seven broad categories
- Outage causes are also grouped into seven broad categories

For each group of outage causes there is a corresponding availability mechanism (AM) in place to counter the outage cause as is attested to in equation (1). The pairing was based on a study of the specific AMs and outage causes as described in chapter 2:

- From section 2.6.2 a limit detection policy was suggested to proactively counter resource exhaustion as opposed to failover policies.
- In section 2.6.4 node management was suggested to counter node failures.
- In section 2.6.7 cluster management was suggested to effectively counter issues that may cause outages in a cluster.
- It is reasonable to expect that when a hardware component fails then the logical AM to have in place is another hardware component that can take the place of the failed one; thus hardware issues have both component redundancy and Active-X variant as possible AMs.
- Node failures may occur at either the node or at cluster level, thus node management and cluster management respectively were expected to proactively counter this outage cause.
- Similarly the same logic would apply should there be configuration issues, since configuration occurs only at node, cluster, or datacenter level.
- A network failure at infrastructure level was expected to be countered by some redundant variant that would switch service provision from one datacenter to another and hence Active-X variant and component redundancy were the redundant variants expected to counter this outage cause.
- Checkpointing was theoretically the best AM to use against security issues such that a previous state of the system could be recovered.
- In the event of natural disasters the only option that made logical sense was a fault tolerant strategy.

The FWA model attempts to go further to relate the categories of AMs to those of the outage causes as shown in figure 2.17

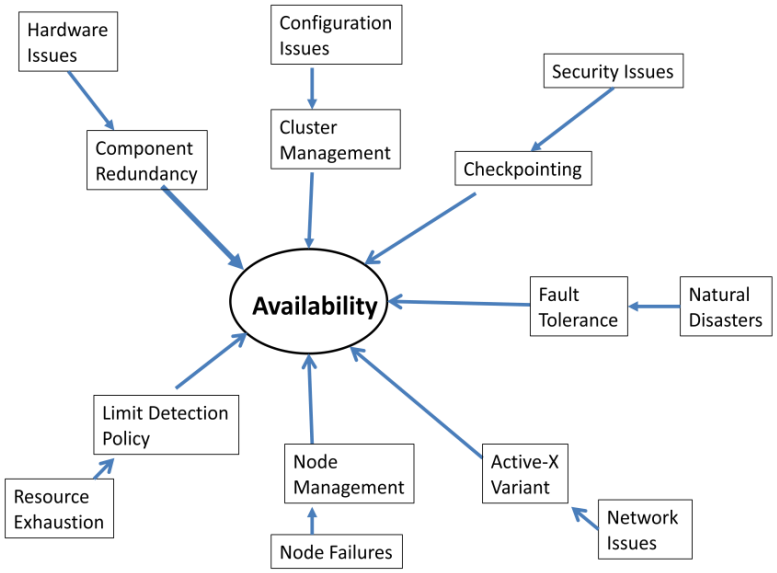

Figure 2.17 Ferris wheel of availability (FWA) model

The AMs aim to make the model proactive in that by putting the respective AM in place then this should increase availability for the service provider. The literature review only directly posited the case for failover policies (Myerson, 2013) to counter resource exhaustion, albeit reactively; and it also stated that configuration free systems can proactively prevent configuration errors (Xu & Zhou, 2015). The remaining pairings in the model add to these AMs. The model uses the imagery of a Ferris wheel since it was proposed that there would be a correlation between direct opposite outages and corresponding AMs, for example node failures and configuration issues are related, network issues and hardware issues are related and resource exhaustion and security issues are related. Natural disasters are the odd outage with no apparent correlation. Consequently some of the correlated outage groupings could use/swop corresponding AMs as opposite AMs on the wheel are conjugate in nature, for example, hardware issues can be countered by component redundancy and Active-X variant AMs. The Ferris wheel seats are designed in such a way that when the seat reaches its lowest points it maintains its perpendicular position thus making it stable for its occupants all around a 360 degree turn; thus the model aims for stability in controlling outages proactively in the cloud environment.

The FWA model uses AMs and outage causes identified in the literature review. These AMs and outage causes are found at infrastructure level at the CSP. Nabi, Toeroe, and Khendek (2016) also describe different works that have attempted to relate AMs to outage causes; however, these works are done at the middleware and application layers of the cloud.

2.8.3 Merits of FWA model

The merits of the FWA model may be described as follows:

1. The FWA model has the merit of examining a cloud environment in an encompassing manner as it views the environment from CSP level.
2. The FWA model gives a bird's eye view to CSPs and stakeholders of the cloud environment as a whole and using the model a CSP can adjust to its particular situation proactively.
3. The FWA model targets the CSPs while most the current works target the middle and upper layers of the cloud computing infrastructure, for example Prasad (2012), and Wu and Guang (2013).
4. This is a first attempt to address these variables (outage causes) together with the corresponding AMs in one generic model, which is meritorious due to the fact that all currently identified outage causes are identified and paired with AMs.

2.8.4 Challenges and Scope of FWA model

Just like with all models FWA does come with a few caveats:

1. The FWA model's scope is heterogeneous environments as opposed to homogenous environments
2. The second challenge the model may face is the cloud itself due to its dynamic nature; the cloud computing environment is changing rapidly due to advances in technology.

2.9 Summary

This chapter has described the different cloud computing architectures, benefits and deployment models. It has also examined the current literature on causes of outages in the clouds and AMs as described by different authors. Availability as defined by different authors has also been discussed. A critique of the works of these authors has also been offered, describing the merits and possible demerits of the different works. The literature has identified outage causes and grouped them. AMs have also been categorized and grouped. The AMs described in this chapter do not address specific outage causes identified by the different authors in this chapter.

This is where the study identified the gap and posited that when researchers can build AMs that address specific outage causes then will availability in the infrastructure be improved as a whole. The study has grouped the AMs and outage causes into seven respective groups, and further established that there is no model so far that relates the outage causes to AMs at infrastructure level. The relationship between AMs, outage causes and availability has also been described, leading to the formulation of the FWA model.

This chapter purposed to address the first three specific objectives of this thesis namely:

1. Identify the causes of outages in cloud computing infrastructures.

   These were identified as hardware issues, configuration issues, security issues, natural disasters, network issues, node failures, and resource exhaustion

2. Identify availability mechanisms in use in cloud computing infrastructures.

   These were identified as component redundancy, cluster management, checkpointing, fault tolerance, Active-X variant, node management, and limit detection policy

3. Formulate a model that establishes correspondences between AMs and outage causes.

   The relationship between AMs, outage causes and availability has been explained leading to the formation of the model.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 Introduction

The previous chapter discussed the current literature on outage causes and AMs. It then went on categorize the AMs and outage causes into seven groups each. The FWA model was also formulated in a step by step manner. This fulfilled the first three objectives of the study as described in the summary of that chapter.

This chapter describes the research design used to test the model, the simulator used for simulation purposes, the algorithms used to configure each simulation, and the parametric configurations for each simulation.

3.2 Methodology

This section describes the methodology used to test the FWA model. The research design was experimental. The methodology used was based on a simulation approach. The simulation approach was used in this case as evaluation using a real cloud or cloud environment would be practically impossible and not cost effective. There are different types of simulations which can be classified as being static or dynamic, discrete or continuous, stochastic or deterministic.

Simulators provide the basic behavior of the system under investigation. Dobre, Pop and Cristea observed that cloud simulation top benefits include flexibility, easy customization and lower costs (as cited by Nita, Pop, Mocanu & Cristea, 2014). To test the FWA model the CloudSim toolkit (version 3.0). CloudSim measures events discretely over time and is a deterministic simulator, i.e. it is a useful approximation of reality and a fixed set of inputs will produce a fixed set of outputs. CloudSim was developed in the CLOUDS laboratory in Australia by a team led by Buyya, Ranjan, and Calheiros (2009). CloudSim has been used in a variety of experiments that aim at investigating various aspects of the cloud, such as VM management (Monil & Rahman, 2016), modeling and evaluating multi-resource dependencies (Taddei, 2015) and performance analysis of heterogenous data centers (Bai, Xi, Zhu & Huang, 2015). Further different researchers have developed various extensions of CloudSim such as DynamicCloudSim (Bux & Lesser, 2015) and DesktopCloudSim (Alwabel, Walters & Willis, 2015); all these

extensions investigate different aspects of the cloud. CloudSim was chosen due to the following features:

- Support for modeling and simulation of large scale Cloud computing environments, including datacenters, on a single physical computing node;
- A self-contained platform for modeling Clouds, service brokers, provisioning, and allocation policies;
- Availability of a virtualization engine that aids in the creation and management of multiple, independent, and co-hosted virtualized services on a datacenter node;
- Flexibility to switch between space-shared and time-shared allocation of processing cores to virtualized services.
  (Buyya et al., 2009);

CloudSim also produces results in the form of a log, which can be configured to capture the parameters that are of interest to the user. An example output log is shown in figure 3.2

```
========== OUTPUT of Customer1 ==========
Cloudlet ID STATUS Resource ID VM ID  Time  Start Time Finish Time
    0       Success      2        0    40.4     5.1        45.5
```

Figure 3.1 Sample output CloudSim

The above output shows a successfully executed cloudlet with ID 0 which was executed in VM with the ID 0; the cloudlet took 40.4 seconds to execute and execution started at the 5.1 second mark and ended at the 45.5 second mark of the simulation.
Further the simulator outputs its results in real time and shows output of events as they take place starting from the time the simulation was started. An example output of such a simulation is shown below:

0.1: Broker_0: VM #4 has been created in Datacenter #2, Host #8
0.1: Broker_0: VM #5 has been created in Datacenter #2, Host #0

This shows that 0.1 seconds after the simulation started VM (Virtual Machine) number 4 was created in datacenter number 2 in host number 8 while VM number 5 was created in datacenter number 2 in host number 0. The broker that made the request for the creation of both the VMs was Broker_0.

The configuration of the machine that was used to perform the simulations in this chapter is shown in table 3.1

Table 3.1 Host machine system specification

Parameter	**Detail**
Manufacturer	Hewlett-Packard
Model	HP ProBook 4340s
Processor	Intel ® Core ™ i5 – 2450M CPU @ 2.50 GHz
Installed Memory (RAM)	8.00 GB
Operating System	Windows 7 Professional
System Type	64-bit Operating System

Each simulation was done once in line with Ritter, Schoelles, Quigley, and Klein (2011) who posited that in a deterministic simulation only one simulation is enough to generate valid predictions.

Each simulation in the different scenarios had different configurations (parameter settings) for datacenter, host, Processing Elements (PEs), RAM, VMs, users and cloudlets as specified in the specific simulations. The parameter settings that were used to test the FWA model are described in section 3.3.

3.3 Simulations and CloudSim parameter settings

The FWA model had proposed that for each group of outage causes there is a corresponding availability mechanism (AM) in place to counter the outage cause. The AMs aim to make the

model proactive in that by putting the respective AM in place then this should logically increase availability for the service provider by preventing the outage cause from failing the infrastructure in the first place. Consequently the design aimed at capturing all the twelve envisioned scenarios leaving out natural disasters as these would be impossible to simulate in this environment. For each scenario the simulation aimed to capture output without the AM in place and output with the AM in place. The parameters that were to be captured as output are described in each scenario, just as the inputs and parameter settings are also described per simulation.

In configuring parameters for the simulator it was important to note that the inputs were in line with tutorials found on the CloudSim site (http://www.cloudbus.org, accessed February 2016) and Buyya et al. (2009) the developers of the simulator, namely:

1. Number of hosts be less than number of VMs
2. Number of VMs be less than number of cloudlets
3. Each VM should run at least one cloudlet in a simulation.

Further the algorithms used for configuring the simulator purposed to determine whether for each outage cause the given AMs would keep the infrastructure available, or whether an outage would occur. The form of the algorithm in each scenario was as follows:

1. Configure simulator settings
2. Configure AM
3. Inject outage cause
4. Observe and record results.

3.3.1 Scenario 1 Node Failure versus Node Management

The model suggests that availability is increased by using node management in a proactive manner to counter node failures in the cloud. Node based availability guarantees the availability of individual nodes, such as individual virtual servers, middleware components or hosted application components (Fehling, Leymann, Mietzner, & Schupeck, 2011). Within the scope of this study nodes are represented by the hosts and by extension the VMs found in the datacenter. The Cloud computing layered architecture is represented as shown in figure 3.2.

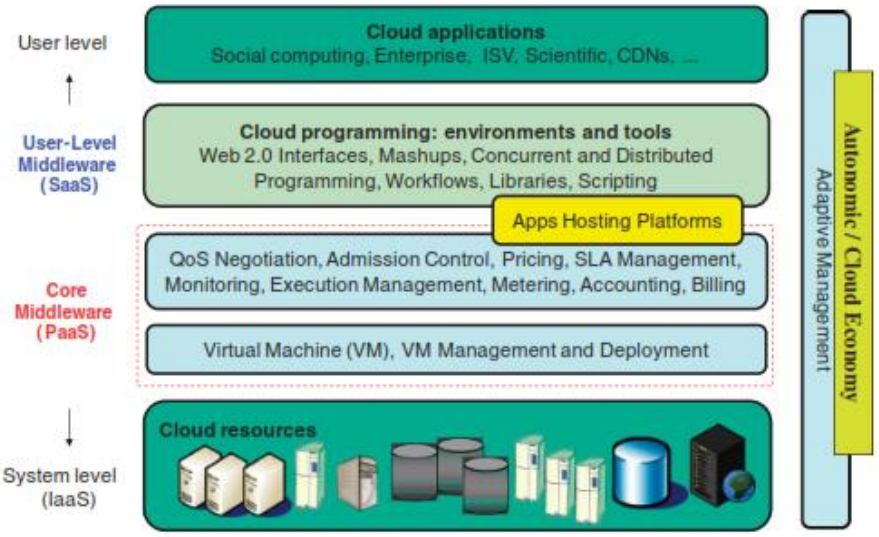

Figure 3.2 Layered Cloud Computing Architecture (Buyya et al., 2009)

Each layer represents a different service as described in chapter two. The FWA model focuses on the infrastructure level components and this is what makes CloudSim an ideal simulator to use to validate this scenario. The CloudSim architecture consists of different classes that can be extended in order to customize scenarios according to the investigator's needs. This requires knowledge of the Java language as the simulator is written in Java. The CloudSim classes can be represented in a class diagram as shown in figure 3.3

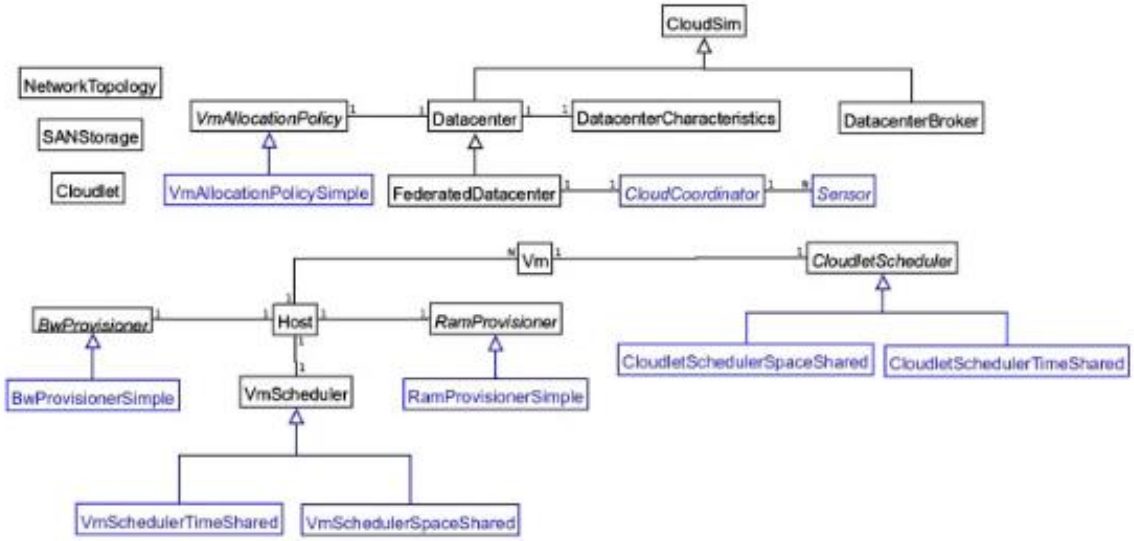

Figure 3.3 CloudSim Class Diagram (Buyya et al., 2009)

The simulator is event driven, and different classes can pass messages to each other. The CloudSim core runs on a discrete event management framework which can be represented in a class diagram as shown in figure 3.4

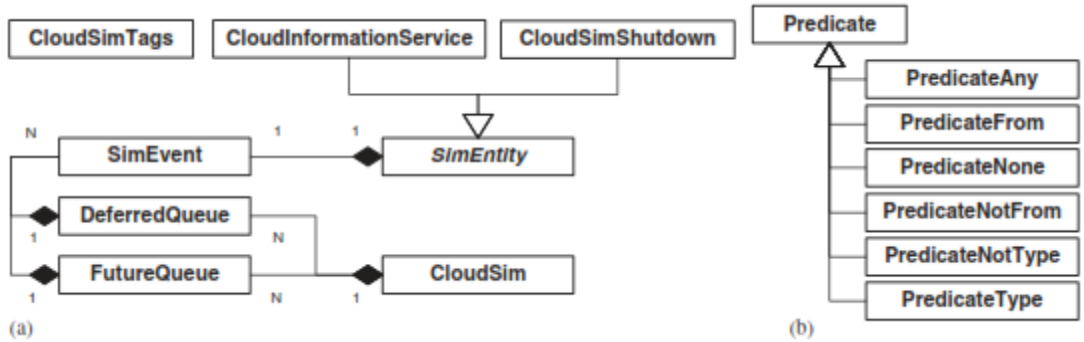

Figure 3.4 CloudSim core simulation framework class diagram. (a) main classes and (b) predicates (Buyya et al., 2009)

As can be seen from figure 3.4 the SimEntity class allows for creation of different entities that are extended from it. Similarly events can be configured by overriding methods found in this

class; while events can also be configured by overriding methods found in the SimEvent class. Both these classes are abstract.

In terms of events that take place the datacenter will register itself with the Cloud Information Service. This service acts like a registry where all datacenters register themselves. A broker acting on behalf of a user will then query the Cloud Information Service for an available datacenter. The Cloud Information Service will return the available datacenter and from then on the broker can communicate directly with the datacenter. The datacenter has a number of hosts within it which are the nodes, which in a real datacenter would typically be represented by the different servers in the datacenter. The host has one or more processing cores called PEs (processing element, performance measured in MIPS), RAM and storage. The broker will then request the datacenter to create the virtual machines which will be used to execute user tasks. The user tasks are called cloudlets. The users are then assigned VMs to execute their tasks (cloudlets). Allocation of VMs on the hosts in the datacenter is done according to a policy; CloudSim by default has two policies: time shared and space shared. The time shared policy allows VMs to be assigned cores on the host according to a particular time slice while space shared allows VMs to be assigned cores according to the capacity of the latter. Users can also create their own provisioning policies according to their needs.

The sequence of events can be captured as follows, and is illustrated in figure 3.5

(1) Datacenter (DC) registers with Cloud Information Service (CIS)

(2) Broker queries CIS for available DCs

(3) CIS allocates DC to broker (in this case DC2)

(4) Users send cloudlets to broker

(5) Broker requests DC to create Virtual Machines (VMs)

(6) Cloudlets assigned to VMs

(7) VMs execute cloudlets according to policy

(8) VMs return results to broker

(9) Broker requests DC to destroy VMs

(10) Broker returns results of cloudlet execution to users

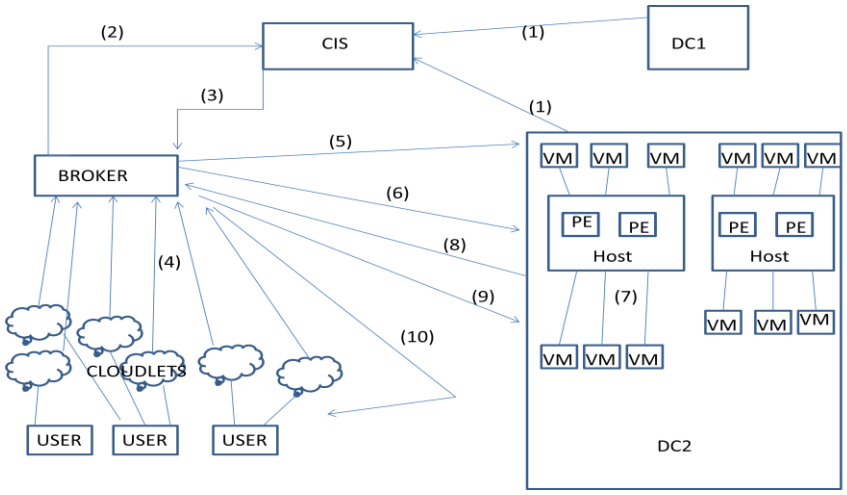

Figure 3.5 Cloudlet execution events in CloudSim

As the FWA model is designed to work in a heterogeneous environment the simulation environment had to be configured as such; this was done by ensuring that the hardware (hosts) were all of varying capacity in terms of PEs and RAM. The simulation was then set up such that it can resemble a real world scenario by ensuring that hosts fail in a random manner using a random number generator.

Simulation 1: this was done by creating a HostFaultInject entity that extends the SimEntity class. In the first procedure the random number generator randomly failed several hosts which ensured that the entity was working. This was done by scheduling an event that would describe how the host failure event would occur, with delay between hosts failing based on the random number generator, and a customized tag that described how the host would fail. When failing hosts it was necessary to ensure that not only the host fails but also the VMs within the host and the PEs. There was no VM migration allowed during the simulation. When a cloudlet completed execution on a VM then it was sent back to the broker who then requested for the VMs to be destroyed once all cloudlets had been executed in the simulation. At the user end the

measure of availability of the cloud was in terms of the tasks (cloudlets) that had been executed. Thus the number of hosts that failed and the number of cloudlets executed were recorded in the simulation.

This sequence of events can be captured in simple form as follows:

1. The Broker sends cloudlets to the datacenter;
2. While the random number generator stays on, the following events occur:
   a. Hosts are failed randomly,
   b. VM Migration is switched off
   c. VMs in the host are failed
3. Send the cloudlets back to the broker

The parameter settings that were configured in CloudSim to perform the simulation are shown in table 3.2, while the applicable tag and event is shown under the table.

Table 3.2 Scenario 1 simulation 1 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300
						pesNumber = 1;

CloudSim tag: HOST_FAILURE; Event: schedule(getId(), delay, HOST_FAILURE)

Simulation 2: Nodes are managed according to some provisioning policy. It is the cores in the nodes that provide the VMs with the execution environment and thus when a node fails it goes along with the VMs that had been created in it. The question arises then as to what happens to the cloudlets being executed in it? Without node management the cloudlets will not be executed and will not be seen in the cloudlet list that the broker produces showing the executed cloudlets. Node management then means that the datacenter should allow VMs to be migrated to other hosts according to some policy. Once the VMs have been migrated to other hosts then cloudlet execution can continue and the user should receive the results of the execution. In this simulation VM migration was enabled in the host, and by using applicable methods at both datacenter and host level the status of tasks could be monitored during the simulation, and thus the cloudlets executed. The same policy, i.e. space-shared policy for the VMs in the host objects was maintained. With the policy enabled Simulation 1 was repeated and the number of hosts failed and cloudlets executed were recorded.

This sequence of events can be captured in simple form as follows:

1. The Broker sends cloudlets to the datacenter;
2. While the random number generator stays ON, the following events occur:
    a. Hosts are failed randomly,
    b. VM Migration is switched ON
    c. Call the method that allows the VM migration to occur
    d. VMs in the host are failed
    e. VMs are migrated to the next available host
    f. Continue processing
3. Send the cloudlets back to the broker

The parameter settings that were used in CloudSim to perform this simulation are shown in table 3.3, while the methods used are mentioned under the table.

Table 3.3 Scenario 1 simulation 2 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	Hosts	VMs	Cloudlets

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Methods:

- addMigratingInVm(Vm vm) – in host
- processCloudletMove(int[] receivedData, int type) – in datacenter and host
- updateVMProcessing() – in datacenter and host

Policy: space – shared.

3.4.2 Scenario 2 Node Failure versus Cluster Management:

A cluster is a group of computers that are connected together such that they appear to be one to the user. When multiple computers are linked together in a cluster, they share computational

workload as a single virtual computer. From the users view point they are a single virtual machine, though in reality they are multiple machines. The users' requests are received and distributed among all the standalone computers to form a cluster. This results in balanced computational work among different machines, improving the performance of the cluster systems (Sadashiv & Kumar, 2011). In a datacenter the nodes work as a cluster via provisioning policies that dictate what should happen should a node fail in the cluster; the policies also state how the workload should be distributed among the nodes in order to increase efficiency. In the absence of these policies the nodes simply work as standalone and tasks would ultimately take a longer time to execute. In ideal situations the cluster will consist of similar machines with similar characteristics and operating systems; this makes it easier to manage and also to make it easier to distribute the workload among the machines that make up the cluster. A typical cluster architecture is shown in figure 3.6

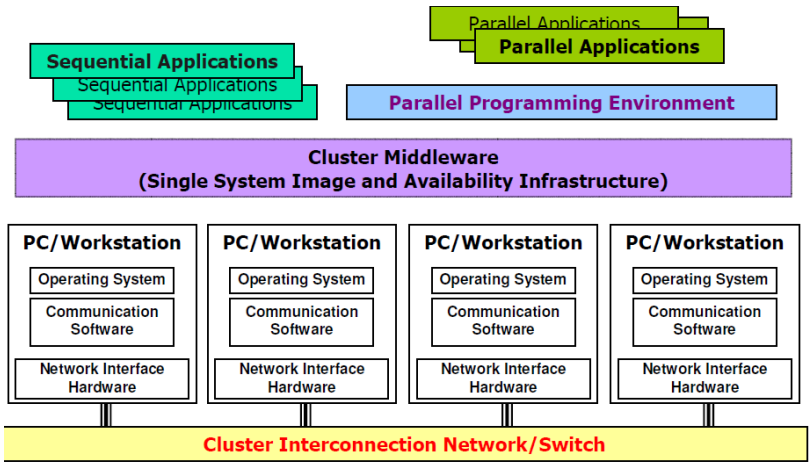

Figure 3.6 Cluster architecture (Buyya, 1999)

In CloudSim cluster characteristics are set up in the datacentercharacteristics class via the resource architecture attribute of the class. It would be expected that without proper cluster management then individual nodes would not function together and this would produce an error when it comes to processing cloudlets as they would not appear to be one.

Simulation 1: The failing of nodes was repeated using the same procedure in scenario 1 and results observed and recorded (hosts failed, cloudlets created and executed).

1. The Broker sends cloudlets to the datacenter;
2. While the random number generator stays ON, the following events occur:
   a. Hosts are failed randomly,
   b. VM Migration is switched OFF
   c. VMs in the host are failed
3. Send the cloudlets back to the broker

The parameters that were used in CloudSim to enact this simulation are shown in table 3.4, while other applicable information is found under the table.

Table 3.4 Scenario 2 simulation 1 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw= 10000;
					50
					size = 10000; //image size (MB);
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM="Xen";//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

CloudSim tag: HOST_FAILURE

Event: schedule(getId(), delay, HOST_FAILURE)

Cluster characteristics: in datacentercharacteristics class

Attribute: resource architecture

VMAllocation Policy: space – shared

Simulation 2: It is important to note that the datacentercharacteristics class is abstract and as such the datacenter class is the one that is managing the cluster. Cluster management in this instance would then imply following the same configuration and approach as in scenario 1 since it would imply moving tasks among the nodes in the cluster such that the user will only see the results of the execution itself and not which node actually performed the task; essentially the difference here being that the migration is in form of load balancing policy across the cluster, while in simulation 2 of scenario 1 the migration was based on migrating the VMs to the next available node. In order to achieve this node failure was configured such that the VMs in each node would not fail but be migrated to other working nodes; thus this would ensure the cloudlets being executed in them would receive the PEs required to complete execution. Further the allocation policy was changed to time shared to accommodate the same MIPS rating as in simulation 1 of this scenario (retaining space shared resulted in a VM creation error due to MIPS). The same parametric results as in simulation 1 were recorded. This can be shown in the following steps:

1. The Broker sends cloudlets to the datacenter;
2. While the random number generator stays ON, the following events occur:
    a. Hosts are failed randomly,
    b. VM Migration is switched ON
    c. Call the method that allows the VM migration to occur
    d. VMs in the host are failed
    e. VMs are migrated according to load balancing policy
    f. Continue processing
3. Send the cloudlets back to the broker

The parameter settings that were used in CloudSim to perform the simulation are shown in table 3.5, while other pertinent information regarding the simulation is mentioned under the table.

Table 3.5 Scenario 2 simulation 2 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	Hosts	VMs	Cloudlets

1	5	2 - for dual core machines	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					50
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

CloudSim tag: HOST_FAILURE

Event: schedule(getId(), delay, HOST_FAILURE)

Cluster characteristics: in datacentercharacteristics class

Attribute: resource architecture

VMAllocation Policy: time – shared

3.4.3 Scenario 3 Configuration Issues versus Cluster Management

Early detection is the key to minimizing failure damage induced by configuration errors, especially those errors in configurations that control failure handling and fault tolerance (Xu & Zhou, 2015). In cloud computing the datacenter needs to be configured in such a way that all the resources required by the users are available in accordance with the agreed service level agreement (SLA). A configuration issue arises when any of the resources becomes unavailable due to a change in some policy or a failure of a component. Yin et al. (as cited in Xu & Zhou, 2015) classify software configuration errors into the following three categories:

- Parameter: erroneous settings of configuration parameters (either an entry in a configuration file or a console command);
- Compatibility: configuration errors related to software incompatibility;
- Component: the other remaining configuration errors (for example, missing a specific software module).

Further Xu and Zhou (2015) posited that most of the existing research efforts focus on parameter configuration errors, because they account for the majority of real-world configuration errors (the authors cited, for example, 70.0%–85.5% of the studied configuration error cases in Yin et al.)

Simulation 1: in this simulation an erroneous setting of PE (Processing Element) configuration settings in the machines in the datacenter was done by intentionally leaving out two zeros in the MIPS (Millions of Instructions Per Second) processing power such that the MIPS = 10 in the method used to create the machines in the cluster, as well as their characteristics in the datacenter; all other configurations were left untampered. Tasks were then sent to the datacenter and the number of cloudlets executed was observed and recorded. This can be represented using the following steps:

1. Create datacenter

2. Configure the datacenter characteristics,
3. Use the following parameter settings in CloudSim: Hosts = 10, VMs = 30, PE = 2 (dual core servers in the datacenter), RAM = 16384 MB,
4. Introduce a configuration error of MIPS = 10 at cluster level
5. Let the Broker send cloudlets to the DC

The parameter settings used in CloudSim for the above simulation are shown in table 3.6, while the applicable method in the simulator is mentioned under the table.

Table 3.6 Scenario 3 simulation 1 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines	10	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name)

VMAllocation Policy: time – shared

Simulation 2: In this procedure the configuration error was corrected and MIPS set to 250 and then to 1000 and cloudlet execution observed and recorded while maintain the remaining configurations of simulation 1.

The parameter settings of CloudSim for the simulation where MIPS was set to 250 are shown in table 3.7

Table 3.7 Scenario 3 simulation 2 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines	250	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of CPUs
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name)

VMAllocation Policy: time – shared

The parameter settings of CloudSim for the simulation where MIPS was set to 1000 are shown in table 3.8

Table 3.8 Scenario 3 simulation 3 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	Hosts	VMs	Cloudlets

1	5	2 - for dual core machines	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name); VMAllocation Policy: time – shared

3.4.4 Scenario 4 Configuration Issues versus Node Management

On the conjugate side of the scenario 3 in the model it is suggested that configuration issues can also be prevented by proper node management.

Simulation 1: In this simulation an erroneous setting of RAM configuration settings in the machines in the datacenter was done by setting RAM = 163 in the method that is used to create the machines and their characteristics in the datacenter; all other configurations were left untampered. Tasks were then sent to the datacenter and the number of cloudlets executed was observed and recorded. The steps followed are shown in the algorithm below:

1. Create datacenter
2. Configure the datacenter characteristics
3. Use the following parameter settings for CloudSim: Hosts = 10, VMs = 30, PE = 2 (dual core servers in the datacenter), PE = 4 (quad core servers in the datacenter), MIPS = 1000,
4. Configure an error at node level of RAM = 163 MB
5. Let the Broker send cloudlets

All the parameter settings that were used in CloudSim for this simulation are shown in table 3.9

Table 3.9 Scenario 4 simulation 1 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines
		4 – for quad core machines
			1000	10
				ram = 163; //host memory (MB)
				long storage = 1000000; //host storage
				bw=10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH= 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name); VMAllocation Policy: time – shared

Simulation 2: The configuration error was adjusted and RAM set to 512 MB for each machine and the resulting simulation results recorded, i.e. the parameter results that were captured in simulation 1. The steps followed are captured in the algorithm below:

1. Create datacenter
2. Configure the datacenter characteristics
3. Use the following parameter setting for CloudSim: Hosts = 10, VMs = 30, PE = 2 (dual core servers in the datacenter), MIPS = 1000,
4. Configure an error at node level of RAM = 512 MB
5. Let the Broker send cloudlets

The full parameter settings that were used in CloudSim for this simulation are shown in table 3.10

Table 3.10 Scenario 4 simulation 2 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines
		4 – for quad core machines
			1000	10
				ram = 512; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name); VMAllocation Policy: time – shared

Simulation 3: The configuration error was adjusted and RAM set to 16384 MB for each machine and the resulting simulation results recorded. The parameter settings that were used in CloudSim for this simulation are shown in table 3.11

Table 3.11 Scenario 4 Simulation 3 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	2 - for dual core machines
		4 – for quad core machines
			1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

Method: createDatacenter(String name)

VMAllocation Policy: time – shared

3.4.5 Scenario 5: Resource Exhaustion versus Limit Detection Policy

Chapter two identified resources as networks, servers, storage, applications, and services. From the user perspective resources allow the user to utilize the cloud services in a scalable manner, i.e. the more tasks the user needs to complete the more resources are availed by the cloud and *per se* the datacenter. User tasks are handled by the virtual machines in the datacenter and neither the user nor the broker knows which VM is handling which task; the broker simply submits the tasks to the datacenter and returns the result of execution to the user. Resource exhaustion occurs when the resources required to execute an action are entirely expended, preventing that action from occurring; indeed Antunes, Neves, and Verissimo (2008) opined that resource exhaustion can be caused due to bad design, inefficient utilization of resources on the service side, and resource leakage. In the cloud this means that resources required to execute a cloudlet are expended and thus it is not executed. A threshold policy would help in ensuring that workloads are balanced dynamically and thus resource exhaustion does not occur; therefore ensuring efficient utilization of resources. At infrastructure level this would involve implementation of virtual machine policy to ensure the number of virtual machines running on the same host is below or at the threshold level. In CloudSim there are three policies in place that implement different forms of balancing.

Simulation 1: In this simulation 10 hosts were configured to run 30 VMs. The scenario was set up to have five users send 500 tasks (cloudlets) to the datacenter. The VM scheduling policy used was VMScheduler space shared policy. This policy is a VM allocation policy that allocates one or more PE (processing element or CPU core) to a VM, and doesn't allow sharing of PEs. If there is no free PEs to the VM, allocation fails. The simulation was run and the number of VMs created, cloudlets executed and time observed and recorded. The steps are shown in the algorithm below:

1. Count all VMs
2. Count all PEs available
3. Allocate VMs to PEs
4. If all PEs are allocated, stop allocation
5. Send message no available PE

6. Let Broker send cloudlets

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.12

Table 3.12 Scenario 5 simulation 1 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Space – shared

Simulation 2: In this simulation 10 hosts were configured to run 30 VMs. The same five users then sent 10000 tasks (cloudlets) to the datacenter. The VM scheduler space shared policy was used again and the simulation repeated. The number of VMs created, cloudlets executed and

time observed and recorded. The parameter settings that were used in CloudSim to enact this scenario are shown in table 3.13

Table 3.13 Scenario 5 simulation 2 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						10000
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Space – shared

Simulation 3: In this simulation the same setup as for simulation 1 was used. The only difference made was that the VM scheduler policy was changed to a time shared policy. This policy allocates one or more PE to a VM, and allows sharing of PEs by multiple VMs. The same parameters, i.e. number of VMs created, cloudlets executed and time observed and recorded. The steps used in the simulation are shown in the algorithm below:

1. Count all VMs
2. Count the available PEs
3. Allocate VMs to PEs
4. If all PEs have been allocated, repeat allocation for remaining VMs to share PEs
5. Let Broker send cloudlets

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.14

Table 3.14 Scenario 5 simulation 3 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					30
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Time – shared

Simulation 4: In this simulation the same setup as for simulation 2 was used. The VM scheduler policy used this time was the time shared policy. The number of VMs created, cloudlets executed and time observed and recorded. The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.15

Table 3.15 Scenario 5 simulation 4 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**Hosts**	**VMs**	**Cloudlets**

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					50
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						10000
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Time – shared

Simulation 5: In this simulation the setup for simulation 1 was repeated using VM scheduler time shared over subscription policy. This policy allows over-subscription. In other words, the policy still allows the allocation of VMs that require more CPU capacity that is available. Further, each virtual PE cannot be allocated more CPU capacity than MIPS of a single PE. The simulation was repeated and number of VMs created, cloudlets executed and time observed and recorded. The simulation steps are shown in the algorithm below:

1. Count the number of VMs
2. Count the number of PEs available
3. Allocate the VMs to the PEs
4. If all PEs have been allocated, repeat allocation for the remaining VMs to share PEs
5. If any VMs power is less than the required power,
    a. Check available CPU capacity
    b. If available allocate to the VM
6. Let Broker send cloudlets

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.16

Table 3.16 Scenario 5 simulation 5 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	Hosts	VMs	Cloudlets

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					50
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						500
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Time shared over subscription

Simulation 6: In this simulation the setup for simulation 2 was repeated using VM scheduler time shared over subscription policy. The same parameters, i.e. number of VMs created, cloudlets executed and time observed and recorded. The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.17

Table 3.17 Scenario 5 simulation 6 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	Hosts	VMs	Cloudlets

1	5	Mixed (2 for dual core and 4 for quad core)	1000	10
				ram = 16384; //host memory (MB)
				long storage = 1000000; //host storage
				bw = 10000;
					50
					size = 10000; //image size (MB)
					RAM = 512; //vm memory (MB)
					MIPS = 250;
					BANDWIDTH = 1000;
					pesNumber = 1; //number of cpus
					VMM = "Xen";
					//VM Manager
						10000
						length = 40000;
						fileSize = 300;
						outputSize = 300

VM Scheduling Policy: Time shared over subscription

3.4.6 Scenario 6: Security Issues versus Checkpointing

Data breaches and data loss are considered top threats to cloud computing and the measures that are put in place to mitigate them are interlinked. For example the decision to encrypt the data stored on cloud can reduce the possibility of a data breach but the corruption of the encryption key can lead to data loss (Racuciua & Eftimie, 2015). The FWA model is concerned with availability of data and therefore focus is on data loss as opposed to data breaches. In a data

breach the data can be accessed and compromised but not necessarily lost; conversely, data loss implies the loss of data such that it is not available. The model opines that a security breach of this nature can be countered using checkpointing. As described by Singh et al. (2012) a checkpoint is a local state of a job saved on stable storage. Checkpoints work like restore points for an operating system such that the status of a process can be saved at consistent intervals so that if there is failure computation can be resumed from the earlier checkpoints, thereby avoiding restarting execution of the job from the beginning again. This implies that should the data be lost at some point then it can be recovered by restoring from the last checkpoint. To demonstrate checkpointing and recovery, FTCloudSim designed by Zhou et al (2013) was used. FTCloudSim is an extension of CloudSim that adds six modules as seen in figure 3.7

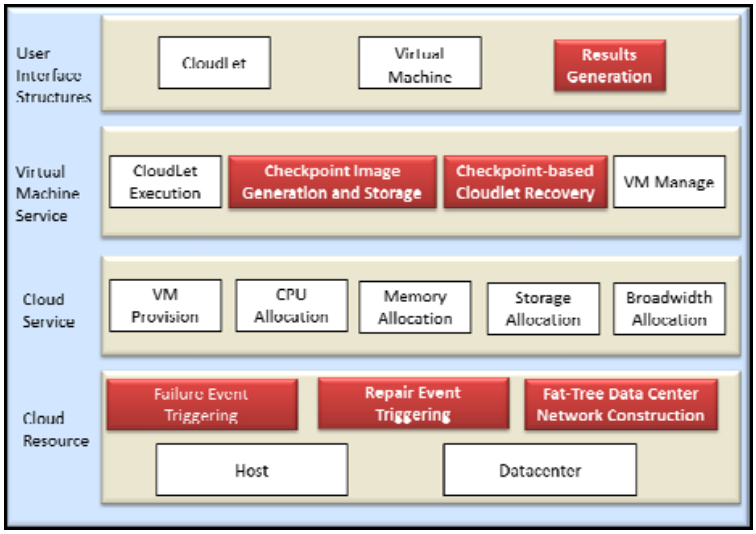

Figure 3.7 FTCloudSim architecture (Zhou et al, 2013)

The architecture fitted the purposes of this study due to two modules not found in the original CloudSim architecture, namely checkpoint image generation and storage, and checkpoint-based cloudlet recovery. As the names suggest the former module is responsible for generating checkpoint images at specific times and storing them to a database while the latter is responsible for recovering cloudlets from a checkpoint image.

The checkpointing process can be summarized in an algorithm as follows:

1. Broker sends cloudlets
2. While the VM is processing cloudlets
    a. Image is sent to checkpoint image generation and storage at programmed intervals of time up to the point when the last cloudlet is executed
    b. Image is then stored in database
3. If a cloudlet's data has been lost
4. Recover latest image using checkpoint-based cloudlet recovery from storage database
5. Send the image to the VM processing that cloudlet
6. VM processes the cloudlet

Simulation 1: In this simulation 4 hosts were configured to run 50 cloudlets. The tasks were set to run. Next random failures were generated using a normal distribution and the results from the event log and database were saved. The sequence of events can be shown in the algorithm below:

1. Broker sends cloudlets
2. VM run tasks
3. While tasks are running
    a. Tasks are failed randomly using normal distribution
4. Save results

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.18, with other information regarding the simulation found under the table.

Table 3.18 Scenario 6 simulation 1 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	RAM	Hosts	VMs	Cloudlets
1	1	Mixed (2 for dual core and 4 for quad core)	1000	4096 //host memory (MB)	4	8 size = 10000; //image size (MB) RAM = 512; //vm memory (MB) MIPS = 250; BANDWIDTH = 10000; pesNumber = 1; //number of cpus VMM = "Xen"; //VM Manager	50 length = 40000; fileSize = 300; outputSize = 300

Classes: TaskGeneration, FailureEventGeneration

Database: Db1.accdb

VMScheduler: time – shared

Simulation 2: In this procedure several classes were used to create checkpoints and recover the failed tasks from the checkpoint. From the literature Singh et al. (2012) had proposed that a checkpoint interval of five seconds is optimal and this is the interval that was used in all

simulations involving checkpointing. In order to recover failed tasks they have to be stored in a local database, and one such database was configured using MS Access software. The status of each task was monitored so that tasks could be stored at various stages of execution, and parameters for scheduling the recovery of cloudlets were set. The events were then run and results from event log and the created database logged for analysis. The algorithm for this process is as follows:

1. Broker send cloudlets;
2. While cloudlets are being processed
    a. Schedule checkpoints
    b. Check cloudlet status
    c. Invoke checkpoint
    d. Store in database
3. If a cloudlet shows it status as failed, then
        a. Recover the cloudlet from last checkpoint
4. Else process execution
5. Return processed cloudlet to broker.

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.19, with other information regarding the simulation found under the table.

Table 3.19 Scenario 6 simulation 2 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	RAM	Hosts	VMs	Cloudlets
1	1	Mixed (2 for dual core and 4 for quad core)	1000	4096 //host memory (MB)	4	8 size = 10000; //image size (MB) RAM = 512; //vm memory (MB) MIPS = 250; BANDWIDTH = 10000; pesNumber = 1; //number of cpus VMM = "Xen"; //VM Manager	50 length = 40000; fileSize = 300; outputSize = 300

Classes: AppCheckPoint, CheckPointMakingScheduler, CheckPointStorageIndex, CloudletRecoveryScheduler

CloudSim tags: INVOKE_CHECKPOINT, SCHEDULE_APP_CHECKPOINT and SCHEDULE_SYSTEM_CHECKPOINT

Database: Db1.accdb

VMScheduler: time – shared

3.4.7 Scenario 7 Resource Exhaustion versus Checkpointing

Resource exhaustion occurs when the resources required to execute an action are entirely expended, preventing that action from occurring. In the context of the FWA model checkpointing as a countermeasure for resource exhaustion was not practically possible. This is because at infrastructure level the state of a task can be recovered using checkpointing but not the state of a resource. This pairing will be discussed more in chapter four.

3.4.8 Scenario 8 Security Issues versus Limit Detection Policy

This was not possible to simulate using the available instruments and will be discussed further in chapter four. Further a limit detection policy would not be applicable in the event of data loss due to corruption arising from an attack.

3.4.9 Scenario 9 Hardware Issues Vs Component Redundancy

There are several hardware components that may fail in the datacenter and these include memory modules, cables, switches and routers. Using the current setup this was not possible to simulate and would require different configurations for each of the hardware devices mentioned above. However, it was possible to simulate failure at the node level as in scenario 1. Hardware failure at the server level is represented by failure of the cores in the datacenter. The cores provide the environment for the hosts which in turn provide the VMs which perform task execution. Failure at the core level directly implies failure at all levels above it. Component redundancy means having a redundant core available to take over the tasks should the current core fail. With the current simulator this would be implemented by allowing the VMs to be moved to the redundant server (hosts) in the event that the current server fails. This is depicted in figure 3.8

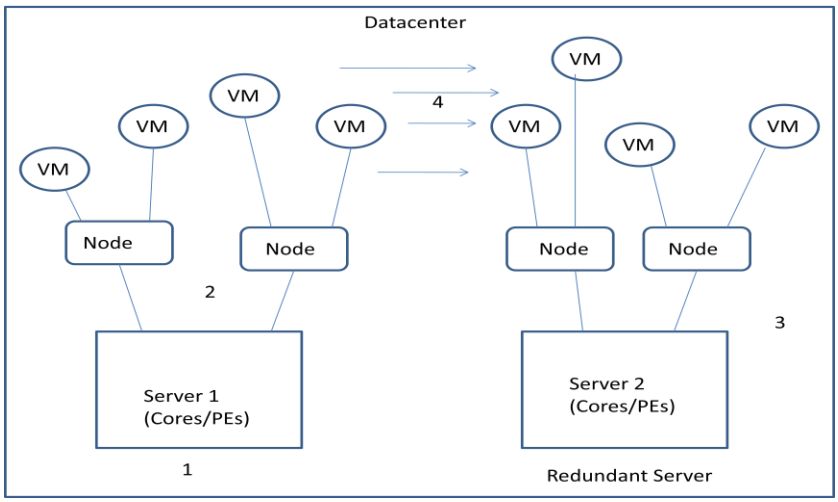

Figure 3.8 Component redundancy

Step 1: Server 1 fails

Step 2: Nodes in server 1 fail

Step 3: Server 2 takes over

Step 4: VMs are migrated to server 2

Control of the servers and VM migration happen at datacenter and host levels; the controller at datacenter level monitors all the hosts and once it detects failure it immediately moves the VMs to the redundant server and processing of tasks continues.

For simulation purposes it was found that the configuration of simulation 2 of scenario 1 where node management was invoked as a counter to node failure, was similar to the simulation required for this simulation since the migration methods in the datacenter and at host level are the same, and failure at core level also implied failure at the node level.

3.4.10 Scenario 10 Network issues versus Active-X variant

The Active –X variant is inspired by the heartbeat program designed for high availability in Linux environments described in detail in the section 2.5.5 of this study. Summarily the Active –X variant works in such a way that, in an active-passive environment when one server fails (the active one) then the other server (passive one) immediately takes over and thus continuity is assured. In an active-active environment both servers are active and share workloads from users

based on some load balancing algorithm. In the Linux variant the changeover takes less than half a second. This system is shown in figure 3.9

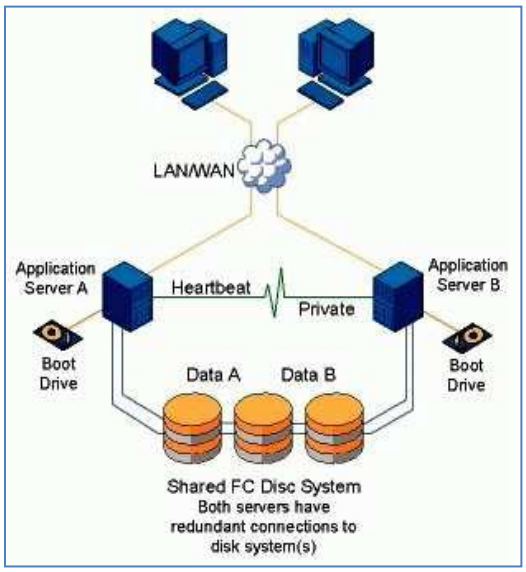

Figure 3.9 Dual server heartbeat system (linux-ha.org)

Simulation 1: In this simulation two datacenters were created with 4 cores (processing elements) and 4 hosts. The server in one datacenter was then configured to totally fail randomly using the HostFaultInject entity used in scenario 1. The details of the tasks executed were then recorded. The algorithm of scenario 1 simulation 1 (node failure) applies here. The parameter settings of CloudSim used in this simulation are shown in table 3.20

Table 3.20 Scenario 10 simulation 1 parameter settings of CloudSim

Datacenter	Users	PEs	MIPS	RAM	Hosts	VMs (20 in total for 2 Datacenters)	Cloudlets

0	2	(4 - for quad core machines) 4 cores in total	1000	4096	2	20
						size = 10000; //image size (MB)
						RAM = 512; //vm memory (MB)
						MIPS = 250;
						BANDWIDTH = 1000;
						pesNumber = 1; //number of cpus
						VMM = "Xen";
						//VM Manager
							100
							length = 40000;
							fileSize = 300;
							outputSize = 300

CloudSim tag: HOST_FAILURE

Event: (schedule(getId(), delay, HOST_FAILURE)

Simulation 2: In this simulation the second server in the other datacenter was configured such that when the servers (in datacenter 0) failed then all tasks were redirected to the servers (in datacenter 1). The method used in ensuring cloudlet execution in this simulation mimics an active- passive setup. The details of the tasks executed were then recorded for analysis. The algorithm explaining the steps is shown below.

1. Broker sends cloudlets to DC0 (the first datacenter) and to DC1 (the second datacenter),
2. While the random number generator is on perform the following steps,
    a. Fail hosts in datacenter 0,
    b. Allow VM Migration,
    c. If a host has failed,
    d. Migrate its VMs to DC1 (the second datacenter)
3. Send cloudlets to the broker

The parameter settings of CloudSim that were used to enact this simulation are shown in table 3.21

Table 3.21 Scenario 10 simulation 2 parameter settings of CloudSim

Datacenter	**Users**	**PEs**	**MIPS**	**RAM**	**Hosts**	**VMs (Total = 20)**	**Cloudlets**

1	2	(2 - for dual core machines)
		(4 - for quad core machines)
		(This is the active server)
			1000	4096	2	size = 10000; //image size (MB)
						RAM = 512; //vm memory (MB)
						MIPS = 250;
						BANDWIDTH = 1000;
						pesNumber = 1; //number of cpus
						VMM = "Xen";
						//VM Manager
							100
							length = 40000;
							fileSize = 300;
							outputSize = 300

Datacenter	**Users**	**PEs**	**MIPS**	**RAM**	**Hosts**	**VMs (Total = 20)**	**Cloudlets**

1	1 (datacenter must have at least one user when running the main program)	(2 - for dual core machines)
		(4 - for quad core machines)
		(This is the passive server)
			1000	4096	4	size = 10000; //image size (MB)
						RAM = 512; //vm memory (MB)
						MIPS = 500;
						BANDWIDTH = 1000;
						pesNumber = 2; //number of cpus
						VMM = "Xen"
							As received from server 1 in datacenter 0

Policy: failover policy (custom policy)

3.4.11 Scenario 11 Network Issues versus Component Redundancy

Component redundancy as a measure against network issues implies having two servers one acting as a redundant solution for the other. In designing such a simulation the configuration would be the same as in scenario 9 and thus the results will be the same. The analysis applicable to scenario 9 would be correspondent of this scenario. However, an additional simulation was performed using the same configurations, but redundancy in this case was enabled using a load balancing policy across the datacenters. This was done to check the effect of having both servers sharing the workload. The server in one datacenter was then failed, and VMs migrated to the remaining datacenter.

1. The Broker sends cloudlets to both datacenters;
2. While the random number generator stays ON, the following events occur:
    a. Hosts in datacenter 0 are failed randomly,
    b. VM Migration is switched ON
    c. Call the method that allows the VM migration to occur
    d. VMs in the host are failed
    e. VMs are migrated to datacenter 1 according to load balancing policy
    f. Continue processing
3. Send the cloudlets back to the broker

3.4.12 Scenario 12: Hardware versus Active-X Variant

The same principle as in scenario 9 (hardware issues vs component redundancy) would apply here; however, in the case of failure at server level it would be expected that the results obtained from scenario 10 (network issues vs Active-X variant) would be applicable. This will be discussed further in chapter four.

3.5 Summary

This chapter described the research design and methodology used to test the FWA model. The methodology involved using simulations, performed using CloudSim toolkit which is a deterministic simulator that is used widely in research and investigation of various cloud computing theories. CloudSim was used since using a real environment would not be practical or cost effective. The parameter settings in CloudSim that were used for all the simulations were described in tables and the theory behind the configurations explained using figures and algorithms; twelve scenarios were envisioned and described in this chapter. Each scenario tested the effect of an AM on outage cause as per the model; conjugate AMs were also swapped to test their effects on the corresponding outage cause and the results saved for evaluation purposes.

The methodology described in this chapter purposed to address part of the fourth specific objective of this thesis namely:

4. Evaluate the performance of the model by measuring its service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters.

   Testing is part of the evaluation process. The model was implemented in CloudSim, and all the configured scenarios tested whether applying the specific AM to the outage cause would result in the desired outcome, i.e. the system remaining available despite the outage cause being effected.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 Introduction

This chapter describes the results that were obtained from the design setups in chapter three. The chapter presents the results of each simulation, makes observations based on the results, and discusses them. Further service availability as well as execution availability are defined and explained, and their respective measures in the simulations are also computed and presented.

4.2 Scenario 1: Node Failure versus Node Management

This section presents the results for scenario 1. This scenario purposed to test whether node management is a suitable AM for node failure. In cloud computing the nodes are the hosts, which in turn create the VMs. The VMs are the ones that perform execution of tasks (cloudlets). Two simulations were configured and run. In the first simulation node failure was configured and the simulation was allowed to run without any node management. In the second simulation node management was configured and the same parameters used in the first simulation (node failure) were allowed to run.

4.2.1 Simulations

The results for both simulations are presented below. In the first simulation the simulator was configured to force nodes to fail randomly using a random number generator. In the second simulation node management was introduced and simulation ran again.

Simulation 1 Observations (no node management)

In this simulation the simulator was configured to create the hosts and the VMs in them. To simulate a heterogeneous environment the hosts were configured using different cores namely dual and quad cores. The dual core hosts created two VMs in them while the quad core hosts created four VMs in them. Figure 4.1 shows the number of VMs that were created per host.

Host Ids with even numbers (0, 2, 4, 6 and 8) were created in quad core machines, hence creating four VMs each in them. Host Ids with odd numbers (1, 3, 5, 7 and 9) were created in dual core machines and had two VMs each in them.

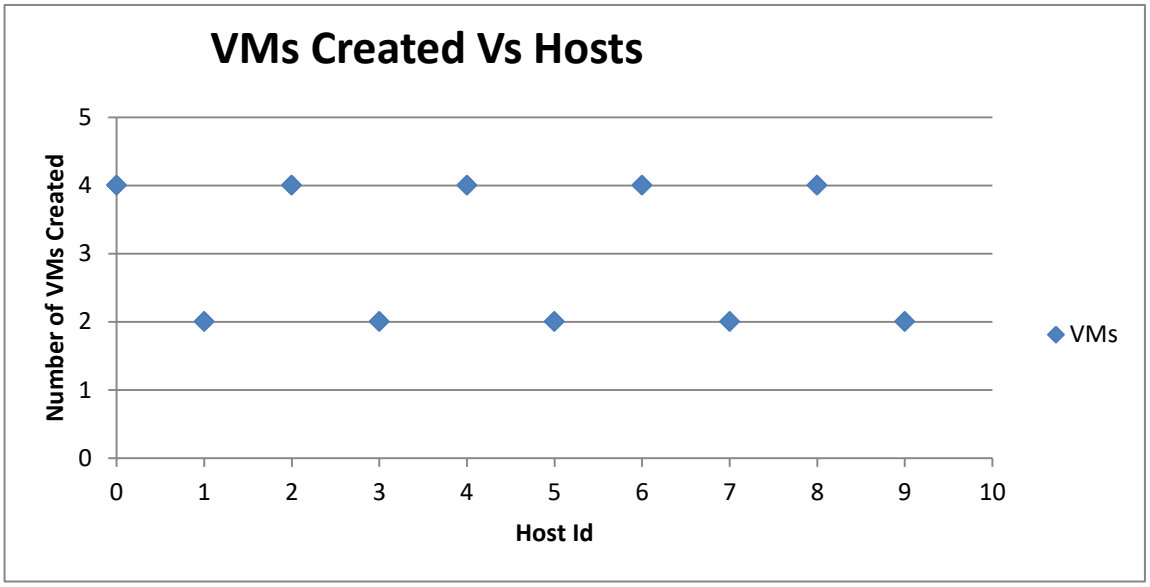

Figure 4.1 Hosts and VMs created

The simulator was configured to forcefully fail hosts using a random number generator. Figure 4.2 shows the number of hosts that failed randomly during the simulation run. It was observed that the hosts started failing at the two second mark with host Id 1 being the first one to fail. At the four second mark three hosts failed simultaneously, i.e. host Id 2, 4 and 8. By the end of the simulation all 10 hosts had failed.

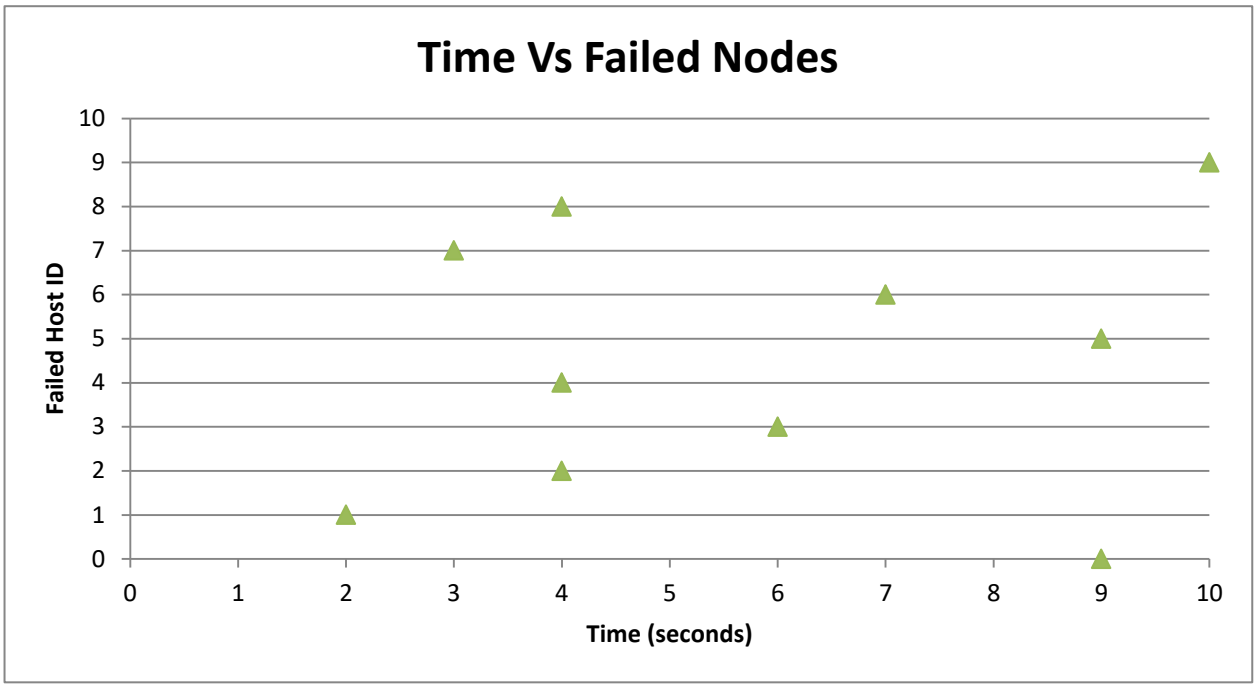

Figure 4.2 Failed nodes during simulation

Five hundred cloudlets were then sent by the broker to the datacenter during the simulation. Individual cloudlets are identified by their Id number and the cloudlets are assigned to the available VMs. Figure 4.3 shows the allocation of cloudlets to VMs during the simulation. The graph shows that all the cloudlets were allocated VMs with the last cloudlet being allocated to VM Id 19.

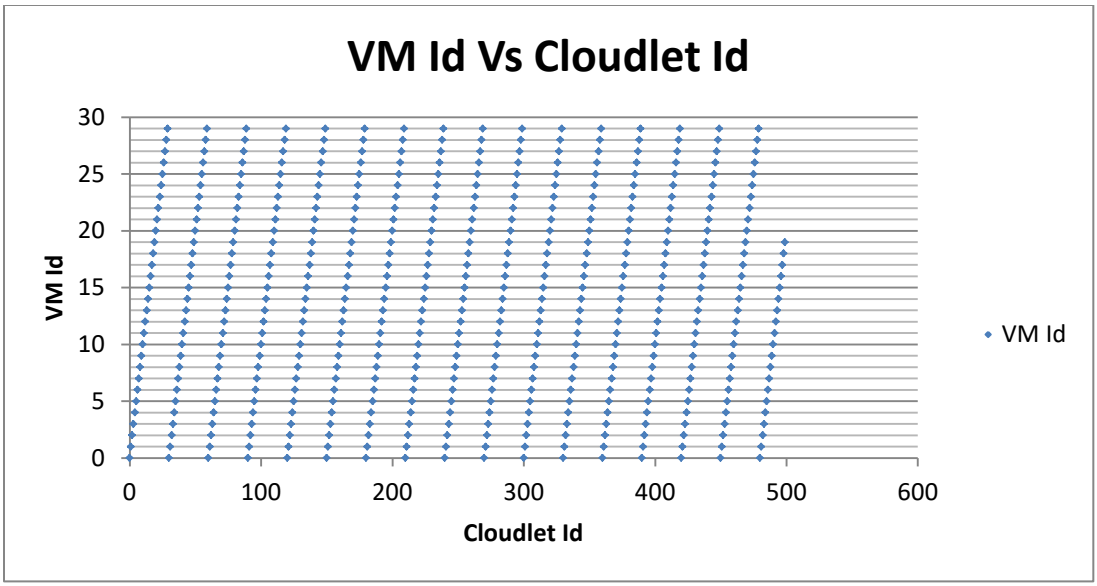

Figure 4.3 Cloudlet allocation to VMs

Simulation 2 Observations (node management)

In the second simulation the simulator was configured to fail the hosts randomly but node management was introduced to counter the failure of the nodes. Figure 4.4 shows the number of VMs that were created by the individual hosts. It was observed that the dual core machines created two VMs each while the quad cores created four VMs each. The hosts with odd host Id numbers (1,3,5,7 and 9) were on the dual core machines , while the remaining hosts were on quad core machines.

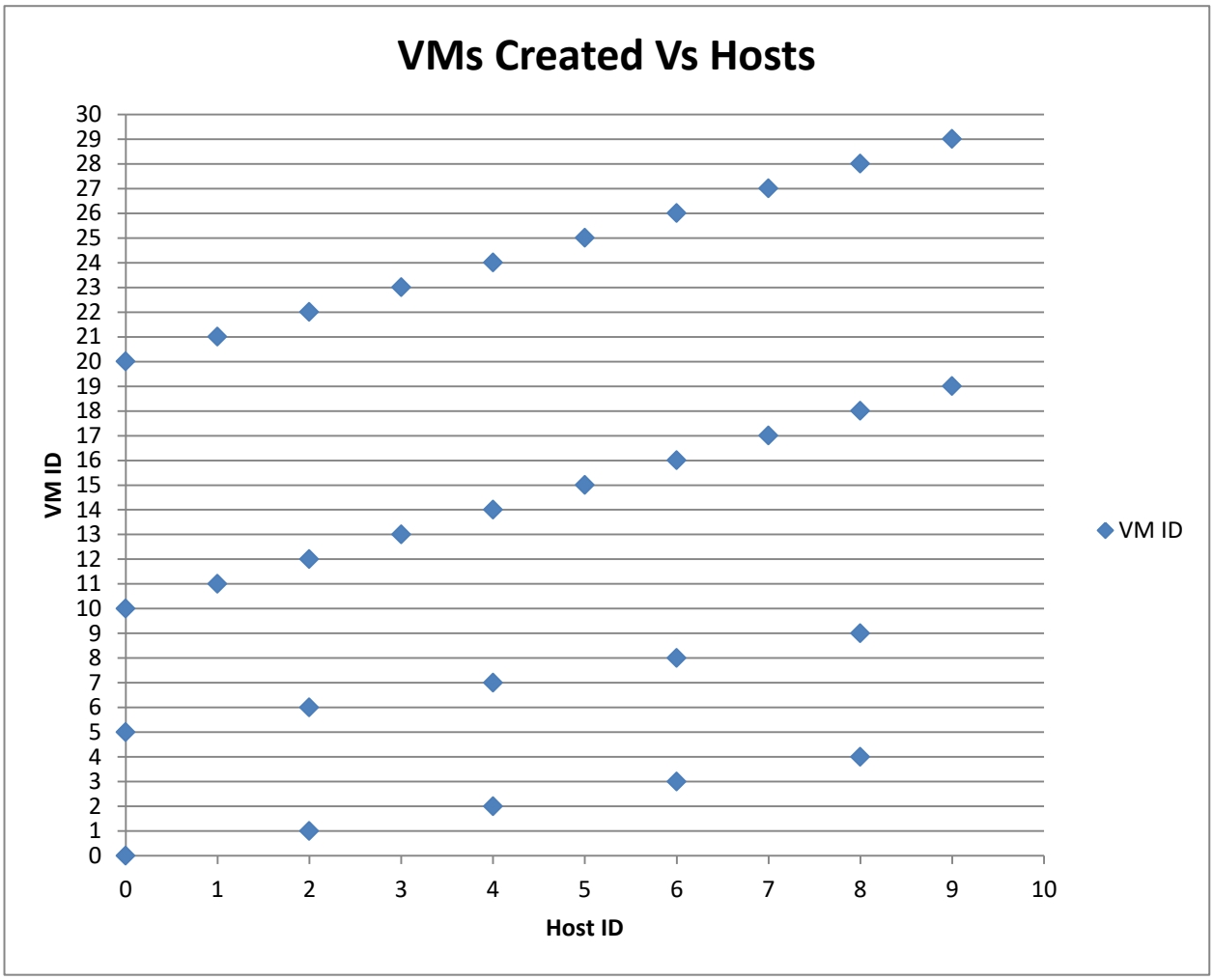

Figure 4.4 VMs created per host

The hosts were then configured to fail randomly over time using a random number generator. Figure 4.5 shows the points in time when the different hosts failed. Host Id 8 was the first to be failed while host Id 2 was the last one to be failed. The figure shows the random order of the failing of hosts spread over the simulation.

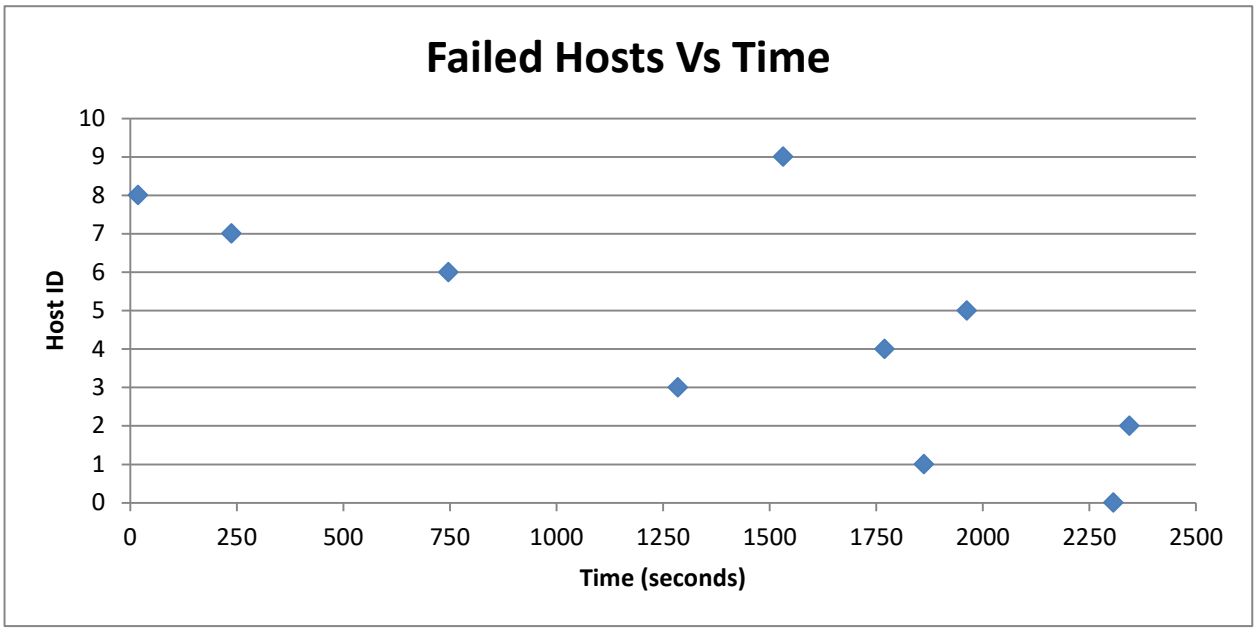

Figure 4.5 Failed hosts over time

The cloudlets that were executed by the individual VMs is shown in figure 4.6. It was observed that all the five hundred cloudlets were executed without exception during this simulation. The figure shows that VM Id 0 upto VM Id 19 each executed seventeen cloudlets with the remaining VMs, i.e. Id 20 to Id 29 each executing sixteen cloudlets. This makes up a total of five hundred executed cloudlets. The reason for the lack of uniformity in number of cloudlets executed by all the VMs is due to the allocation policy which is described in section 4.2.2.

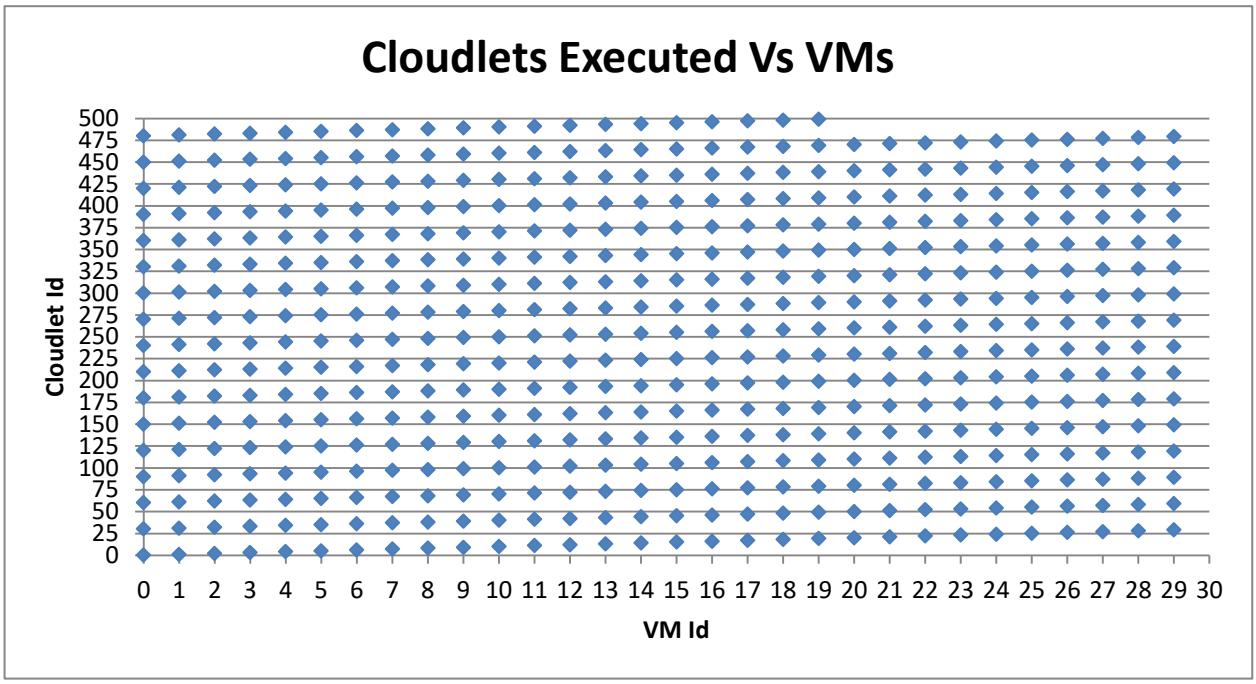

Figure 4.6 Cloudlets executed per VM

The hosts on the dual core machines took a longer time to execute the cloudlets assigned to them, compared with the hosts on the quad core machines. The execution time for each of the machines is shown in figure 4.7. All the hosts in the quad core machines took 2560 seconds to execute the cloudlets assigned to them, while those in dual cores took 2720 seconds to execute the cloudlets assigned to them.

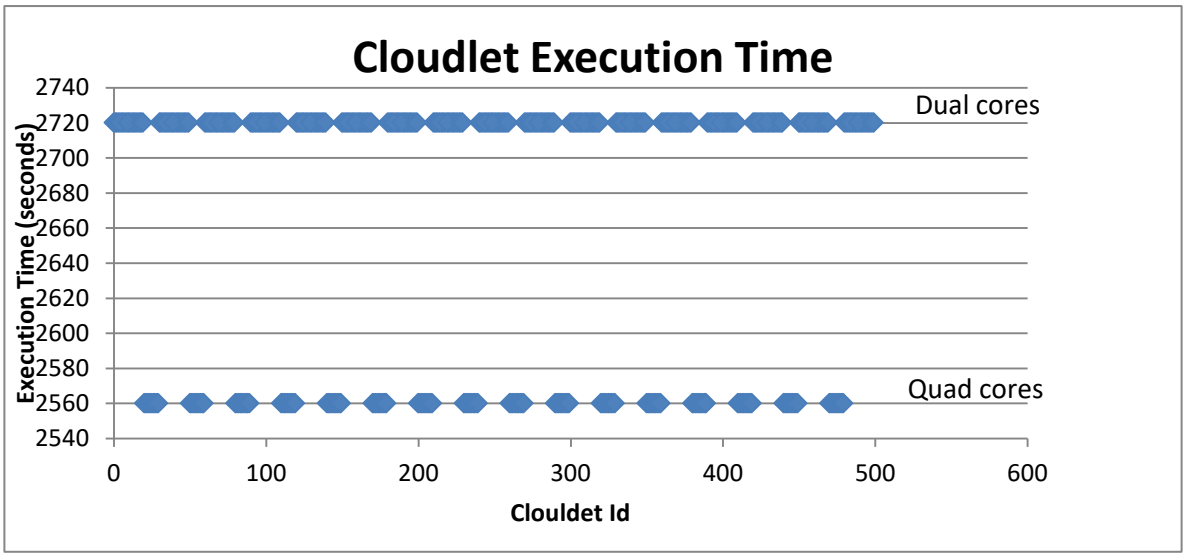

Figure 4.7 Cloudlets executed over time

4.2.2 Discussion

In simulation 1 of this scenario there was no node management and hosts were forcefully failed using a random number generator. During the simulation it can be seen from figure 4.2 that the hosts failed randomly over a 10 second period, for example at the 4 second mark three hosts all failed at the same time, i.e. host ids 2, 4 and 8, while at the 10 second mark only host id 9 failed. The random failing of hosts was done purposely to reflect the dynamic nature of the cloud. In a real world environment it would be expected that hosts would not fail in a linear or some preordained manner, but rather randomly. As seen in figure 4.1 the results also show that despite VMs being created in the host this did not have any effect on cloudlet execution, or lack thereof. All 30 VMs were created during the simulation. Figure 4.3 also shows that the broker did send all the 500 cloudlets to the VMs that had been created, and allocation was done. However, as seen in figure 4.2 the hosts started failing at the 2.0 second mark. This resulted in no cloudlet being returned to the broker from that point on to the end of the simulation. The reason for this is that while the VMs were created in the hosts, there was no mechanism to move them as their respective hosts failed. Essentially this means that the cloudlets all terminated in the VMs that they had been assigned to. The assignment of cloudlets to the VMs occurs according to the assigned policy, in this case a simple round robin policy (Buyya et al., 2009) that allocates first cloudlet to the first VM (VM #0), next one to the next VM (VM #1), and so on, till all cloudlets have been assigned. This explains why VM Id 0 to VM Id 19 in figure 4.3 had more cloudlets allocated to them than the others. As seen in figure 4.1 the hosts in the dual core machines each created 2 VMs while those in the quad core machines created 4 VMs each. The cores provide the processing power for the hosts. The hosts create VMs in them according to the processing power and task requirments. Figure 4.8 demonstrates the node failure process in four steps:

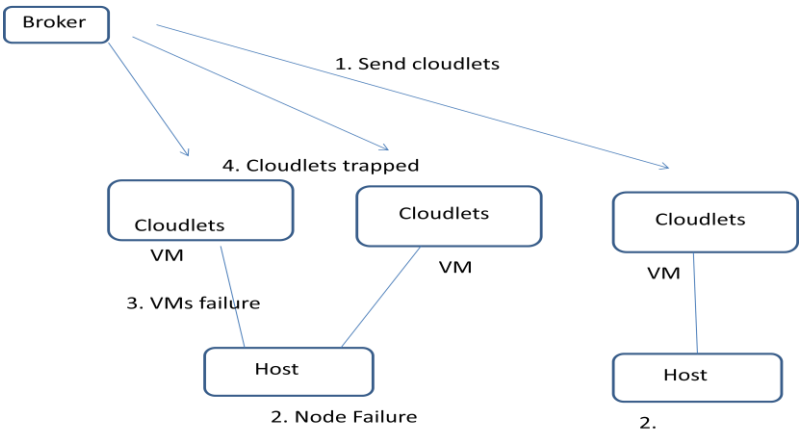

Figure 4.8 Node failure

Step 1: Broker sends the cloudlets to the the VMs created in the hosts.

Step 2: Nodes are failed.

Step 3: VMs fail

Step 4: Cloudlets get trapped in the VMs.

Simulation 1 showed node failure resulted in zero availability of the datacenter to the broker and cloudlets; though the cloudlets were sent none of them were executed by the VMs in the datacenter. Jose (2013) defined service availability as the ratio of the resources allocated to the resources requested. Simulation 1 showed that resources may be allocated but this does not necessarily mean that execution of tasks will occur. Figure 4.3 in simulation 1 showed that all cloudlets were allocated to the VMs. However, despite the cloudlets being allocated to the VMs none of them were executed. This means that the service may be available but this does not necessarily mean that cloudlets will be executed. Jose (2013) examined availability from a service perspective, assuming that if the resources requested are allocated then this would imply availability of the infrastructure to execute tasks. Figure 4.3 disputes this notion since simulation 1 clearly showed that resources requested were allocated but execution did not occur. From a user perspective this equates to non-availability of the infrastructure since the user does not get back results of the tasks. There is therefore a need to introduce a parameter that will measure availability in terms of execution. This parameter should measure the ratio of tasks executed to tasks requested. When this parameter is used together with service availability defined by Jose

(2013) it should give a better pictue of the availability of a cloud infrastructure. The parameter introduced is execution availability. Execution availability is defined as follows:

Let the number of cloudlets executed = μ;
Let the number of cloudlets requested = λ;

Execution availability $(\mathcal{E}) = \mu/\lambda$

Service availability as defined by Jose (2013) is computed as follows:

Let the number of resources allocated = $R_{A;}$
Let the number of resources requested = $R_{R;}$

Service Availability $(\eta) = R_A/R_R$

In this scenario service availability changes over the whole period of the simulation. Indeed in most cases the value of η will change over time and thus its value will be dynamic. From figure 4.5 it is assumed the resource being measured is the node itself as it is the one that will provide the service of cloudlet execution. Table 4.5 shows the changing value of η over time during the simulation.

Table 4.1 Service availability over time

S/No	Time (seconds)	Resources failed (total)	Resources Allocated	Service Availability (η)
1	500	2	8	8/10(80%)
2	1000	3	7	7/10 (70%)
3	1500	4	6	6/10 (60%)
4	2000	7	3	3/10 (30%)
5	2500	10	0	0

The service availability ratio depends on what time it is measured. Nonetheless as the nodes were failing together with the VMs then the service availability percentages will still remain the same even when measuring the VM as the requested resource since the ratio of the resource requested versus the resource allocated is 100%. This study posits that service availability alone would be an erroneous measure to use since no execution occurs and thus from a user's point of view the service was not available to complete the requested tasks (cloudlets).

In simulation 2 involving node management, the same configuration was used only this time the VM migration was allowed by configuring a VM migration policy at datacenter level. As was shown in figure 4.4 all VMs were created, with 2 VMs being created in the hosts residing in the dual core machines and 4 VMs in the hosts residing in the quad core machines. Figure 4.5 showed the result of failing the hosts using the random number generator. The failures were this time spread over a longer time period but still randomly. Nonetheless, as seen from figure 4.5 all the hosts failed during the simulation period. Figure 4.6 showed all the executed cloudlets, while figure 4.7 showed cloudlets executed over time. It was observed that in this simulation all the cloudlets were executed and returned to the broker; further, hosts that were running on quad core took shorter time to execute cloudlets (2560 seconds) compared to those running on dual cores (2720 seconds). In this simulation VM migration was enabled at datacenter level. VM migration allows the VM to be migrated to a different host in the event of failure by some configured policy. As discussed in the literature VMWare Inc have used this technique in development of their VMwareHA solution (2007). The solution enables VMs to be migrated from a host in case it fails, to another host. There are two types of VM migration that are used: hot (live) migration and cold migration. When the former is used service will not be interrupted during migration, while with the latter users are likely to notice the service interruption. In live migration the VM is transferred together with its state (Kaur and Rani, 2015). This is the type of migration used in this simulation. This ensures that the jobs (cloudlets) in it are not affected and thus they are executed. Figure 4.9 demonstrates the migration policy at datacenter level:

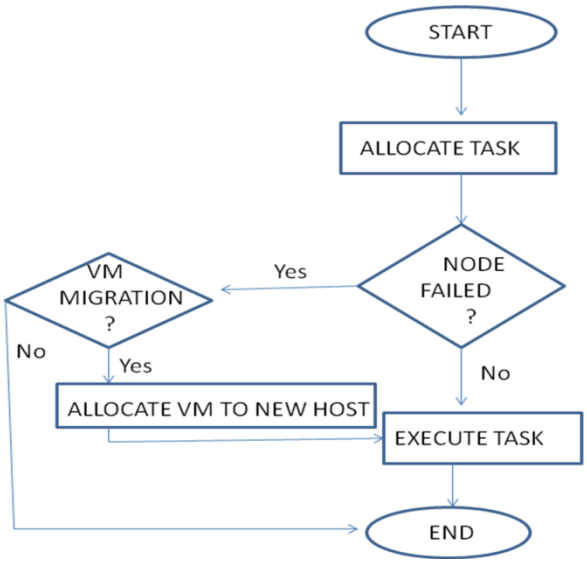

Figure 4.9 VM migration

The engine behind the cloud computing paradigm is actually virtualization more than anything else. This is because users are offered virtualized services at their level and the infrastructure runs on hosts where VMs perform the actual job execution. While Kaur and Rani (2015) describe the details of the two types of VM migration this study's focus was on whether node management is indeed a proactive approach to countering node failures in the cloud as per the FWA model. There are many ways to implement node management but for the purposes of this study VM migration was chosen for the following reasons: when a node fails it is the VMs in it that will cause unavailability of the resources; VMs are the driving engines as far as execution of cloudlets is concerned and VM migration policies can be set proactively. Another approach to consider in managing nodes would be to investigate what causes the nodes to fail in the first place and build for availability from this point; however, this is beyond the scope of this study. Nodes may also be managed by using an active-passive node approach similar to the Linux HA approach (linux-ha.org); however, this too will have to be investigated further as it is beyond the scope of this study. In terms of service availability if the nodes are defined as the resources then the pattern would be similar to that in simulation one; decreasing percentages of service availability. However, since the broker requests for VM creation depending on the tasks at hand, then the resource requested in this instance is the VM itself. Further since all VMs requested

were available throughout the simulation then service availability is at 100%. In terms of execution availability all cloudlets sent were executed and sent back to the broker.

Service availability = 100%; Execution Availability = 100%

4.3 Scenario 2: Node Failure versus Cluster Management

This section presents the results for scenario 2. This scenario purposed to test whether cluster management is a suitable AM for node failure. In cloud computing clusters can be configured at both physical and virtual levels. A collection of nodes (at physical level) or a collection of hosts (all with same configuration) will constitute a cluster. The nodes are the physical servers that house the hosts, which in turn create the VMs. In the first simulation node failure was configured and the simulation was allowed to run without any form of cluster management. In the second simulation cluster management was configured and the same parameters used in the first simulation (node failure) were allowed to run.

4.3.1 Simulations

The results for both simulations are presented below. In the first simulation the simulator was configured to force nodes to fail randomly using a random number generator.

Simulation 1 Observations (no node management)

Similar observations and results described in simulation one of section 4.2.1 where nodes were randomly forcefully failed was observed in this simulation. The only difference was that since all machines were dual core they all produced 2 VMs each as opposed to creating 4 VMs, as was the case in scenario one (section 4.2).

Simulation 2 Observations (cluster management)

Similar observations and results described in simulation two of section 4.2.1 where node management was introduced were observed in this simulation. The only difference here was that cloudlet execution took 2720 seconds for all the machines since there were no quad core machines in this scenario like in scenario one.

4.3.2 Discussion

The failing of nodes was described in section 4.1.2. This scenario examined whether cluster management may also increase availability in cases of node failures; this is in line with the proposed characteristics of the FWA model which suggest that opposite AMs are conjugate in nature and that outage causes can swop AMs on opposite sides of the model. This simulation therefore intended to examine whether cluster management can be used in place of node management to counter node failures. In this scenario the cluster was configured using dual core machines only as a cluster would require all machines to be of the same configuration (similar characteristics and operating systems). This configuration makes it easier to manage and distribute the workload among the machines that make up the cluster. The simulation run showed that while cloudlets were sent to the datacenter and random nodes failed the nodes completed cloudlet execution since they were communicating with each other to enable load balancing. The two policies that were available were time-shared policy and space-shared policy. The difference between the two was described by Hamani and Sidhu (2014) and these are both policies that are used to allocate tasks to VMs in the hosts. The main difference is that whereas both use the same round robin policy of task allocation described earlier, the former schedules all tasks at the same time and allocates time slices to each task, while the latter simply executes one task after another regardless of time; the allocation policy used in this simulation was time-shared. This study's scope was focused on whether cluster management does indeed increase availability proactively. Thus using either time-shared or space-shared policy in task allocation was not going to affect the objective of the simulations in this scenario since both policies use the round robin policy of task allocation. The results indicate service availability and execution availability measures the same as for node management in section 4.2.

The high availability solution in HA-OSCAR proposed by Thanakornworakij et al. (2012) was described in chapter two; their solution using an enhancement of HA-OSCAR produced an availability of 0.99999. In HA-OSCAR the clusters kept executing tasks by ensuring that there was redundancy from the head node all the way down to the switches sending tasks to the clusters. This ensured that the tasks reached the clusters and that they were executed by the clusters but did not examine the possibility of nodes in the cluster failing. Further, even the HA solution provided by heartbeat described in the literature only envisions a redundant solution

should a failure occur; the strength of this being in how quickly the changeover occurs. The cloud takes cluster computing to a new level by the introduction of cluster virtualization. This allows configuration of a cluster at both physical and virtual levels. The configuration of the cluster at both levels is referred to as cluster management. In the cloud ,however, at infrastructure level this study focused on cluster management at the virtual level; the physical level configuration was invoked by using machines (cores) of the same configuration. From a user perspective the execution of tasks is done by VMs and what lies beyond them is not of concern to the user save for the fact that their tasks are executed. Virtualization is indeed one of the biggest strengths of the cloud.

From the FWA model perspective it was proposed that node failures can be managed proactively using node management and on the conjugate side, cluster management. The simulations suggest that proper node management does increase both service and execution availability. Further the results of the simulation using cluster management also indicate that this is an effective AM against node failure. Interestingly the results suggest that effective management of the individual node does result in effective management of the cluster itself, be it in a shared-nothing or shared-everything environment; further the converse also appears to be true, that is, effective management of the cluster results in effective management of the individual nodes as a whole. This implies that node failures are effectively countered from an availability perspective using either node management or cluster management.

4.4 Scenario 3: Configuration Issues versus Cluster Management

This section presents the results for scenario 3. This scenario purposed to test whether cluster management is a suitable AM for configuration issues. In the first simulation a deliberate error in MIPS setting of the machines (cores) in the cluster was made and the simulation was allowed to run without any form of cluster management. In the subsequent simulations the value of MIPS (processing power) was adjusted upwards till the required setting of 1000 MIPS was achieved.

4.4.1 Simulations

There were three simulations performed. Each simulation, but one, allowed for a deliberate error in configuration settings for the machines processing power setting (MIPS). The MIPS setting for each simulation was as follows:

- Simulation 1: MIPS = 10
- Simulation 2: MIPS = 250
- Simulation 3: MIPS = 1000

Simulation 1 Observations (MIPS = 10)

In this simulation requests for VMs were not fulfilled, i.e. VM creation failed in all the hosts

Allocation of VM failed by MIPS

Simulation 2 Observation (MIPS = 250)

In this simulation the MIPS setting for the cores was adjusted upward to 250. The initial request had been for the creation of 30 VMs. The datacenter created only 20 as opposed to 30. Figure 4.10 shows the VMs that were created by each host. It was observed that VM Id 0 to VM Id 19 were created by the 10 hosts, making a total of 20 VMs.

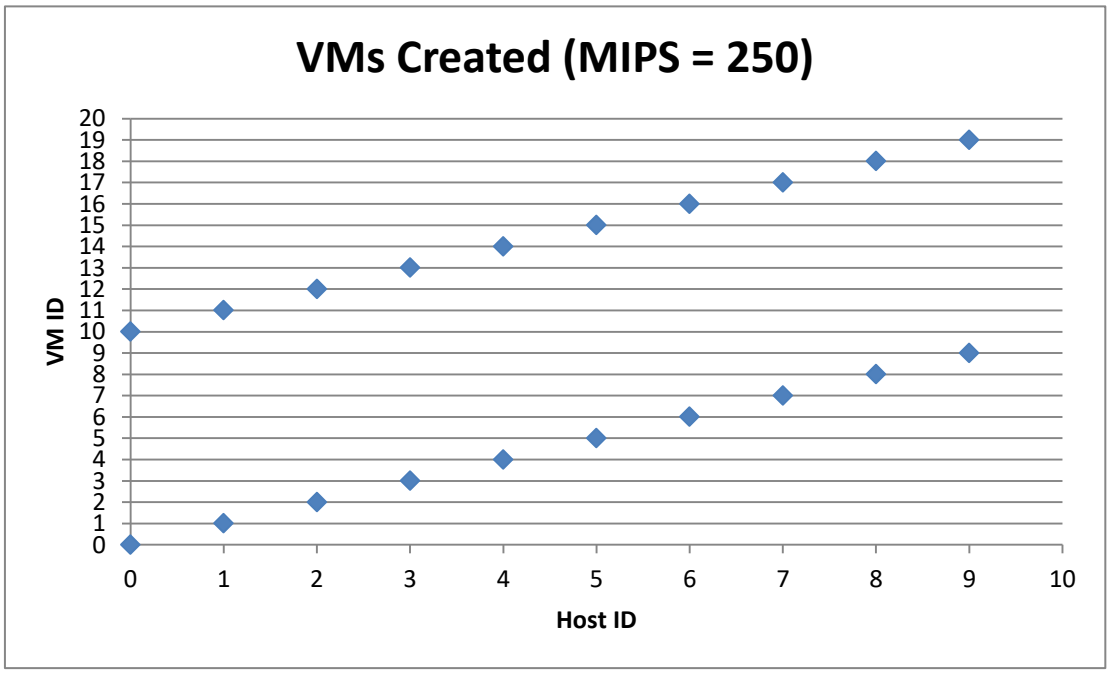

Figure 4.10 VMs created

Figure 4.11 shows the VMs that were not created during the simulation. These are VM Id 20 to VM Id 29.

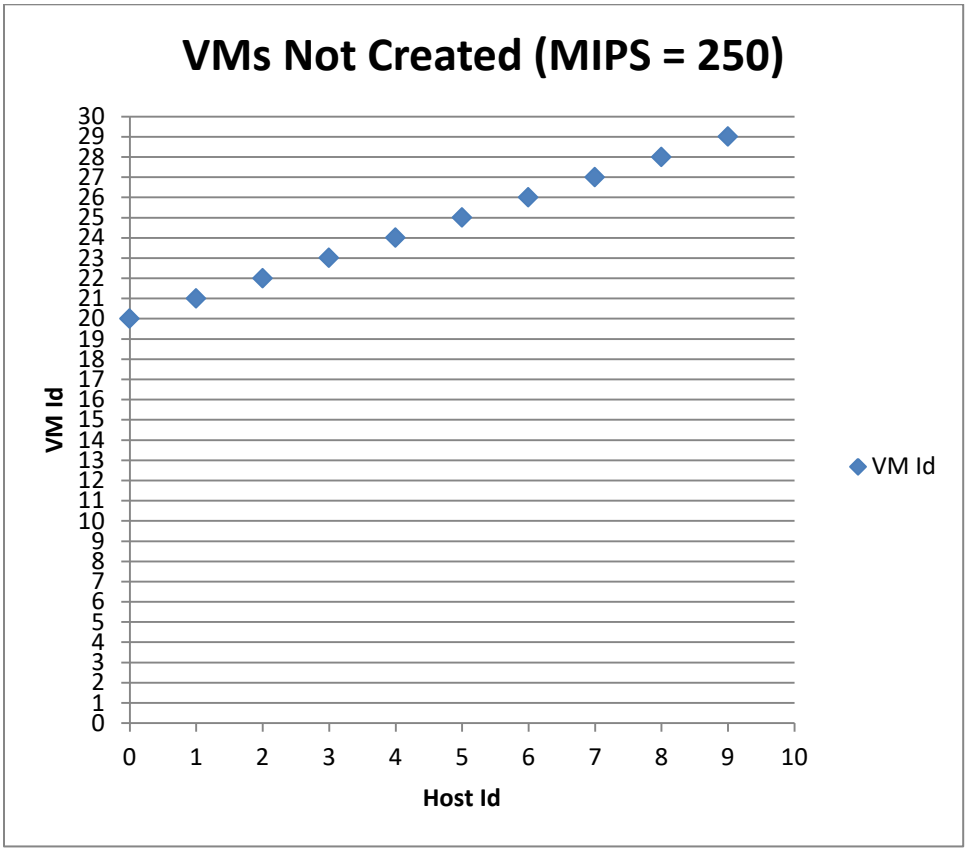

Figure 4.11 VMs not created

This simulation can be summarized as follows:

- VMs created 20
- VMs failed: 10
- Cloudlets executed: ALL
- Execution time: 4000 seconds

Simulation 3 Observations (MIPS = 1000)

In this simulation the MIPS setting was adjusted to 1000 which was the required setting for the machines. Figure 4.12 shows that all the 30 VMs were created as per the requirements.

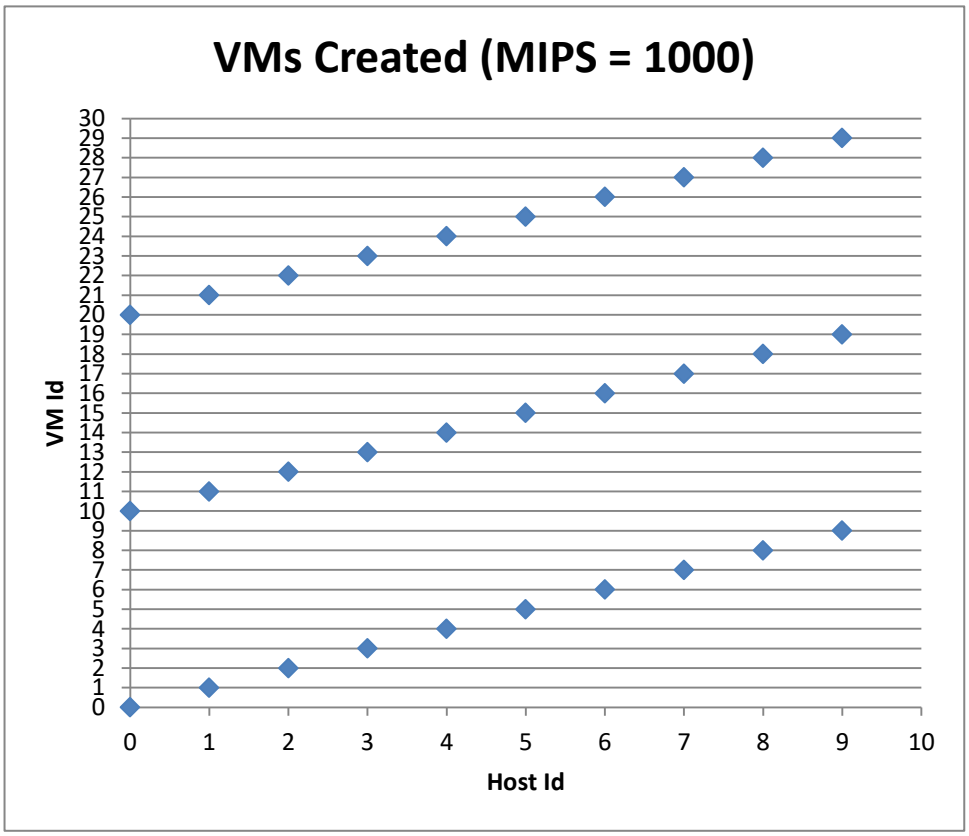

Figure 4.12 VMs created

The simulation may be summarized as follows:

- All VMs were created (30 VMs in total)
- Execution time: 2640 seconds
- Cloudlets executed: ALL

4.4.2 Discussion

Configuration errors were identified in the literature as outage causes. Human errors inevitably produce these configuration errors. The simulations show the extent to which configuration errors may cause lack of availability of the cloud infrastructure.

In the first simulation simply by configuring an MIPS of 10 both the service and execution availability becomes zero percent. This is because the processing power of the machines is too low to create any VM. Table 4.2 presents the results of the three simulations.

Table 4.2 Cluster MIPS configuration settings vs availability

Simulation Number	Cluster configuration setting (MIPS)	VMs created	VMs failed	Service Availability (η)	Execution Availability (Ɛ)
1	0	0	30	0	0
2	250	20	10	20/30 (66.67%)	500/500 (100%)
3	1000	30	0	30/30 (100%)	500/500 (100%)

As is observed with the configuration error in simulation 2, the service availability by the end of the simulation was 66.67%; however, it is inferred that the value of η was 66.67% from the point when the broker sent the cloudlets since the 10 VMs were not created in the first place. The value of Ɛ for simulation 2 was also measured at one, meaning all cloudlets requested were executed. In the third simulation η was measured at 100% as all the VM resource was available throughout the simulation since all VMs were created. Further Ɛ was observed at 1. The significance of juxtaposing η with Ɛ is to give a better idea of the overall availability of the infrastructure. As was pointed out in section 4.2.2 the value of η was at 100% at one point yet no cloudlet execution occurred, thus measuring the two together gives a better insight into overall availability of the infrastructure. Further both values affect the user in terms of the time s/he takes to receive results of their tasks. The service provider needs to ensure that the infrastructure is fast if they are to keep users who would not like to tap their fingers on a desk waiting for the results of a task.

In this case cluster management had been suggested as an AM to counter configuration issues by the FWA model. In this instance the configuration issue has been caused by a typo due to human error, as most configuration errors are. The contention that arises is whether the error was part of the cluster configuration, and if so how could it be prevented in the first instance? As it stands the error was only observed during the output in both simulations 1 and 2 and correcting it was a reactive rather than a proactive measure. The model aims to increase availability from a proactive perspective and therefore proactive measures are more desirable. However, it is worth noting that configuration errors are some of the hardest errors to deal with. This is because a configuration error in one setting may destabilize more than one area of the infrastructure, for example, performance degradation may require investigating more than the MIPS settings of the infrastructure. Xu and Zhou (2015) had suggested the use of configuration free systems where users do not have to key in any configurations; this is restrictive and does not allow infrastructure engineers to make their own configurations. The most practical approach to this would be in restricting input when configuring the cluster during installation, thus making it impossible for erroneous configuration entries. This may be done by use of input masks and/or validation alerts in the configuration scripts and programs at user interface level. Practically an input mask would restrict values that the user enters when configuring the cluster, for example, a code may be put in place that restricts the MIPS rating for the cluster to be not less than say 1000. The validation alert would tell the user that the value they have entered is invalid, and suggest the range of values valid for that particular parameter setting. To make it even more restrictive infrastructure engineers may also opt not to allow users to enter certain parameters and make these settings part of an input script to be picked directly by the server; unfortunately, this too isn't free of human error completely, making the input mask a more practical solution.

It is not conclusive as to whether cluster management is an effective AM for configuration issues; however, the simulations show that it is an effective AM if the configuration error occurred in the cluster at cluster level. It is cognizable that the majority of configuration errors will occur somewhere in the cluster as this is where the machines, hosts and VMs are to be found and these are the key components of the infrastructure. Still, if the error occurs outside of the cluster say, at the datacenter level then cluster management becomes ineffective as an AM.

4.5 Scenario 4: Configuration Issues versus Node Management

This section presents the results for scenario 4. This scenario purposed to test whether node management is a suitable AM for configuration issues. In the first simulation a deliberate error in RAM setting of the nodes in the cluster was made and the simulation was allowed to run without any form of node management. In the subsequent simulations the value of RAM was adjusted upwards till the required setting of 16384 MB was achieved. There were 500 cloudlets that were sent to the datacenter in each simulation.

4.5.1 Simulations

There were three simulations performed. Each simulation, but one, allowed for a deliberate error in configuration settings for the node RAM setting.

Simulation 1 Observations (RAM = 163 MB)

In this simulation requests for VMs were not fulfilled, i.e. VM creation failed in all the hosts
Allocation of VM failed by RAM

Simulation 2 Observations(RAM = 512 MB)

In this simulation RAM setting for the hosts was set at 512 MB. After the simulation run it was observed that 10 VMs were created out of a request for creation of 30 VMs. Figure 4.13 shows the VMs that were created in the hosts. Each respective host created only one VM in it.

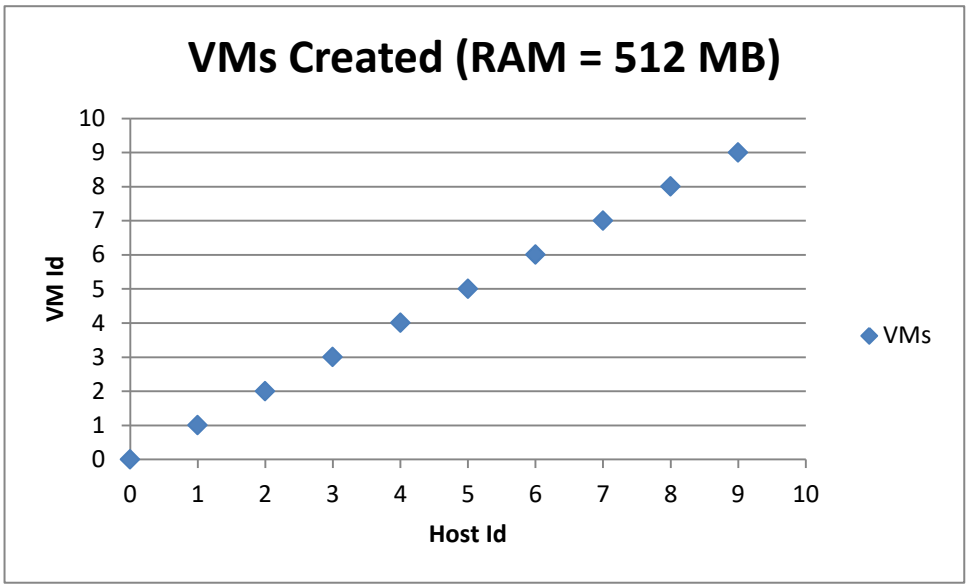

Figure 4.13 VMs created

In summary the output for this simulation was as follows:

- VMs created 10
- VMs failed: 20
- Cloudlets executed: ALL
- Execution time: 8000 seconds

Simulation 3 Observations (RAM = 16384 MB)

In this simulation the RAM setting was set at 16384 MB for the hosts. The results show that all 30 VMs were created and same number of cloudlets as for simulation two executed in a shorter time. The results were consistent with Figure 4.4, 4.6 and 4.7.

In summary the output for this simulation was as follows:

- VMs created: 30
- VMs failed: 0

- Cloudlets executed: ALL
- Execution time: 2560 seconds (quad cores) and 2720 seconds (dual cores)

4.5.2 Discussion

The FWA model suggests on the conjugate side that node management may also be used to proactively counter configuration issues as an outage cause. Simulations one and two show how configuration errors affect availability of the cloud:

In simulation 1 by erroneously setting RAM at 163 MB instead of 16384 MB at node level it was observed that no allocation of VMs occurred and thus there was no execution of tasks at all. This is due to the inability of the hosts to generate enough memory for the VMs to be created in the datacenter. User tasks are thus not executed and the broker is not able to send any of the tasks to the hosts. Table 4.3 shows the service availability and execution availability levels with the different node configuration settings.

Table 4.3 Node RAM configuration settings vs availability

Simulation Number	Node RAM configuration setting (MB)	VMs created	VMs failed	Service Availability (η)	Execution Availability (ε)
1	163	0	30	0	0
2	512	10	20	10/30 (33.33%)	500/500 (100%)
3	16384	30	0	30/30 (100%)	500/500 (100%)

As is observed in simulation 2 the service availability value at RAM of 512 MB is very low at 33.33% while the execution availability value is 1. When the correct RAM configuration of

16384 MB is set service availability rises to 100%; suffice to say that if the RAM parameters are set higher at the nodes then it would be expected that overall execution time would improve significantly. The observations and discussion of section 4.4 (configuration issues vs cluster management) apply here. It is desirable to prevent configuration errors from occurring in the first place. Node management may be used to counter configuration errors according to the model, but only at node level. The parameter settings at node level are restricted to the cores themselves and the hosts created in each core. This is a more abstract level of the infrastructure compared to the cluster level. In a typical datacenter configuration settings would typically be at three levels: the datacenter, the clusters and finally the nodes in the clusters. In section 4.4 it had been concluded that cluster management may counter configuration issues but only up to the cluster level. From the simulations above it is also be inferred that node management may be used to counter configuration issues, but only at node level. Further as nodes make up the cluster it is more desirable to use cluster management to proactively manage configuration errors as opposed to node management; it may thus be concluded that node management is not a viable alternative to cluster management in proactively countering configuration issues.

4.6 Scenario 5: Resource Exhaustion versus Limit Detection Policy

This section presents the results for scenario 5. This scenario purposed to test whether limit detection policy is a suitable AM for resource exhaustion. A total of six simulations were performed. The purpose of the simulations was to test the effect of different policies on cloud execution in the infrastructure.

4.6.1 Simulations

A total of six simulations were performed. Table 4.4 shows the results of the simulations. In simulation 1 and simulation 2 the space shared policy was used to send 500 cloudlets and 10000 cloudlets respectively for execution. The time to complete execution was then recorded. This process was repeated using time shared policy and time shared over subscription policy. It was observed that there wasn't a significant difference in time to complete execution regardless of the policy used.

Table 4.4 Limit detection policy

Simulation Number	**VM Scheduler Policy**	**Cloudlets Executed**	**Execution Time(seconds) – Time To Complete (TTC) (accuracy to 3 decimal places)**
1	Space Shared	500	2640.071
2	Space Shared	10000	53358.775
3	Time Shared	500	2640.071
4	Time Shared	10000	53358.776
5	Time Shared OverSubscription	500	2640.071
6	Time Shared OverSubscription	10000	53358.775

4.6.2 Discussion

The policies described in table 4.4 are VM Scheduler policies. The main resources when it comes to the possibility of resource exhaustion are at CPU and RAM level as far as physical dependencies are concerned. The available CPU cores and their load are managed by the VM scheduler while for RAM it is managed by the RAM provisioner, which for the purposes of this study was the RAMProvisionerSimple. The RAMProvisionerSimple policy simply allocates available RAM to the VM as long as the requested RAM doesn't exceed the RAM provided by the host. From the literature review in chapter two it was determined that resources include networks, servers, storage, applications, and services. However, the FWA model examines resource exhaustion from a general perspective, i.e. a resource exhaustion that would result in

performance degradation or no service at all. The purpose of the simulation was to determine whether the load balancing policies had a significant effect in proactively preventing resource exhaustion at the VM level. The results in the table indicate that the TTC was the same for the same number of cloudlets regardless of the policy used. In an experiment studying space shared and time shared policies Himani and Sidhu (2014) configured 1000 MIPS, available bandwidth of 10 Gbit/s, storage capacity of 1 TB and a Random Access Memory (RAM) of 2048 MB. They performed simulations to compare the performance of space shared and time shared policies and concluded that space shared outperforms time shared, though very minimally. Unfortunately they did not state the number of VMs used or the number of hosts used in their experiment. Their focus was on the TTC of the tasks using the different policies; further they did not state the size of the tasks and this would definitely have played a role in the TTC of the tasks using the different policies. The CloudSim model assumes independence of all resources. In practical sense it would be expected for example that when the CPU is getting overloaded then it would call for more RAM to enable it to complete tasks and this would mean there is some interdependence between the two. In CloudSim monitoring such progress is not possible, i.e. observing the CPU calling for more RAM, as is also the case in reality. In reality observing such progress is internal machine functionality. Taddei (2015) developed a Resource Dependency Aware (RDA) module that compared allocation policies to resources used with a view to determining the best allocation policies to use under different circumstances. The module would then determine the best allocation policy to use and thus prevent resource exhaustion. This supports the FWA model in suggesting that resource exhaustion can be countered using these policies. In the simulation results captured in table 4.4 it may be concluded that the different allocation policies under the given circumstances only produced an equal TTC. However, the whole purpose of the allocation policies is to increase efficiency of task execution and prevent system failure by balancing the task load in different ways.

The focus of this study was not to study how the allocation policies behave under different loads but to determine whether using different policies with the same load would prevent or allow resource exhaustion. The table shows that the tasks are all executed within the same TTC regardless of the policy. Taddei (2015) alluded to the role of the allocation policies in distribution of resources and thus developed the RDA module to help in determining the best

allocation policy to use in a given scenario. The work of Hamani and Sidhu (2014) is more focused on the task profit and execution using the different policies, and they observed a nominal improvement using space shared as opposed to time shared; however, they did not test the time shared over subscription policy.

4.7 Scenario 6: Security Issues Vs Checkpointing

This section presents the results for scenario 6. This scenario purposed to test whether checkpointing is a suitable AM for security issues. Two simulations were performed using FTCloudSim. The purpose of the simulations was to test the effect of checkpointing on cloudlet execution in the infrastructure when random failures are injected.

4.7.1 Simulations

There were two simulations performed. The first simulation injected random failure while the second one made use of checkpoints in order to attempt to recover cloudlets.

Simulation 1 Observations (Random Failure)

In this simulation cloudlets were failed randomly at the datacenter. The simulation showed that out of 50 cloudlets requested, 18 cloudlets were failed while 32 cloudlets were successfully executed. Figure 4.14 shows the 18 cloudlets that were randomly failed during the simulation.

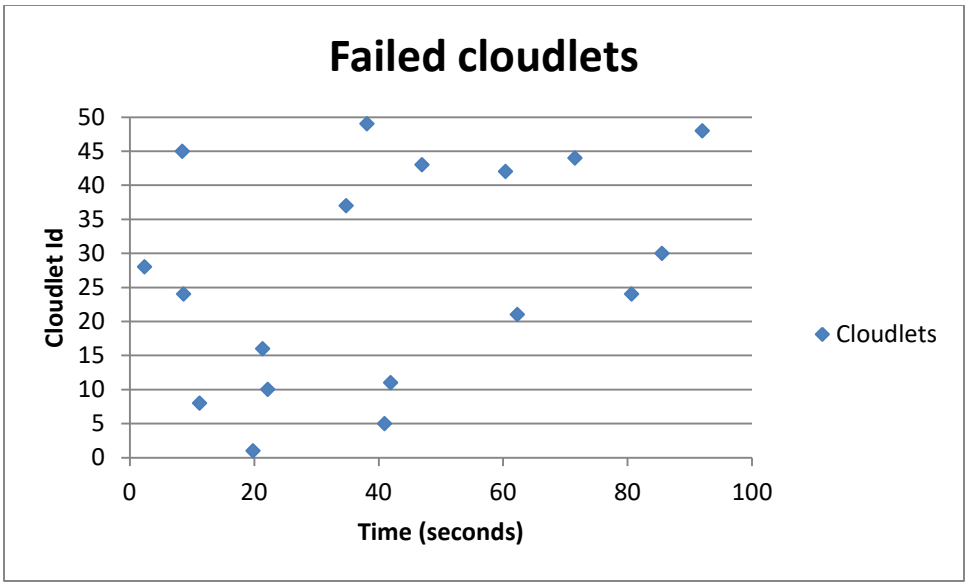

Figure 4.14 Failed cloudlets

The simulation can be summarized as follows:

- Number of Failed Cloudlets: 18
- Successful Cloudlets: 32

Simulation 2 Observations (Checkpoint Invocation)

In this simulation checkpoints were set at 5 seconds apart. Figure 4.15 shows the checkpoint Ids against time. A total of 80 checkpoints were invoked in time of 400 seconds.

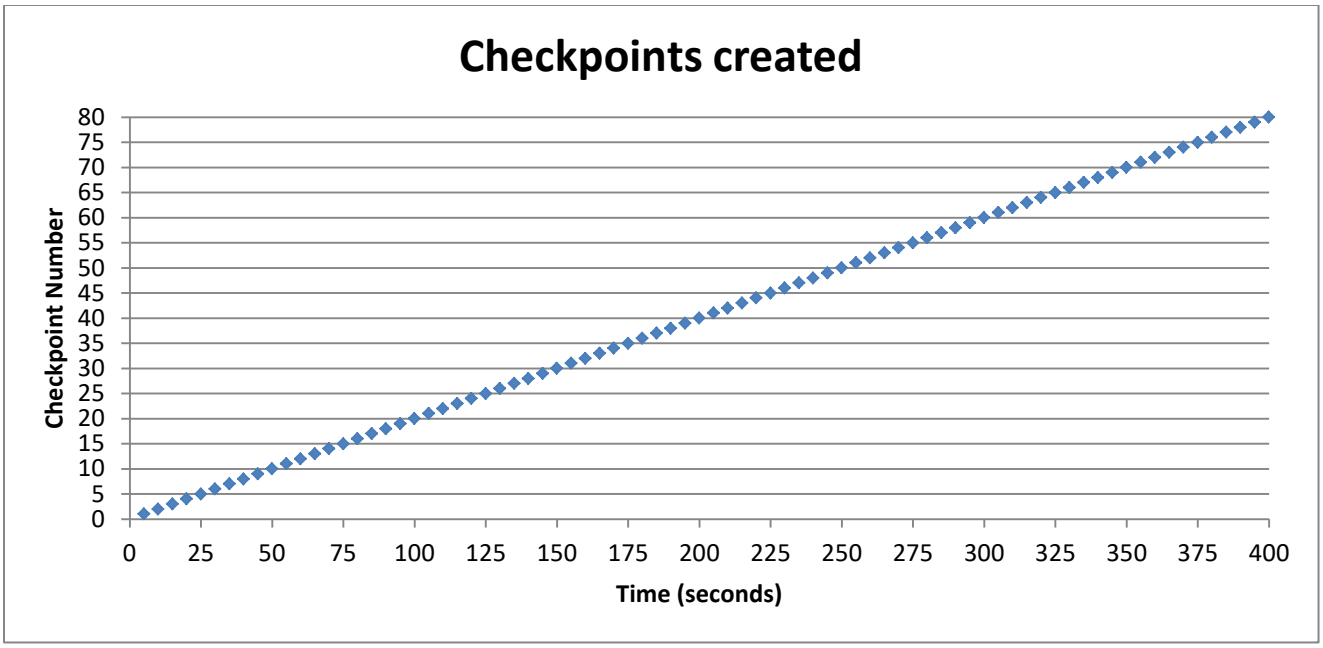

Figure 4.15 Checkpoint invocation

In this simulation 15 cloudlets randomly failed. Table 4.5 shows the time to failure (TTF) of the 15 cloudlets. It was observed that 14 out of the 15 cloudlets were recovered by the use of checkpointing. One cloudlet was not recovered.

Table 4.5 Cloudlet time to failure (TTF)

Cloudlet Id	Time To Failure (seconds)
1	323.142
4	240.901
8	93.533
9	92.886
16	277.364
20	118.863
26	321.506
27	255.977
30	318.149
31	287.057
37	305.441
42	37.928
47	81.021
48	4.895
49	379.321

Simulation 2 can be summarized as follows:

- Checkpoint Interval: 5 seconds
- Failed Cloudlets: 15
- Recovered Cloudlets: 14

4.7.2 Discussion

In simulation 1 above it was observed that 18 cloudlets failed out the 50 cloudlets sent to the datacenter. In terms of availability this equates to:

$\eta = 100\%$ (all resources requested were allocated)

$\mathcal{E} = 32/50 = 0.64 \Rightarrow 64\%$

In simulation 2 a checkpoint of 5 seconds was introduced in line with the recommendations of Singh et al. (2012) and the simulation ran again. The result was that 14 out of the 15 failed cloudlets were recovered, indicating a recovery success rate of 93.3%. Figure 4.14 shows the failure of clouds was done randomly to simulate a normal environment as much as possible. Checkpointing as measure of fault tolerance in cloud computing has been described in section 2.5.3. The principle is that using checkpoints the state of a failed cloudlet can be recovered. The challenges of checkpointing lie in overheads and most importantly the checkpoint interval. In simulation 2 cloudlet Id 48 could not be recovered since it failed before the first checkpoint, therefore, shorter checkpoint intervals are desirable. Further, shorter checkpoint intervals introduce more load in the database as more jobs are executed. In designing a checkpoint based recovery system the service provider must take into account the intervals as well as latency between the times it takes to recover the last state of the failed task from the checkpoint server which hosts the database. The system should have a mechanism to remove the completed tasks to backup for example, so that the database is free to work with incumbent tasks. This can be captured thus:

Let there be N cloudlets to execute with cloudlet Ids from 0 to N

Cloudlets = {0, 1, 2,……N};

Let the checkpoint intervals be denoted by X such that the checkpoint intervals are X_1, X_2……X_t

Checkpoint Interval (σ) = {X_1, X_2……X_t} where t is the time when the last checkpoint was inserted;

Let the checkpoint Id be denoted by ϱ such that it corresponds with the checkpoint interval σ i.e. ϱ_n corresponds to σ_n

Let the state of the cloudlets captured at the different intervals be denoted by A, B,….Z

Cloudlet state (α) = {A_p, B_p, C_p,…..Z_p, $A_{np,}$ $B_{np,}$ …..Z_{np}} where Z_n (or Z_{np}) is the last state when the task is completed; p representing the cloudlet Id number, and n representing an integer value should there be more than 26 states of the cloudlet captured.

Table 4.6 shows the different states in time for a cloudlet with Id 1.

Table 4.6 Cloudlet Id 0 storage of time and state

Time	Checkpoint Id	State
X_1	ϱ_1	A_0
X_2	ϱ_2	B_0
X_3	ϱ_3	C_0
X_4	ϱ_4	D_0

Table 4.7 shows the different states in time for a second cloudlet, say cloudlet Id 1.

Table 4.7 Cloudlet Id 1 storage of time and state

Time	Checkpoint Id	State
X_1	ϱ_1	A_1
X_2	ϱ_2	B_1
X_3	ϱ_3	C_1
X_4	ϱ_4	D_1

These states will be captured in the database in the form above. After a designated period of time T an incremental backup is performed removing all cloudlets and their states so the database is free. This makes it work faster in retrieving states as required.

As described by Cao et al (2014) checkpointing can even be used by clients as a service in itself in higher layers of the cloud regardless of the type of cloud being used. It can therefore be inferred from the simulations above that checkpointing can be effectively used to counter security issues related to data loss in the cloud. However, further experiments should be done to identify the ideal checkpointing interval under different circumstances. This should be based on recovery of cloudlets, latency and the overheads described above. In this particular scenario service availability = 100%, execution availability = 93.3%. Checkpointing is therefore a suitable AM for countering security issues.

4.8 Scenario 7: Resource Exhaustion versus Checkpointing

This section discusses scenario 7. This scenario purposed to test whether checkpointing is a suitable AM for resource exhaustion. There were no simulations done to test this scenario. This is discussed further hereunder.

Resource exhaustion occurs when the resources required to execute an action are entirely expended, preventing that action from occurring. In the context of the FWA model checkpointing as a countermeasure for resource exhaustion was not practically possible. This is because at infrastructure level the state of a task can be recovered using checkpointing but not the state of a resource. In the literature review Myerson (2013) noted *inter alia*, that resource optimization failure, threshold implementation policy failure and hypervisor failure as being the causes of resource exhaustion. Each of these can be prevented by implementing a limit detection policy as depicted in section 4.6. Capturing the state of a resource is the main challenge as the state of a resource can only be captured as being on or off with the current experimentation tools; indeed even RDA module developed by Taddei (2015) could only capture dependencies between resources in order to balance them but could not prevent exhaustion were it to occur. Theoretically thus checkpointing would not be suitable in proactively preventing resource exhaustion but perhaps further research into this area in terms of looking for parameters to measure resource exhaustion may be undertaken.

4.9 Scenario 8: Security Issues versus Limit Detection Policy

This section discusses scenario 8. This scenario purposed to test whether limit detection policy is a suitable AM for checkpointing. There were no simulations done to test this scenario. This is discussed further hereunder.

As described in section 4.6 a threshold policy would help in ensuring that workloads are balanced dynamically and thus resource exhaustion does not occur. From an infrastructure perspective this would mean resources at the infrastructure level. The fathomable security issue that would bring this about would be a denial of service (DoS) attack. This is because it would be expected that in the datacenter measures against attacks by viruses and malware related threats would be prevented using suitable antivirus solutions. The simulator used for this study CloudSim did not have provision for simulating DoS attacks. A study of the functionalities of other network simulators available online such as Cloonix, MaxiNet, CORE, Mininet, GNS3 and Imunes indicate that these are able to simulate a network or a Distributed DoS but do not have the provision to implement a limit detection policy as this would not be applicable in the context

of this study. Practically a limit detection policy would not do anything to prevent a DoS or a DDoS attack for that matter as it is focused on resource usage. Thus it cannot prevent the security issue at hand. It is therefore be concluded that the conjugate side of the model, namely limit detection policies to prevent security issues does not apply; perhaps a security policy would be more appropriate in these circumstances.

4.10 Scenario 9: Hardware Issues versus Component Redundancy

This section discusses scenario 9. This scenario purposed to test whether component redundancy is a suitable AM for checkpointing.

4.10.1 Simulations

The setup for this simulation was described in section 3.4.9. The setup is similar to the simulations of section 4.2.1. These were the simulations that investigated whether node management was a suitable AM for node failure. It was found that the results were similar to figure 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7.

4.10.2 Discussion

In a study on hardware failures at datacenter level, Vishwanath and Nagappan (2010) observed that 70% of all server failures are due to hard disks, 6% due to RAID controller and 5% due to memory and the rest (19%) due to other factors. It is apparent that based on these findings 81% of the failures are server related. As described in the section 3.4.9 of the methodology the simulation of a hardware failure would cascade to the hosts in it. This would in turn ideally result in a failure of the VMs and the tasks they were processing. Introducing component redundancy would mean having a redundant server that would take over the tasks should the incumbent fail. The configuration therefore as far as the simulation is concerned would be a replica of simulation 2 of scenario 1, and as is characteristic of a deterministic simulation the results would be the same. In that simulation it was found that execution availability and service availability were both 100% after the simulation run. It is concluded then that component redundancy does proactively counter hardware issues as an AM but with a caveat. The literature found that only 81% of the hardware failures were server related. This implies that the findings

of the simulation are at best 81% conclusive. The simulator was not capable of simulating the remaining 20% and this will have to be researched further.

4.11 Scenario 10: Network Issues versus Active-X Variant

This section presents the results for scenario 10. This scenario purposed to determine whether Active-X variant is a suitable AM for network issues. Two simulations were performed with the purpose of simulating a failure first, then migrating the VMs to a separate datacenter.

4.11.1 Simulations

There were two simulations performed. In the first simulation two datacenters were created with a total of 20 VMs between them. Each datacenter had two machines configured with a single host one each one.

Simulation 1 Observations (node failure)

In this simulation a random number generator was used to fail the hosts belonging to datacenter 0. This led to the two hosts residing in datacenter 0 (DC0) failing at the 288 second and 488 second mark respectively. Figure 4.16 shows the two failed hosts and the time they were failed.

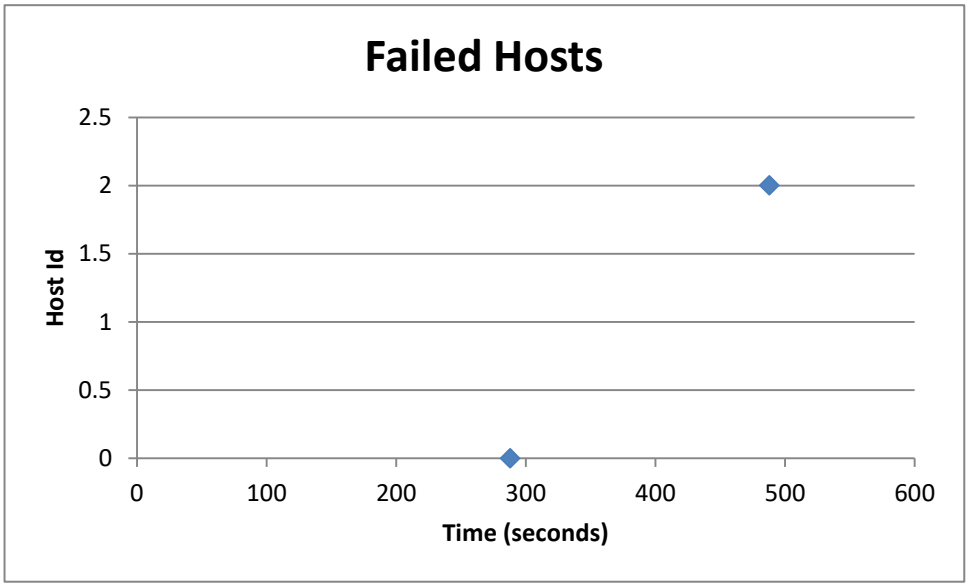

Figure 4.16 Failed hosts

The simulation can be summarized as follows:

- Hosts Failed: 2
- Cloudlets Requested: 100
- Cloudlets executed: 0

Simulation 2 Observations (Active-X Variant)

In this simulation the hosts in DC0 were failed again. However, a policy was put in place to migrate the VMs in the hosts to DC1. Figure 4.17 shows the two hosts together with their respective VMs in DC0.

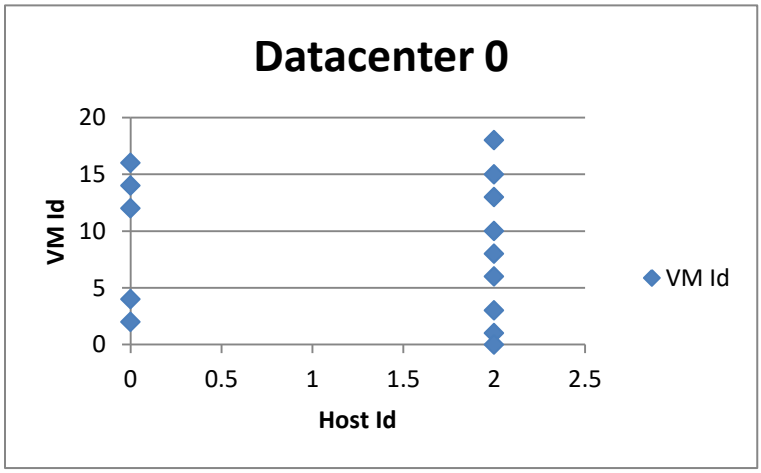

Figure 4.17 Hosts and VMs in DC0

Figure 4.18 shows the hosts and VMs created in DC1.

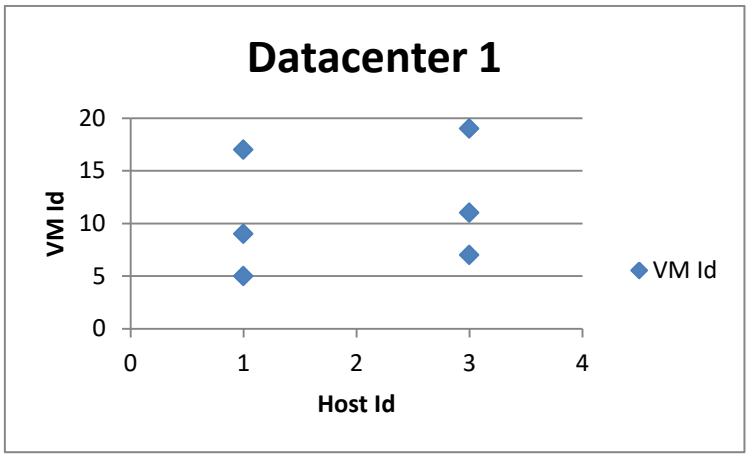

Figure 4.18 Hosts and VMs in DC1

Hosts Failed: 2

Cloudlets Requested: 100

Cloudlets Executed: 100

TTC: 800.2 seconds

4.11.2 Discussion

The active –x variant is inspired by the heartbeat program designed for high availability in Linux environments. As described in the literature the active –x variant works in such a way that, in an active-passive environment when one server fails (the active one) then the other server (passive one) immediately takes over and thus continuity is assured. In an active-active environment both servers are active and share workloads from users based on some load balancing algorithm; if one server fails then the remaining one takes over the full workload. The FWA model envisions the use of this environment to counter network issues between the server and the broker and in turn the user. In the simulations performed in this scenario two datacenters were configured with two users requesting a total of 100 tasks. Fourteen VMs were created in DC0 while six were created in DC1. Since the configuration consisted of 4 hosts for the purpose of the simulation two were failed in one of the datacenters during the simulations. The first simulation enacted node failure in the datacenter similarly to simulation one of scenario 1. No VM migration was

allowed and as expected there was no cloudlet execution. In simulation two the TTC was 800.2 seconds. Host 0 failed first at the 288 second mark followed by host 1 at the 488 second mark. According to policy the VMs in host 0 were migrated first to DC1; this would then assign the VMs to the first available host in this case host id 1. At the 488 second mark when host 1 failed its VMs were also migrated to DC1 which then assigned the tasks to the next free host in this case host id 3. The final TTC was the same for all cloudlets (800.2 seconds). An analysis of execution availability and service availability reveal the following:

$\eta = R_A / R_R = 20/20 = 100\%$ (up to the 288 second mark)
Resource in this instance is the VMs. 20 VMs were requested and 20 VMs were allocated.

At the 288 second mark,
$\eta = 15/20 = 75\%$ (but only for a fraction of time);
This is because 5 VMs failed at this point leaving only 15 VMs functioning.

At the 488s mark,
$\eta = 11/20 = 55\%$ (since the 5 previously hanging VMs have already been migrated to DC1);
This is because the 5 VMs having been migrated to DC1 now make up a total of 11 available VMs (6 VMs that were already present in DC1 plus the 5 migrated VMs).

And by the end of the simulation (800.2 second mark)
$\eta = 20/20 = 100\%$

$\mathcal{E} = \mu / \lambda = 100/100 = 1 \Rightarrow 100\%$
These results are captured in table 4.8

Table 4.8 Service availability at inter-datacenter level

S/No	Time (seconds)	Service Availability (η) (%)
1	< 288	100
2	288	75
3	488	55
4	800.2	100

In this simulation an active-passive variant was used as opposed to an active-active configuration since when configuring more than one core in the datacenter the default policy for migration is an active-active configuration. From the simulation results and discussion it is inferred that Active-X variant is a suitable AM for the network issue described above.

4.12 Scenario 11: Network Issues versus Component Redundancy

This section presents the results for scenario 11. This scenario purposed to determine whether component redundancy is a suitable AM for network issues. Two simulations were performed with the purpose of simulating a failure first, then migrating the VMs to a separate datacenter using redundancy at datacenter level. The configuration setup was similar to section 4.11, the only difference being in an additional simulation described in section 4.12.1

4.12.1 Simulations

There were two simulations performed. In the first simulation two datacenters were created with a total of 20 VMs between them. Each datacenter had two machines configured with a single host one each one.

Simulation 1 Observations (node failure)

The simulation results were consistent with section 4.11.1 when the nodes at one datacenter were failed, i.e.

- Hosts Failed: 2
- Cloudlets Requested: 100

- Cloudlets executed: 0

Simulation 2 Observations (Component redundancy)

In this simulation the same procedure as in section 4.11.1 was followed. The difference here was the policy put in place to migrate the VMs in the hosts to DC1. This policy followed a load balancing script as described in section 4.12.2. When 100 tasks were sent to DC0 the results were the same as in section 4.11.1, i.e.

- Hosts Failed: 2
- Cloudlets Requested: 100
- Cloudlets Executed: 100
- TTC: 800.2 seconds

Further when the cloudlets were increased to 200 the following was observed:

- Hosts Failed: 2
- Cloudlets Requested: 200
- Cloudlets Executed: 200
- TTC: 1600.2 seconds

4.12.2 Discussion

Component redundancy is a scenario where a resource has a redundant identical resource available to take over its functions should it fail. The only question here is how the redundancy/switch over is achieved. In both OSCAR and heartbeat the configurations are ideally active-passive, i.e. the redundancy of components is such that the redundant component sits idle until the active one fails. The second type of redundancy is where the two devices both share workloads and when one device fails the remaining one takes over the full workloads till the failed one is repaired or replaced. CloudSim implements the latter type of redundancy by default, i.e. sharing of workloads across nodes; when one node goes down enabling VM

migration allows the tasks in it to be migrated to other nodes according to the configured policy. VM migration is available for configuration but is not there by default and users have to configure it at the level they desire.

In section 4.11 an active-passive environment was configured between DC0 and DC1, with the latter being passive (no tasks sent to it) until the hosts in the former failed. The results of the simulation were discussed in that scenario. Using the same configuration this scenario was reenacted only this time workloads were shared between DC0 and DC1 and VM migration enabled in the same manner. The results were exactly the same in terms of TTC, and thus values of both service availability and execution availability also same; doubling the tasks to 200 resulted in a TTC of 1600.2 seconds, roughly double the time it took to execute 100 tasks. Logically this result isn't surprising; this is because the processing was continuous from the beginning of the simulation to the end as in section 4.11. This would seem to mean sharing the workload did not reduce processing time as would have been expected; rather the processing time remained the same. The observation is that despite failing the two hosts (half the number) with workload shared across the two datacenters the tasks were still migrated and completed with the same execution availability and service availability. This leads to the conclusion that component redundancy is a suitable AM to deal with network issues.

4.13 Scenario 12: Hardware versus Active-X Variant

This scenario required to test whether Active-X variant is a suitable AM to hardware issues. It required simulating a hardware failure then using an Active-X variant proactively to counter the hardware failure.

4.13.1 Simulations

Simulation 1 (hardware failure)

This was found to have been enacted in scenario 1 (section 4.2) where node failure was configured and run. The simulation results were found to be the same.

Simulation 2 (Active – X variant)

Active-X variant – using either an active-active setup or an active-passive setup to proactively handle the failure. The following was found:

Active- passive was enacted in simulation 2 of section 4.11 (network issues vs Active – X variant) while active-active was described in simulation 2 of section 4.12 (network issues vs component redundancy). The results were consistent with those simulations.

4.13.2 Discussion

As has been described in section 4.12 the simulator uses an active-active environment by default; a simulation of a host failure within the datacenter and migration of VMs was explained in section 4.2. A similar simulation of a host failure within an active-passive environment was enacted in section 4.11. In section 4.10 an active-active environment for component redundancy was simulated, and thus the workload was shared across the hosts in the datacenter. However, examining section 4.11 leads to making an inference that active –X variant as an AM is a suitable approach to use against hardware failures (hardware failure at server level was earlier found to be 81% of the failures at the datacenter). The reasoning behind this is that at datacenter level it was found that DC1 suitably completed the tasks as a standby server for DC0; this was a networked environment. The simulator did not take into account any cross network parameters like latency and the like, as this is not what it has been configured to study; such simulators are like the ones described earlier in section 4.9. Therefore assuming that issues like latency would only slow the results nominally then section 4.11 would ideally be a replica of this scenario, and thus it is justified to draw this conclusion.

4.14 Summary of the simulations

The simulations results discussed in this chapter enable a summary of the findings in table 4.4. The table shows that only 2 out of the 12 mappings were inconclusive or not applicable. This implies a success rate in improving service availability of 83.3% of the FWA model.

Table 4.9 Summary of findings

S/No	Outage Cause	Availability Mechanism (AM)	Result	Conjugate	Result
1	Node Failure	Node Management	Effective ($\eta = 100\%$)	Cluster Management	Effective ($\eta = 100\%$)
2	Configuration Issues	Cluster Management	($\eta = 100\%$) if issue is at cluster level	Node Management	($\eta = 100\%$) if issue is at node level (this level is very abstract; better to use cluster management)
3	Resource Exhaustion	Limit detection policy (this AM should be renamed limit prevention policy)	Effective ($\eta = 100\%$)	Checkpointing	Not applicable as an AM for resource exhaustion
4	Hardware Issues	Component Redundancy	Effective ($\eta = 81\%$ at most since 19% of the hardware could not be simulated)	Active-X Variant	Effective for active-active as well as for active-passive ($\eta = 81\%$ at most since 19% of the hardware could not be simulated)
5	Network Issues	Active-X Variant	Effective ($\eta = 100\%$)	Component Redundancy	Effective ($\eta = 100\%$)
6	Security Issues	Checkpointing	Effective ($\eta = 100\%$)	Limit Detection Policy	Inconclusive

It is important to note that the service availability figures that are shown in table 4.9 are based on the simulations performed. Where applicable the execution availability was also computed. In section 2.4 of the thesis where availability was defined, it was pointed out that CSPs are required to commit to a certain percentage of availability in the SLA. From the simulations it has been shown that the service availability is 100% in 83.3% of the model, and theoretically this would mean that the CSP should commit to the five 9s of availability. However, this still needs to be tested in a live environment. From table 4.9 the AM for node failure was node management and on the conjugate side cluster management. The simulations showed that both AMs are effective in enhancing availability; further the study suggested that effective management of the cluster results in effective management of the node as well. The AM for configuration issues was cluster management with node management being the conjugate AM. The simulations showed that cluster management is effective if the issue is at cluster level; node management is effective if the configuration issue is at node level. The study suggests that node management is abstract as it applies to the node only, and therefore cluster management would be more desirable in dealing with configuration issues. The AM for resource exhaustion was limit detection policy with checkpointing being the conjugate AM. The findings showed that limit detection policy was effective as an AM; however, it is suggested that limit detection policy be renamed limit prevention policy in the model. Checkpointing was not an effective AM for resource exhaustion. The AM for hardware issues was component redundancy with Active-X variant being the conjugate AM. The findings showed that component redundancy was effective though not absolutely (maximum of 81%). Active-X variant as an AM proved effective (also not absolutely due to the 19% of hardware that could not be simulated). The AM for network issues was Active-X variant with component redundancy being the conjugate AM. The findings showed that both AMs were effective in dealing with this outage cause. Checkpointing was the AM for security issues with limit detection policy being the conjugate AM. The findings showed that checkpointing was effective as an AM while the results were inconclusive for limit detection policy.

This chapter purposed to address the fifth specific objective of this study namely:

5. Evaluate the effectiveness of the model by measuring its service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters.

   The chapter discussed the service availability levels of all the simulations and went further to introduce a new availability measure called execution availability. This measure examines the tasks requested by the broker versus the tasks actually executed. Execution availability taken together with service availability is a better measure to use in determining overall availability of the infrastructure. The reasoning behind not using service availability on its own is that the simulation results showed more than once that the service may be available as per the definition of service availability but it is not able to execute tasks due to other factors. From a user perspective the only indication that the service is available is if and when they get the results of their requested tasks, making execution availability a viable measure to use together with service availability. If at the end of the simulation the execution availability is less than one (or less than 100%) it means that there were tasks that weren't completed, and this may slow the whole infrastructure leading to doubts as to its level of availability. Further execution availability ratios lead infrastructure engineers to look beyond the resources to determine what is causing tasks not to be fully executed.

CHAPTER FIVE

CONCLUSIONS AND RECOMMENDATIONS

5.1 Introduction

The main objective of this study was to develop and evaluate a quantitative model that relates service availability mechanisms and service outage causes in cloud computing environments. Towards the achievement of this the study sought to fulfill the following specific objectives, (1) identify the causes of outages in cloud computing infrastructures, (2) identify AMs in use in cloud computing infrastructures, (3) formulate a model that establishes relationships between AMs and outage causes, (4) test the functionality of the model, and (5) evaluate the effectiveness of the model by measuring its service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters. This chapter summarizes the study findings, concludes, discusses the contribution, and makes recommendations and suggestions for further research. Towards the achievement of this, the next section offers a summary of the research findings.

5.2 Summary

This section is segmented based on the study objectives. This enables a more straight forward understanding of each objective and the accompanying findings.

5.2.1 Identify the causes of outages in cloud computing infrastructures.

These were identified as hardware issues, configuration issues, security issues, natural disasters, network issues, node failures, and resource exhaustion

5.2.2 Identify Availability Mechanisms in use in cloud computing infrastructures

These were identified as component redundancy, cluster management, checkpointing, fault tolerance, Active-X variant, node management, and limit detection policy

5.2.3 Formulate a model that establishes correspondences between AMs and outage causes

The model was formulated based on the literature review, and an analysis of the relationship between AMs and outage causes. The relationship between AMs and outage causes was also explained and demonstrated mathematically in chapter 2.

The model aimed to correlate various causes of outages to known availability mechanisms in order to increase availability. The Ferris Wheel of Availability (FWA) model presents classification of availability mechanisms and outage causes in the following manner:

- Availability mechanisms (AMs) are grouped into seven broad categories namely cluster management, component redundancy, checkpointing, limit detection policy, node management, Active-X variant and fault tolerance

- Outage causes are also grouped into seven broad categories namely configuration issues, hardware issues, resource exhaustion, security issues, node failures, network issues and natural disasters.

  Figure 5.1 shows the FWA model, capturing the relationships between the AMs and the outage causes:

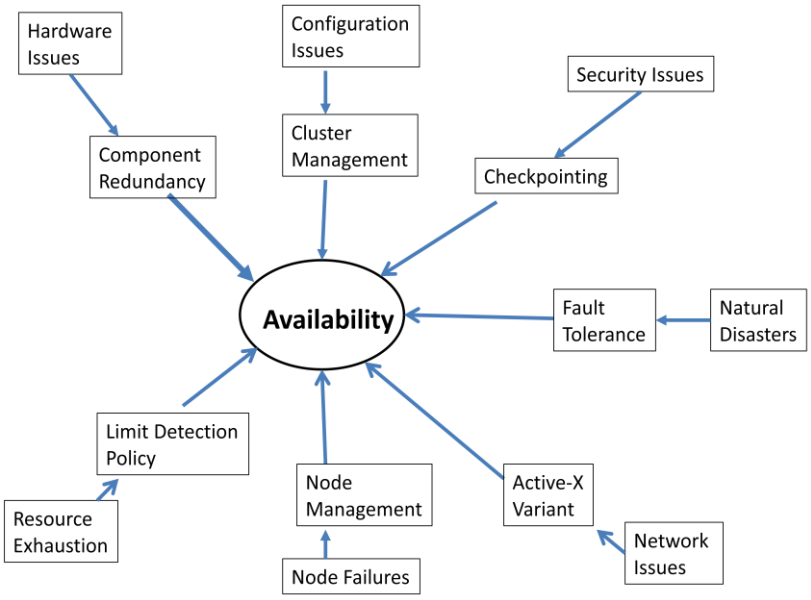

Figure 5.1 Ferris wheel of availability (FWA) model

For each group of outages there is a corresponding AM in place to counter the outage. The AMs aim to make the model proactive in that by putting the respective AM in place then this should logically increase availability for the service provider. The model uses the imagery of a Ferris

wheel since it was proposed that there would be a correlation between direct opposite outages and corresponding AMs, for example node failures and configuration issues are related, network issues and hardware issues are related and resource exhaustion and security issues are related. Natural disasters are the odd outage with no apparent correlation. Consequently some of the correlated outage groupings could use/swop corresponding AMs as opposite AMs on the wheel are conjugate in nature. The Ferris wheel seats are designed in such a way that when the seat reaches its lowest points it maintains its perpendicular position thus making it stable for its occupants all around a 360 degree turn; thus the model aims for stability in controlling outages proactively in the cloud environment. The merits, challenges and scope of the model were also discussed.

5.2.4 Evaluation of the effectiveness of the model by measuring service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters (Testing the model)

The first part of evaluating the model was to test it. CloudSim toolkit was used to test the functionality of the model. CloudSim is a deterministic simulator that is widely used in research of various cloud computing scenarios for example VM management (Monil and Rahman, 2016), modeling and evaluating multi-resource dependencies (Taddei, 2015) and performance analysis of heterogenous data centers (Bai et al., 2015). It is used mostly since using a real datacenter is not practical and would be too expensive. Dobre et al. observed that cloud simulation top benefits include flexibility, easy customization and lower costs (cited by Nita et al, 2014). Users are able to build their own scenarios using the toolkit and run them in order to see the results of their applications. The functionality of the model was tested using different configuration scenarios, with the general algorithm being explained. Each scenario in the model was configured using the theory discussed behind the development of the model. The configurations for each scenario were tabulated and explanations provided using either figures or tables, or a combination of both. All the configured scenarios tested whether applying the specific AM to the outage cause would result in the desired outcome, i.e. the system remaining available despite the outage cause being effected. The scenarios also tested whether swapping the AMs with their conjugates also resulted in availability or not. Twelve scenarios were described and investigated: node failure versus node management, node failure versus cluster management, configuration

issues versus cluster management, configuration issues versus node management, resource exhaustion versus limit detection policy, security issues versus checkpointing, resource exhaustion versus checkpointing, security issues versus limit detection policy, hardware issues versus component redundancy, network issues versus Active-X variant, network issues versus component redundancy and hardware versus Active-X variant

5.2.5 Evaluation of the performance of the model by measuring service availability levels in cloud computing environments in relation to the settings of the cloud computing system parameters (Measuring performance)

The study discussed the service availability levels of all the simulations and went further to introduce a new availability measure called execution availability. This measure examined the tasks requested by the broker versus the tasks actually executed. Execution availability taken together with service availability is a better measure to use in determining overall availability of the infrastructure. The reasoning behind not using service availability on its own is that the simulation results had shown more than once that the service may have been available as per the definition of service availability but was not able to execute tasks due to other factors. From a user perspective the only indication that the service is available is if and when they get the results of their requested tasks, making execution availability a viable measure to use. Further, availability to the user is in terms of how much they can get done in a given period of time; if they send several tasks to the datacenter and they get the impression that the system is hanging due to its slowness then it implies less tasks done and hence less availability from their perspective. The effectiveness of the model using service availability as a measure was computed in chapter 4 of this study and the findings were as follows:

1. The AM for node failure was node management and on the conjugate side cluster management. The simulations showed that both AMs are effective in enhancing availability; further the study suggested that effective management of the cluster results in effective management of the node as well.
2. The AM for configuration issues was cluster management with node management being the conjugate AM. The simulations showed that cluster management is effective if the issue is at cluster level; node management is effective if the configuration issue is at node

level. The study suggests that node management is abstract and therefore cluster management would be more desirable in dealing with configuration issues.

3. The AM for resource exhaustion is limit detection policy with checkpointing being the conjugate AM. The findings show that limit detection policy is effective as an AM; however, it is suggested that limit detection policy be renamed limit prevention policy in the model. Checkpointing is not an effective AM for resource exhaustion.
4. The AM for hardware issues is component redundancy with Active-X variant being the conjugate AM. The findings show that component redundancy is effective though not absolutely. Active-X variant as an AM proved effective (also not absolutely) for both active-active configuration and active-passive configuration.
5. The AM for network issues was Active-X variant with component redundancy being the conjugate AM. The findings show that both AMs are effective in dealing with this outage cause.
6. The AM for security issues was checkpointing with limit detection policy being the conjugate AM. The findings show that checkpointing is effective as an AM while the results were inconclusive for limit detection policy.

5.3 Conclusion

The results that have been discussed in chapter four show that ten out of the twelve pairings in the FWA model returned positive results. This implies a success rate of 83% of the model. The remaining two pairings were inconclusive and not applicable respectively. This whole study purposed to examine whether the FWA model does indeed increase availability at infrastructure level, using service availability as the parameter that measured performance. The study also described the importance of using execution availability together with service availability in determining how available a cloud computing infrastructure truly is. For a user of the cloud service availability is when they are able to get results of their requests in a timely manner, hence the need to measure performance using service availability and execution availability. From the results it can therefore be concluded that the FWA model indeed does increase availability at infrastructure level and CSPs can adopt this model in developing their infrastructure.

5.4 Contributions of the thesis

The contributions of this thesis can be divided into four categories: classification and analysis of AMs and outage causes in the availability area, evaluation of AMs against outage causes, introduction of execution availability as availability measure, and introduction of a model for establishing relationships between AMs and outage causes. The key contributions are:

1. Development of the FWA model that establishes correspondences between AMs and outage causes. A model that establishes these correspondences had not been developed before. Discovery of the relationships between AMs and outage causes based on simulation tests and consequent analysis was also published.

2. Development of methodology for evaluating AMs against outage causes using the CloudSim toolkit. In the absence of the FWA model this methodology had not been developed before.

3. Introduction of an availability parameter called execution availability that measures the ratio of tasks executed versus tasks requested.

5.5 Recommendations

There are some recommendations based on the summary, discussions and conclusions of this thesis:

1. **To researchers and scientists:** those who are looking to investigate in greater details the study of availability at infrastructure level may be required to use more than one simulation tool. It is recommended to study the feasibility of merging two or more simulators to achieve results which were inconclusive using one simulator; alternatively exploration of an extension to the simulator in use may also be viable. In the duration of this study there was no existing extension to CloudSim that could assist in obtaining conclusive inferences to the areas that remained inconclusive.

2. **To CSPs and all third parties involved in the development of cloud infrastructure:** the use of the FWA model at CSP level is also recommended as it assists analysts and

developers to build for availability from the very foundation as opposed to adapting a wait-and-see attitude in countering outages as they occur. With AMs in place they can focus more on service delivery and improving on other aspects of cloud computing management. CSPs should also implement execution availability as part of their SLAs with their customers to improve on efficiency as it is measure that focuses on actual performance of the infrastructure in terms of measuring cloudlet execution. The reluctance of CSPs in sharing downtime data for research purposes was observed and documented. This has been an impediment in studying outage causes and thus build for availability and it is recommended that perhaps a laboratory such as the CLOUDS laboratory could implement a datacenter with minimal requirements and users that can capture this elusive data. The existing workloads on PlanetLab in CloudSim provide traces of CPU and VM utilization which can be used in development and testing of algorithms in the area but do not help in studying availability as a subject.

5.6 Suggestions for future research

Despite the contributions of this thesis in enhancing availability at infrastructure level, there are still some open research challenges and incomplete and/or inconclusive parts of the model that require further investigation. These include, but are not limited to:

1 Study of relationships between outage causes themselves.

The model should also eventually examine if there is any causal (direct or indirect) relationship between the outage causes themselves

2 Study of how to develop limit detection policy

This could not be investigated as trying to exhaust a resource resulted in the simulator hanging, making the assertion that the resource was exhausted inconclusive; thus only prevention of resource exhaustion could be investigated. A study may be undertaken to investigate factors that cause resource exhaustion and more specifically at what point does a resource get exhausted; and develop an algorithm or policy that can detect that limit and determine what action to take from there.

3 Study of limit detection policy on security issues

This also could not be investigated using existing tools; however, with a limit detection policy in place that would detect a DDoS attack by monitoring resources an investigation could yield some new knowledge in this area.

4 Combining multiple AMs in an Infrastructure

Whereas a CSP can implement all the AMs in the FWA model can the combination of say two AMs ward off multiple outage causes in the infrastructure?

5 Study of relationships between AMs

This thesis showed that effective cluster management results in effective node management as far as node failures are concerned. Are there other related AMs and if so what would be their effect on overall availability of the infrastructure?

6 Study of node failure to increase service availability

A study of the different factors that cause node failure in a datacenter and what proactive approach to use in preventing these failures

7 Study of ideal checkpointing interval in cloud computing environments

A study that will identify the ideal checkpointing interval based on recovery of cloudlets, latency and the overheads in a cloud infrastructure.

8 Study into measurement of resource exhaustion

A study that will identify or that looking for parameters to measure resource exhaustion may be undertaken. With the current literature the state of a resource is either off or on; is there parameters/measures that can be used to measure a resource between the two states?

9 Application of availability parameters in cloud computing environments

This study will consider all the different types of availability defined by different authors, including the one introduced in this study (execution availability), and attempt to categorize them to the different aspects under investigation in the cloud.

REFERENCES

Availability. Dictionary.com. Retrieved from https://www.dictionary.com/

Ahuja, S. P., & Mani, S. (2012). Availability of Services in the Era of Cloud Computing. *Network and Communication Technologies*, *1*(1), p2. https://doi.org/10.5539/nct.v1n1p2

Alwabel, A., Walters, R., & Wills, G. B. (2015). DesktopCloudSim : Simulation of Node Failures in The Cloud. *The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization CLOUD COMPUTING 2015*, (c), 14–19.

Antunes, J., Neves, N. F., & Verissimo, P. (2008). Detection and prediction of resource-exhaustion vulnerabilities. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, *27513*, 87–96. https://doi.org/10.1109/ISSRE.2008.47

Bai, W., Xi, J., Zhu, J., & Huang, S. (2015). Performance Analysis of Heterogeneous Data Centers in Cloud Computing Using a Complex Queuing Model. *Mathematical Problems in Engineering*, *2015*(4).

Barr, J., & Narin, A. (2010). Building Fault-Tolerant Applications on AWS Failures Shouldn ' t be THAT Interesting. Retrieved from http://d36cz9buwru1tt.cloudfront.net/AWS_Building_Fault_Tolerant_Applications.pdf

Bas, C., Zaidman,A., Deursen, v.A., Leon, M., & Rainer, K. (2009). A Systematic Survey of Program Comprehension through Dynamic Analysis. *IEEE Transactions on Software Engineering* (TSE): 35(5): 684-702, 2009.

Bajaber, W., AlQulaity, M., & Alotaibi, F. S. (2017). Different Techniques to Ensure High Availability in Cloud Computing. *International Journal of Advanced Research in Computer and Communication Engineering*, *6*(11), 11. https://doi.org/10.17148/IJARCCE.2017.61102

Bigelow, S. (2011). The causes and costs of data center system downtime: Advisory Board Q&A. In http://searchdatacenter.techtarget.com/feature/The-causes-and-costs-of-data-center-system-downtime-Advisory-Board-QA, accessed 11/08/14

Brim, M.J., Mattson, T.G., & Scott, S.L.(2001): OSCAR: Open Source Cluster Application Resources. *Ottawa Linux Symposium , Canada*

Bux, M., & Leser, U. (2015). DynamicCloudSim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, *46*, 85–99. https://doi.org/10.1016/j.future.2014.09.007

Buyya, R. (ed.), High Performance Cluster Computing: Architectures and Systems, vol. 1, Prentice Hall, 1999.

Buyya, R., Ranjan, R., & Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. *Proceedings of the 2009 International Conference on High Performance Computing and Simulation, HPCS 2009*, 1–11. https://doi.org/10.1109/HPCSIM.2009.5192685

Caldarelli, A., Ferri, L., & Maffei, M. (2017). Expected benefits and perceived risks of cloud computing: An investigation within an Italian setting. *Technology Analysis & Strategic Management*, *29*(2), 167–180. https://doi.org/10.1080/09537325.2016.1210786

Cao, J., Simonin, M., Cooperman, G., & Morin, C. (2014). Checkpointing as a Service in Heterogeneous Cloud Environments. [Research Report] RR-8633, INRIA-IRISA Rennes Bretagne Atlantique; Northeastern University (Boston, Mass); INRIA. 2014, pp.10. <hal-01086834v2>

Christophe, C., France, C. C., France, P. D., France, M. G., France, Q. G., & Guillaume, N. (2013). Downtime statistics of current cloud solutions, (June), 2–4.

Christophe, C., France, C. C., France, P. D., France, M. G., France, Q. G., & Laurent, S. (2014). Downtime Statistics of Current Cloud Solutions, (March), 1–5.

Cloudharmony.com (2018). Cloud Outages for Amazon, Microsoft and Google 2015, 2016 and 2017 (https://www.theinformation.com/articles/how-aws-stacks-up-against-rivals-on-downtime, accessed 19/02/2018)

Cloud Security Alliance (2010).Top Threats to Cloud Computing V1.0, 1–14.

Crago, S., Dunn, K. , Eads, P., Hochstein, L., Kang,D-I., Kang, M., Modium, D., Singh, K., Suh, J., & Walters, J.P. (2011). Heterogeneous Cloud Computing. *IEEE International Conference on Cluster Computing* , 378–385.

Dang, F., Li, Z., Liu, Y., Zhai, E., Chen, Q. A., Xu, T., … Yang, J. (2019). Understanding Fileless Attacks on Linux-based IoT Devices with HoneyCloud. *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '19*, 482–493. https://doi.org/10.1145/3307334.3326083

Das, P., & Khilar, P. (2013). LBVFT: A Load Balancing Technique for Virtualization and Fault Tolerance in Cloud Computing. *International Journal of Computer Applications*, *69*(28), 14–18.

Davis, J. P., Eisenhardt, K. M., & Bingham, C. B. (2007). Developing Theory Through Simulation Methods. *Academy of Management Review*, *32*(2), 480–499. doi:10.5465/AMR.2007.24351453

Endo, P. T., Rodrigues, M., Gonçalves, G. E., Kelner, J., Sadok, D. H., & Curescu, C. (2016). High availability in clouds: Systematic review and research challenges. *Journal of Cloud Computing*, *5*(1), 16. https://doi.org/10.1186/s13677-016-0066-8

Fehling, C., Leymann, F., Mietzner, R., & Schupeck, W. (2011). A Collection of Patterns for Cloud Types, Cloud ServiceModels, and Cloud-based Application Architectures. *Institute of Architecture of Application Systems*

Forbes.com, "Gmail And Google+ Go Down On Friday, Impacting Millions Of Users", from http://www.forbes.com, accessed 24th January 2014

Fox, A. & Brewer, E.A. (1999). Harvest, yield, and scalable tolerant systems. *HOTOS '99: Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, 174

Gagnaire, M., Diaz, F., Coti, C., & Cerin, C. (2012). Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency*, 2–3. Retrieved from http://iwgcr.org/wp-content/uploads/2012/06/IWGCR-Paris.Ranking-002-en.pdf

Gagnaire, M., Diaz, F., Coti, C., Christophe, C., Kazuhiko, S., Yingjie, X., Delort, P., Smets, J.P., Le Lous, J., Lubiarz, S., Leclerc, P. Downtime statistics of current cloud solutions, IWGCR Report 2013

Gilbert, S. & Lynch, N. (2002). Brewer's conjecture and the feasibility of consistent available partition- tolerant web services. ACM SIGACT News, p.2002

Grispos, G., Storer, T. & Glisson, W. B. (2012). Calm Before the Storm: The Challenges of Cloud Computing in Digital Forensics. *International Journal of Digital Crime and Forensics (IJDCF)*, *4*(2), 28–48. https://doi.org/10.4018/jdcf.2012040103

Gwen, R. (2010). Guidance notes on planning a systematic review. *James Hardiman Library*.

Hauck, M., Huber, M., & Klems, M. (2010). Challenges and opportunities of cloud computing. *Karlsruhe Institute of Technology Technical Report*, *2010*, 31. Retrieved from http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/1978786

Himani, H.S., & Sidhu, H. S. (2014). Comparative Analysis of Scheduling Algorithms of Cloudsim in Cloud Computing. *International Journal of Computer Applications*, *97*(16), 29–33.

IBM. (2010). Taking the Enterprise Data Center into the Cloud. *Computing*. Retrieved from http://dcs.asu.edu/faculty/BruceMillard/indexonly/OffTheNet/%5Cnhttp://resources.idgenterprise.com/original/AST-0019060_Taking_the_Enterprise_Data_Center_into_the_Cloud.pdf

Jose, A.T. (2013). Benchmarking Service Availability for Cloud Computing. *IOSR Journal of Engineering*, *3*(8), 01–03. https://doi.org/10.9790/3021-03860103

Kalyan, R., & Kumar, A. (2015). Trends towards Failover Techniques for Cloud Computing Environment. *International Journal of Advanced Research in Computer Science and Software Engineering*, *5*(1), 983–989.

Kanso, A., & Lemieux, Y. (2013). Achieving High Availability at the Application Level in the Cloud. *2013 IEEE Sixth International Conference on Cloud Computing*, 778–785. doi:10.1109/CLOUD.2013.24

Kaur, P., & Rani, A. (2015). Virtual Machine Migration in Cloud Computing. *International Journal of Grid Distribution Computing*, *8*(5), 337–342.

Keus, K. and Ullman, M. "Availability: Theory and fundamentals for practical evaluation and use". In *Proc. of the lOth Annual Computer Security Applications Conference,* pp. 258-264, USA 1994.

Khalil, I., Khreishah, A., & Azeem, M. (2014). Cloud Computing Security: A Survey. *Computers*, *3*(1), 1–35. https://doi.org/10.3390/computers3010001

Ko, R., Lee, S., & Rajan, V. (2013). Cloud Computing Vulnerability Incidents: A Statistical Overview. *Cloud Security Alliance*, 21.

Leppinen, H., Niemela, P., Silva, N., Sanmark, H., Forsten, H., Yanes, A., … Praks, J. (2019). Developing a Linux-based nanosatellite on-board computer: Flight results from the Aalto-1 mission. *IEEE Aerospace and Electronic Systems Magazine*, *34*(1), 4–14. https://doi.org/10.1109/MAES.2019.170217

Li, Z., Liang, M., Brien, L. O., & Zhang, H. (2013). The Cloud ' s Cloudy Moment : A Systematic Survey of Public Cloud Service Outage. *International Journal of Cloud Computing and Services Science*, *2*(5), 321–331. https://doi.org/10.11591/closer.v2i5.5125

McCrum-Gardner, E. (2008). Which is the correct statistical test to use? *The British Journal of Oral & Maxillofacial Surgery*, *46*(1), 38–41. doi:10.1016/j.bjoms.2007.09.002

Mesbahi, M. R., Rahmani, A. M., & Hosseinzadeh, M. (2018). Reliability and high availability in cloud computing environments: A reference roadmap. *Human-Centric Computing and Information Sciences*, *8*(1), 20. https://doi.org/10.1186/s13673-018-0143-8

Mohapatra, S., Smruti R. K., & Mohanty, S. (2013). A Comparison of Four Popular Heuristics for Load Balancing of Virtual Machines in Cloud Computing. *International Journal of Computer Applications*, *68*(6), 975–8887. https://doi.org/10.5120/11586-6922

Monil, M. A. H., & Rahman, R. M. (2016). VM consolidation approach based on heuristics, fuzzy logic, and migration control. *Journal of Cloud Computing*, *5*(1), 8. https://doi.org/10.1186/s13677-016-0059-7

Msagha, J.M. (2012). High Availability in the Cloud: Building Robust and Dependable Mechanisms. *International Journal of Professional Practice* Vol 3, pp 38-46

Myerson, J. M. (2013). Mitigate risks of cloud resource exhaustion outages Use service level agreements and other proactive tools to avoid, IBM developerworks, 1–9.

Nabi, M., Toeroe, M., & Khendek, F. (2016). Availability in the cloud: State of the art. *Journal of Network and Computer Applications*, *60*, 54–67. https://doi.org/10.1016/j.jnca.2015.11.014

Nagpal, S., Shivam and Kumar, P. (2013). A Study on Adaptive Fault Tolerance in Real Time Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(3), 246–248.

Nasuni white paper, (2010) Disaster Recovery and the Cloud: New Solutions for offsite data protection. Retrieved from http://www.nasuni.com

NIST (2018). Final Version of NIST Cloud Computing Definition Published. Retrieved from **https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published**. (created October 2011, updated January 2018. Accessed on 1st March 2018)

Nita, M. C., Pop, F., Mocanu, M., & Cristea, V. (2014). FIM-SIM: Fault injection module for CloudSim based on statistical distributions. *Journal of Telecommunications and Information Technology*, *2014*(4), 14–23.

Patterson, D. a. (2002). A Simple Way to Estimate the Cost of Downtime. Proc 16th Systems Administration Conf LISA, (November), 185–8. Retrieved from http://www.usenix.org/event/lisa02/tech/full_papers/patterson/patterson_html/

PCWorld.com (2009). "Google Outage Lesson: Don't Get Stuck in a Cloud" from http://www.pcworld.com , accessed 15th May 2009

Pham, C., Phuong Cao, Kalbarczyk, Z., & Iyer, R. K. (2012). Toward a high availability cloud: Techniques and challenges. *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, 1–6. https://doi.org/10.1109/DSNW.2012.6264687

Ponemon Institute (2018). *2018 Cost of Data Breach Study: Impact of Business Continuity Management.* Retrieved from https://www.ibm.com/downloads/cas/AEJYBPWA, accessed 29th November 2019

Pope, C & Mays, N (1995). Qualitative Research: Reaching the parts other methods cannot reach: an introduction to qualitative methods in health and health services research. BMJ 1995, 311:42

Potharaju, R., & Jain, N. (2013). When the Network Crumbles: An Empirical Study of Cloud Network Failures and their Impact on Services. *Proceedings of the 4th Annual Symposium on Cloud Computing - SOCC '13*, *13*, 1–17. https://doi.org/10.1145/2523616.2523638

Prasad, D. (2012). High Availability Based Migration Analysis to Cloud Computing for High Growth Businesses. *International Journal of Computer Networks (IJCN)*, (4), 35–52.

Rabkin, A., & Katz, R. H. (2013). How Hadoop Clusters Break. *IEEE software*, *30*(4), 88-94.

Racuciu, C. & Eftimie,S. (2015). Security threats and risks in cloud computing. *"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XVIII – 2015 – Issue 1 XVIII(1).*

Ritter, E.F., Schoelles, M.J., Quigley, K.S., Klein, L. C. (2011). Determining the number of simulation runs: Treating simulations as theories by not sampling their behavior. *Human-in-the-Loop Simulations: Methods and Practice*, 97–116. https://doi.org/10.1007/978-0-85729-883-6

Rohani, H & Roosta,A.K.(2014). Calculating Total System Availability, *Information Services Organization Amsterdam*, 2014.

Satria, D., Park, D., & Jo, M. (2017). Recovery for overloaded mobile edge computing. *Future Generation Computer Systems*, *70*, 138–147. https://doi.org/10.1016/j.future.2016.06.024

Schwarzkopf, M., Murray, D., & Hand, S. (2012). The seven deadly sins of cloud computing research. *HotCloud, June* 2012

Sheats, R. D., & Pankratz, V. S. (2002). Common statistical tests. *Seminars in Orthodontics*, *8*(2), 77–86. doi:10.1053/sodo.2002.32073

Singh, D., Singh, J., & Chhabra, A. (2012). Evaluating Overheads of Integrated Multilevel Checkpointing Algorithms in Cloud Computing Environment. *International Journal of Computer Network and Information Security*, *4*(5), 29–38. doi:10.5815/ijcnis.2012.05.04

Solissa, D. F., & Abdurohman, M. (2018). Hadoop High Availability with Linux HA. *2018 6th International Conference on Information and Communication Technology (ICoICT)*, 66–69. https://doi.org/10.1109/ICoICT.2018.8528789

Stanik, A., Hoger, M., & Kao, O. (2013). Failover Pattern with a Self-Healing Mechanism for High Availability Cloud Solutions. *2013 International Conference on Cloud Computing and Big Data*, 23–29. doi:10.1109/CLOUDCOM-ASIA.2013.63

Taddei, A. B. (2015). Design and Development of a CloudSim Module to Model and Evaluate Multi-resource Dependencies. BachelorsThesis, University of Zurich

Tchana, A., Broto, L., & Hagimont, D. (2012). Fault Tolerant Approaches in Cloud Computing Infrastructures. *ICAS 2012, The Eighth International Conference on Autonomic and Autonomous Systems pp 42-48*

Teich, P. (2014). Software Defined Availability (SDA): Critical for Managing Datacenter Scale, 1–9.

Thanakornworakij, T., Sharma, R, Blaine. S, Chokchai , L., Zeno, D. G., Riteau, P., & Morin, C. (2012). High availability on cloud with HA-OSCAR. *Euro-Par 2011: Parallel Processing Workshops*, 7156 LNCS(PART 2), 292–301. https://doi.org/10.1007/978-3-642-29740-3-33

Vishwanath, K. V., & Nagappan, N. (2010). Characterizing Cloud Computing Hardware Reliability. *Proceedings of the 1st ACM Symposium on Cloud Computing - SoCC '10*, 193. https://doi.org/10.1145/1807128.1807161

Vmblog.com (2013). The Outrageous Cost of Downtime. http://vmblog.com/archive/2013/09/10/infographic-the-outrageous-costs-of-data-center-downtime.aspx#.U_hrfle03Eo, accessed 11/08/14

VMware. (2007). *VMware High Availability: Concepts, Implementation and Best Practices*, VMWare, Inc.

Vulnerabilities, C., & Group, W. (2013). Cloud Computing Vulnerability Incidents : A Statistical Overview.

Web chaos as Amazon cloud failure crashes major websites….and Playstation Network goes down AGAIN.(2011, 28th May) Retrieved from http://www.dailymail.co.uk/scientech/article-1379474/Web-chaos-Amazon-cloud-failure-crashes-major-websites-Playstation-Net..

Weissman, J., & Ramakrishnan, S. (2009). Using Proxies to Accelerate Cloud Applications. *HotCloud '09 Workshop in Conjunction with USENIX Annual Technical Conference*, 20. Retrieved from http://portal.acm.org/citation.cfm?id=1855553

Weygant, P.S. (2001). Clusters for High Availability: A Primer of HP Solutions: Prentice Hall Professional.

Wu, Y & Guang, H. (2013). Model-based high availability configuration framework for cloud.*MDS 2013 Proceedings of the 2013 Middleware Doctoral Symposium*, (6). https:// doi.org/10.1145/2541534.2541595

Xu, T., & Zhou, Y. (2015). Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Computing Surveys*, *47*(4), 1–41. https://doi.org/10.1145/2791577

Zhou, A., Shangguang, W, Qibo, S, Hua, Z, & Fangchun , Y.(2013). FTCloudSim: a simulation tool for cloud service reliability enhancement mechanisms. *Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference* (MiddlewareDPT '13).

ACM, New York, NY, USA, Article 2 , 2 pages.
DOI=http://dx.doi.org/10.1145/2541614.2541616

APPENDICES

Appendix A: Sample java code for simulator (scenario 1 simulation 1: node failure)

SIMULATION 1 SCENARIO 1: NODE FAILURE CODE

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
```

```
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
 *               of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */


/** This simulation is for Scenario 1 Simulation 1 of Msagha J Mbogholi Thesis
 * entitled: A MODEL DRIVEN APPROACH TO RELATING AVAILABILITY MECHANISMS TO OUTAGE CAUSES IN CLOUD COMPUTING
 * Scenario is based on Node Management
 * In this scenario we have prevented VM migration while also injecting random node failures:
 * The simulation run shows that while cloudlet were sent to the datacenter they failed to return to
 * the broker due to failure of the nodes, i.e. they were stuck and lost in the datacenter.
 */
package org.cloudbus.cloudsim.examples;


import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;


import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
```

```
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.lists.HostList;
import org.cloudbus.cloudsim.lists.VmList;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;


/**
 * In this scenario we attempt to fail several hosts using 10 hosts, 5 users
 * and 30 VMs
 *
 * The scenario further adds an event that allows us to view the host IDs and the hosts themselves as 2 separate outputs
 */


public class Scenario6 {


        /** The cloudlet list. */
        private static List<Cloudlet> cloudletList;
```

```
/** The host list */
private static List<Host> hostList;

/** The host list with hosts removed */
private static List<Host> newhostList;

    /** The vmList. */
    private static List<Vm> vmList;

    private static List<Vm> createVM(int userId, int vms, int idShift) {
            //Creates a container to store VMs. This list is passed to the broker later
            LinkedList<Vm> list = new LinkedList<Vm>();

            //VM Parameters
            long size = 10000; //image size (MB)
            int ram = 512; //vm memory (MB)
            int mips = 250;
            long bw = 1000;
            int pesNumber = 1; //number of cpus
            String vmm = "Xen"; //VMM name

            //create VMs
            Vm[] vm = new Vm[vms];

            for(int i=0;i<vms;i++){
                    vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new
CloudletSchedulerTimeShared());
                    list.add(vm[i]);
            }
```

```
		return list;
	}

	private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift){
		// Creates a container to store Cloudlets
		LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
		//cloudlet parameters
		long length = 40000;
		long fileSize = 300;
		long outputSize = 300;
		int pesNumber = 1;
		UtilizationModel utilizationModel = new UtilizationModelFull();

		Cloudlet[] cloudlet = new Cloudlet[cloudlets];
		for(int i=0;i<cloudlets;i++){
			cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize,
utilizationModel, utilizationModel, utilizationModel);
			// setting the owner of these Cloudlets
			cloudlet[i].setUserId(userId);
			list.add(cloudlet[i]);
		}

		return list;
	}

	//Create the host list

	public <T extends Host> List<T> getHostList() {
		return (List<T>) hostList;
```

```
}


    ////////////////////////// STATIC METHODS //////////////////////

    /**
     * Creates main() to run this scenario
     */
    public static void main(String[] args) {
            Log.printLine("Starting Scenario4...");

            try {
                    // First step: Initialize the CloudSim package. It should be called
                    // before creating any entities.
                    int num_user = 5;   // number of grid users
                    Calendar calendar = Calendar.getInstance();
                    boolean trace_flag = false;  // mean trace events

                    // Initialize the CloudSim library
                    CloudSim.init(num_user, calendar, trace_flag);



                    // Second step: Create Datacenters
```

```
                    //Datacenters are the resource providers in CloudSim. We need at list one of them to run
a CloudSim simulation
                    @SuppressWarnings("unused")
                    Datacenter datacenter0 = createDatacenter("Datacenter_0");

            for (Host host : datacenter0.getHostList()) {

                    HostFaultInjection hostFaultInjection = new HostFaultInjection("HostFaultInjection" +
host.getId());
            hostFaultInjection.setHost(host);

            //newhostList = hostList;
            }




            //Use only one datacenter for this scenario

                    //Third step: Create Broker
                    DatacenterBroker broker = createBroker("Broker_0");
                    int brokerId = broker.getId();

                    //Fourth step: Create VMs and Cloudlets and send them to broker
                    vmList = createVM(brokerId, 30, 0); //creating 30 vms
                    cloudletList = createCloudlet(brokerId, 500, 0); // creating 500 cloudlets
          //hostList created here

                    broker.submitVmList(vmList);
```

```
                broker.submitCloudletList(cloudletList);



                // Fifth step: Starts the simulation
                CloudSim.startSimulation();



                // Final step: Print results when simulation is over
                List<Cloudlet> newList = broker.getCloudletReceivedList();

 List<Host> newList2 = datacenter0.getHostList();
newList2.addAll(datacenter0.getHostList());




        CloudSim.stopSimulation();

    printCloudletList(newList);

                 printHostList(newList2);

                Log.printLine("Scenario with faultInjection finished!");
    }
    catch (Exception e)
    {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
    }
```

```
}

private static Datacenter createDatacenter(String name){

        // Here are the steps needed to create a PowerDatacenter:
        // 1. We need to create a list to store one or more
        //    Machines
        ArrayList<Host> hostList = new ArrayList<Host>();

        // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
        //    create a list to store these PEs before creating
        //    a Machine.
        List<Pe> peList1 = new ArrayList<Pe>();

        int mips = 1000;

        // 3. Create PEs and add these into the list.
        //for a quad-core machine, a list of 4 PEs is required:
        peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
        peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
        peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

        //Another list, for a dual-core machine
        List<Pe> peList2 = new ArrayList<Pe>();

        peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
        peList2.add(new Pe(1, new PeProvisionerSimple(mips)));
```

```
//4. Create Hosts with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 16384; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList1,
                new VmSchedulerSpaceShared(peList1)
        )
); // This is our first machine

hostId++;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerSpaceShared(peList2)
        )
); // Second machine
```

```
hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Third machine

hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Fourth machine


hostId++;
```

```
        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Fifth machine

hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Sixth machine

hostId++;

        hostList.add(
                new Host(
```

```
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Seventh machine


hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Eigth machine

hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
```

```
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Ninth machine


hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Tenth machine



        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";      // system architecture
        String os = "Linux";          // operating system
```

```
		String vmm = "Xen";
		double time_zone = 10.0;	// time zone this resource located
		double cost = 3.0;		// the cost of using processing in this resource
		double costPerMem = 0.05;		// the cost of using memory in this resource
		double costPerStorage = 0.1;	// the cost of using storage in this resource
			double costPerBw = 0.1;			// the cost of using bw in this resource
		LinkedList<Storage> storageList = new LinkedList<Storage>();	//we are not adding SAN devices by now

		DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
	arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);


		// 6. Finally, we need to create a PowerDatacenter object.
		Datacenter datacenter = null;
		try {
			datacenter = new Datacenter(name, characteristics, new VmAllocationPolicySimple(hostList), storageList, 0);
		} catch (Exception e) {
			e.printStackTrace();
		}

		return datacenter;
	}

	//We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according
	//to the specific rules of the simulated scenario
	private static DatacenterBroker createBroker(String name){

		DatacenterBroker broker = null;
```

```
            try {
                        broker = new DatacenterBroker(name);
            } catch (Exception e) {
                        e.printStackTrace();
                        return null;
            }
     return broker;
    }



    /**
     * Prints the Cloudlet objects
     * @param list  list of Cloudlets
     */
    private static void printCloudletList(List<Cloudlet> list) {
            int size = list.size();
            Cloudlet cloudlet;


            String indent = "    ";
            Log.printLine();
            Log.printLine("========== OUTPUT ==========");
            Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
                                    "Data center ID" + indent + "VM ID" + indent + indent +  indent + "Time" +
indent + "Start Time" + indent + "Finish Time");


            DecimalFormat dft = new DecimalFormat("###.##");
            for (int i = 0; i < size; i++) {
                        cloudlet = list.get(i);
                        Log.print(indent + cloudlet.getCloudletId() + indent + indent);
          //cloudlet execution was successful
```

```
                if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                        Log.print("SUCCESS");

                        Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                        indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                        indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                }

        else if (cloudlet.getCloudletStatus() == Cloudlet.FAILED){
                        Log.print("FAILED!");

                        Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                        indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                        indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                }

        else if (cloudlet.getCloudletStatus() == Cloudlet.FAILED_RESOURCE_UNAVAILABLE){
                        Log.print("NO RESOURCDES AVAILABLE!");

                        Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                        indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                        indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                }

        else if (cloudlet.getCloudletStatus() == Cloudlet.INEXEC){
```

```
                                Log.print("CLOUDLETS IN EXECUTION!");


                                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                                indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                            }


                else if (cloudlet.getCloudletStatus() == Cloudlet.CANCELED){
                                Log.printLine("CANCELLED!");


                                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                                indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                            }


                else Log.print("Status of the cloudlet is unkown");
                    }


            }



        //print the host list


        private static void printHostList(List<Host> list) {
                    int size = list.size();
                    Host host;
```

```
        String indent = "    ";
        Log.printLine();
        Log.printLine("========== OUTPUT ==========");
        Log.printLine("Host ID" + indent + indent +  "VM Id" + indent);

        DecimalFormat dft = new DecimalFormat("###.##");
        for (int i = 0; i < size; i++) {
                host = list.get(i);
      Log.printLine(host.getId() + indent + indent + host.getVmList());



                }
        }



    public static class HostFaultInjection extends SimEntity {

                /**
 * 1 means that the host failure is true and 0 otherwise
 */
private static final int HOST_FAILURE = 1;
private Host host;
private Vm vm;
```

```
                public HostFaultInjection(String name) {
                        super(name);
                }

                 @Override
public void startEntity() {
   int delay = delayRandomly(10);
   Log.printLine(getName() + " is starting...");
   schedule(getId(), delay, HOST_FAILURE);
}

@Override
public void processEvent(SimEvent ev) {
   switch (ev.getTag()) {
      case HOST_FAILURE:
         host.setFailed(true); // set to true
         if (host.isFailed()) {
            Log.printLine(CloudSim.clock() + " ---> Host " + host + " FAILURE..." + "---------->" + host.getId());

            Log.printLine();
            for (Vm vm : host.getVmList()) {
               //vm.setHost(host);
               vm.setFailed(true, vm);
               host.getVmsMigratingIn().remove(this.vm);
```

```
                //remove this vm from vms migrating in this host
                host.deallocatePesForVm(vm);
                //deallocate all the processing elements for this Vm









            }

            host.vmDestroy(vm);
        }
        break;

    default:
        Log.printLine(getName() + ": unknown event type");
        break;
  }
}

@Override
public void shutdownEntity() {
   Log.printLine(getName() + ": is shutting down...");
}

/**
 * The value of the delay will be generated within that range (0 -
```

```
     * MAX_TIME_SIMULATION).
     *
     * @param max_simulation represents the max time for simulation.
     * @return
     */
    public int delayRandomly(int max_simulation) {
        return 1 + (int) (Math.random() * max_simulation);
    }

    /**
     * @return the host
     */
    public Host getHost() {
        return host;
    }

    /**
     * @param host the host to set
     */
    public void setHost(Host host) {
        this.host = host;
    }
}

}
```

Appendix B: Sample output from simulator (simulation 1 scenario 1: node failure)

cd D:\Msagha D on Nettop\PhD\cloudsim-3.0.3; "JAVA_HOME=C:\\Program Files\\Java\\jdk1.8.0_20" cmd /c "\"\"C:\\Program Files\\NetBeans 8.1\\java\\maven\\bin\\mvn.bat\" -Dexec.args=\"-classpath %classpath org.cloudbus.cloudsim.examples.Scenario6\" -Dexec.executable=java -Dexec.classpathScope=runtime -Dmaven.ext.class.path=\"C:\\Program Files\\NetBeans 8.1\\java\\maven-nblib\\netbeans-eventspy.jar\" -Dfile.encoding=UTF-8 org.codehaus.mojo:exec-maven-plugin:1.2.1:exec\""

Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Compile on Save turned on) will be used instead of their jar artifacts.

[INFO] Scanning for projects...

[INFO]

[INFO] --

[INFO] Building cloudsim-toolkit 2.1

[INFO] --

[INFO]

[INFO] --- exec-maven-plugin:1.2.1:exec (default-cli) @ cloudsim-toolkit ---

Starting Scenario4...

Initialising...

Starting CloudSim version 3.0

Datacenter_0 is starting...

HostFaultInjection0 is starting...

HostFaultInjection1 is starting...

HostFaultInjection2 is starting...

HostFaultInjection3 is starting...

HostFaultInjection4 is starting...

HostFaultInjection5 is starting...

HostFaultInjection6 is starting...

HostFaultInjection7 is starting...

HostFaultInjection8 is starting...

HostFaultInjection9 is starting...

Broker_0 is starting...

Entities started.

0.0: Broker_0: Cloud Resource List received with 1 resource(s)

0.0: Broker_0: Trying to Create VM #0 in Datacenter_0

0.0: Broker_0: Trying to Create VM #1 in Datacenter_0

0.0: Broker_0: Trying to Create VM #2 in Datacenter_0

0.0: Broker_0: Trying to Create VM #3 in Datacenter_0

0.0: Broker_0: Trying to Create VM #4 in Datacenter_0

0.0: Broker_0: Trying to Create VM #5 in Datacenter_0

0.0: Broker_0: Trying to Create VM #6 in Datacenter_0

0.0: Broker_0: Trying to Create VM #7 in Datacenter_0

0.0: Broker_0: Trying to Create VM #8 in Datacenter_0

0.0: Broker_0: Trying to Create VM #9 in Datacenter_0

0.0: Broker_0: Trying to Create VM #10 in Datacenter_0

0.0: Broker_0: Trying to Create VM #11 in Datacenter_0

0.0: Broker_0: Trying to Create VM #12 in Datacenter_0

0.0: Broker_0: Trying to Create VM #13 in Datacenter_0

0.0: Broker_0: Trying to Create VM #14 in Datacenter_0

0.0: Broker_0: Trying to Create VM #15 in Datacenter_0

0.0: Broker_0: Trying to Create VM #16 in Datacenter_0

0.0: Broker_0: Trying to Create VM #17 in Datacenter_0

0.0: Broker_0: Trying to Create VM #18 in Datacenter_0

0.0: Broker_0: Trying to Create VM #19 in Datacenter_0

0.0: Broker_0: Trying to Create VM #20 in Datacenter_0

0.0: Broker_0: Trying to Create VM #21 in Datacenter_0

0.0: Broker_0: Trying to Create VM #22 in Datacenter_0

0.0: Broker_0: Trying to Create VM #23 in Datacenter_0

0.0: Broker_0: Trying to Create VM #24 in Datacenter_0

0.0: Broker_0: Trying to Create VM #25 in Datacenter_0

0.0: Broker_0: Trying to Create VM #26 in Datacenter_0

0.0: Broker_0: Trying to Create VM #27 in Datacenter_0

0.0: Broker_0: Trying to Create VM #28 in Datacenter_0

0.0: Broker_0: Trying to Create VM #29 in Datacenter_0

0.1: Broker_0: VM #0 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #1 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #2 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #3 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #4 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #5 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #6 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #7 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #8 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #9 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #10 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #11 has been created in Datacenter #2, Host #1

0.1: Broker_0: VM #12 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #13 has been created in Datacenter #2, Host #3

0.1: Broker_0: VM #14 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #15 has been created in Datacenter #2, Host #5

0.1: Broker_0: VM #16 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #17 has been created in Datacenter #2, Host #7

0.1: Broker_0: VM #18 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #19 has been created in Datacenter #2, Host #9

0.1: Broker_0: VM #20 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #21 has been created in Datacenter #2, Host #1

0.1: Broker_0: VM #22 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #23 has been created in Datacenter #2, Host #3

0.1: Broker_0: VM #24 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #25 has been created in Datacenter #2, Host #5

0.1: Broker_0: VM #26 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #27 has been created in Datacenter #2, Host #7

0.1: Broker_0: VM #28 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #29 has been created in Datacenter #2, Host #9

0.1: Broker_0: Sending cloudlet 0 to VM #0

0.1: Broker_0: Sending cloudlet 1 to VM #1

0.1: Broker_0: Sending cloudlet 2 to VM #2

0.1: Broker_0: Sending cloudlet 3 to VM #3

0.1: Broker_0: Sending cloudlet 4 to VM #4

0.1: Broker_0: Sending cloudlet 5 to VM #5

0.1: Broker_0: Sending cloudlet 6 to VM #6

0.1: Broker_0: Sending cloudlet 7 to VM #7

0.1: Broker_0: Sending cloudlet 8 to VM #8

0.1: Broker_0: Sending cloudlet 9 to VM #9

0.1: Broker_0: Sending cloudlet 10 to VM #10

0.1: Broker_0: Sending cloudlet 11 to VM #11

0.1: Broker_0: Sending cloudlet 12 to VM #12

0.1: Broker_0: Sending cloudlet 13 to VM #13

0.1: Broker_0: Sending cloudlet 14 to VM #14

0.1: Broker_0: Sending cloudlet 15 to VM #15

0.1: Broker_0: Sending cloudlet 16 to VM #16

0.1: Broker_0: Sending cloudlet 17 to VM #17

0.1: Broker_0: Sending cloudlet 18 to VM #18

0.1: Broker_0: Sending cloudlet 19 to VM #19

0.1: Broker_0: Sending cloudlet 20 to VM #20

0.1: Broker_0: Sending cloudlet 21 to VM #21

0.1: Broker_0: Sending cloudlet 22 to VM #22

0.1: Broker_0: Sending cloudlet 23 to VM #23
0.1: Broker_0: Sending cloudlet 24 to VM #24
0.1: Broker_0: Sending cloudlet 25 to VM #25
0.1: Broker_0: Sending cloudlet 26 to VM #26
0.1: Broker_0: Sending cloudlet 27 to VM #27
0.1: Broker_0: Sending cloudlet 28 to VM #28
0.1: Broker_0: Sending cloudlet 29 to VM #29
0.1: Broker_0: Sending cloudlet 30 to VM #0
0.1: Broker_0: Sending cloudlet 31 to VM #1
0.1: Broker_0: Sending cloudlet 32 to VM #2
0.1: Broker_0: Sending cloudlet 33 to VM #3
0.1: Broker_0: Sending cloudlet 34 to VM #4
0.1: Broker_0: Sending cloudlet 35 to VM #5
0.1: Broker_0: Sending cloudlet 36 to VM #6
0.1: Broker_0: Sending cloudlet 37 to VM #7
0.1: Broker_0: Sending cloudlet 38 to VM #8
0.1: Broker_0: Sending cloudlet 39 to VM #9
0.1: Broker_0: Sending cloudlet 40 to VM #10
0.1: Broker_0: Sending cloudlet 41 to VM #11
0.1: Broker_0: Sending cloudlet 42 to VM #12
0.1: Broker_0: Sending cloudlet 43 to VM #13
0.1: Broker_0: Sending cloudlet 44 to VM #14
0.1: Broker_0: Sending cloudlet 45 to VM #15
0.1: Broker_0: Sending cloudlet 46 to VM #16
0.1: Broker_0: Sending cloudlet 47 to VM #17
0.1: Broker_0: Sending cloudlet 48 to VM #18
0.1: Broker_0: Sending cloudlet 49 to VM #19
0.1: Broker_0: Sending cloudlet 50 to VM #20
0.1: Broker_0: Sending cloudlet 51 to VM #21
0.1: Broker_0: Sending cloudlet 52 to VM #22
0.1: Broker_0: Sending cloudlet 53 to VM #23
0.1: Broker_0: Sending cloudlet 54 to VM #24
0.1: Broker_0: Sending cloudlet 55 to VM #25
0.1: Broker_0: Sending cloudlet 56 to VM #26
0.1: Broker_0: Sending cloudlet 57 to VM #27
0.1: Broker_0: Sending cloudlet 58 to VM #28
0.1: Broker_0: Sending cloudlet 59 to VM #29
0.1: Broker_0: Sending cloudlet 60 to VM #0
0.1: Broker_0: Sending cloudlet 61 to VM #1
0.1: Broker_0: Sending cloudlet 62 to VM #2
0.1: Broker_0: Sending cloudlet 63 to VM #3
0.1: Broker_0: Sending cloudlet 64 to VM #4
0.1: Broker_0: Sending cloudlet 65 to VM #5
0.1: Broker_0: Sending cloudlet 66 to VM #6
0.1: Broker_0: Sending cloudlet 67 to VM #7
0.1: Broker_0: Sending cloudlet 68 to VM #8
0.1: Broker_0: Sending cloudlet 69 to VM #9
0.1: Broker_0: Sending cloudlet 70 to VM #10
0.1: Broker_0: Sending cloudlet 71 to VM #11
0.1: Broker_0: Sending cloudlet 72 to VM #12
0.1: Broker_0: Sending cloudlet 73 to VM #13
0.1: Broker_0: Sending cloudlet 74 to VM #14
0.1: Broker_0: Sending cloudlet 75 to VM #15
0.1: Broker_0: Sending cloudlet 76 to VM #16

0.1: Broker_0: Sending cloudlet 77 to VM #17

0.1: Broker_0: Sending cloudlet 78 to VM #18

0.1: Broker_0: Sending cloudlet 79 to VM #19

0.1: Broker_0: Sending cloudlet 80 to VM #20

0.1: Broker_0: Sending cloudlet 81 to VM #21

0.1: Broker_0: Sending cloudlet 82 to VM #22

0.1: Broker_0: Sending cloudlet 83 to VM #23

0.1: Broker_0: Sending cloudlet 84 to VM #24

0.1: Broker_0: Sending cloudlet 85 to VM #25

0.1: Broker_0: Sending cloudlet 86 to VM #26

0.1: Broker_0: Sending cloudlet 87 to VM #27

0.1: Broker_0: Sending cloudlet 88 to VM #28

0.1: Broker_0: Sending cloudlet 89 to VM #29

0.1: Broker_0: Sending cloudlet 90 to VM #0

0.1: Broker_0: Sending cloudlet 91 to VM #1

0.1: Broker_0: Sending cloudlet 92 to VM #2

0.1: Broker_0: Sending cloudlet 93 to VM #3

0.1: Broker_0: Sending cloudlet 94 to VM #4

0.1: Broker_0: Sending cloudlet 95 to VM #5

0.1: Broker_0: Sending cloudlet 96 to VM #6

0.1: Broker_0: Sending cloudlet 97 to VM #7

0.1: Broker_0: Sending cloudlet 98 to VM #8

0.1: Broker_0: Sending cloudlet 99 to VM #9

0.1: Broker_0: Sending cloudlet 100 to VM #10

0.1: Broker_0: Sending cloudlet 101 to VM #11

0.1: Broker_0: Sending cloudlet 102 to VM #12

0.1: Broker_0: Sending cloudlet 103 to VM #13

0.1: Broker_0: Sending cloudlet 104 to VM #14

0.1: Broker_0: Sending cloudlet 105 to VM #15

0.1: Broker_0: Sending cloudlet 106 to VM #16

0.1: Broker_0: Sending cloudlet 107 to VM #17

0.1: Broker_0: Sending cloudlet 108 to VM #18

0.1: Broker_0: Sending cloudlet 109 to VM #19

0.1: Broker_0: Sending cloudlet 110 to VM #20

0.1: Broker_0: Sending cloudlet 111 to VM #21

0.1: Broker_0: Sending cloudlet 112 to VM #22

0.1: Broker_0: Sending cloudlet 113 to VM #23

0.1: Broker_0: Sending cloudlet 114 to VM #24

0.1: Broker_0: Sending cloudlet 115 to VM #25

0.1: Broker_0: Sending cloudlet 116 to VM #26

0.1: Broker_0: Sending cloudlet 117 to VM #27

0.1: Broker_0: Sending cloudlet 118 to VM #28

0.1: Broker_0: Sending cloudlet 119 to VM #29

0.1: Broker_0: Sending cloudlet 120 to VM #0

0.1: Broker_0: Sending cloudlet 121 to VM #1

0.1: Broker_0: Sending cloudlet 122 to VM #2

0.1: Broker_0: Sending cloudlet 123 to VM #3

0.1: Broker_0: Sending cloudlet 124 to VM #4

0.1: Broker_0: Sending cloudlet 125 to VM #5

0.1: Broker_0: Sending cloudlet 126 to VM #6

0.1: Broker_0: Sending cloudlet 127 to VM #7

0.1: Broker_0: Sending cloudlet 128 to VM #8

0.1: Broker_0: Sending cloudlet 129 to VM #9

0.1: Broker_0: Sending cloudlet 130 to VM #10

0.1: Broker_0: Sending cloudlet 131 to VM #11
0.1: Broker_0: Sending cloudlet 132 to VM #12
0.1: Broker_0: Sending cloudlet 133 to VM #13
0.1: Broker_0: Sending cloudlet 134 to VM #14
0.1: Broker_0: Sending cloudlet 135 to VM #15
0.1: Broker_0: Sending cloudlet 136 to VM #16
0.1: Broker_0: Sending cloudlet 137 to VM #17
0.1: Broker_0: Sending cloudlet 138 to VM #18
0.1: Broker_0: Sending cloudlet 139 to VM #19
0.1: Broker_0: Sending cloudlet 140 to VM #20
0.1: Broker_0: Sending cloudlet 141 to VM #21
0.1: Broker_0: Sending cloudlet 142 to VM #22
0.1: Broker_0: Sending cloudlet 143 to VM #23
0.1: Broker_0: Sending cloudlet 144 to VM #24
0.1: Broker_0: Sending cloudlet 145 to VM #25
0.1: Broker_0: Sending cloudlet 146 to VM #26
0.1: Broker_0: Sending cloudlet 147 to VM #27
0.1: Broker_0: Sending cloudlet 148 to VM #28
0.1: Broker_0: Sending cloudlet 149 to VM #29
0.1: Broker_0: Sending cloudlet 150 to VM #0
0.1: Broker_0: Sending cloudlet 151 to VM #1
0.1: Broker_0: Sending cloudlet 152 to VM #2
0.1: Broker_0: Sending cloudlet 153 to VM #3
0.1: Broker_0: Sending cloudlet 154 to VM #4
0.1: Broker_0: Sending cloudlet 155 to VM #5
0.1: Broker_0: Sending cloudlet 156 to VM #6
0.1: Broker_0: Sending cloudlet 157 to VM #7
0.1: Broker_0: Sending cloudlet 158 to VM #8
0.1: Broker_0: Sending cloudlet 159 to VM #9
0.1: Broker_0: Sending cloudlet 160 to VM #10
0.1: Broker_0: Sending cloudlet 161 to VM #11
0.1: Broker_0: Sending cloudlet 162 to VM #12
0.1: Broker_0: Sending cloudlet 163 to VM #13
0.1: Broker_0: Sending cloudlet 164 to VM #14
0.1: Broker_0: Sending cloudlet 165 to VM #15
0.1: Broker_0: Sending cloudlet 166 to VM #16
0.1: Broker_0: Sending cloudlet 167 to VM #17
0.1: Broker_0: Sending cloudlet 168 to VM #18
0.1: Broker_0: Sending cloudlet 169 to VM #19
0.1: Broker_0: Sending cloudlet 170 to VM #20
0.1: Broker_0: Sending cloudlet 171 to VM #21
0.1: Broker_0: Sending cloudlet 172 to VM #22
0.1: Broker_0: Sending cloudlet 173 to VM #23
0.1: Broker_0: Sending cloudlet 174 to VM #24
0.1: Broker_0: Sending cloudlet 175 to VM #25
0.1: Broker_0: Sending cloudlet 176 to VM #26
0.1: Broker_0: Sending cloudlet 177 to VM #27
0.1: Broker_0: Sending cloudlet 178 to VM #28
0.1: Broker_0: Sending cloudlet 179 to VM #29
0.1: Broker_0: Sending cloudlet 180 to VM #0
0.1: Broker_0: Sending cloudlet 181 to VM #1
0.1: Broker_0: Sending cloudlet 182 to VM #2
0.1: Broker_0: Sending cloudlet 183 to VM #3
0.1: Broker_0: Sending cloudlet 184 to VM #4

0.1: Broker_0: Sending cloudlet 185 to VM #5
0.1: Broker_0: Sending cloudlet 186 to VM #6
0.1: Broker_0: Sending cloudlet 187 to VM #7
0.1: Broker_0: Sending cloudlet 188 to VM #8
0.1: Broker_0: Sending cloudlet 189 to VM #9
0.1: Broker_0: Sending cloudlet 190 to VM #10
0.1: Broker_0: Sending cloudlet 191 to VM #11
0.1: Broker_0: Sending cloudlet 192 to VM #12
0.1: Broker_0: Sending cloudlet 193 to VM #13
0.1: Broker_0: Sending cloudlet 194 to VM #14
0.1: Broker_0: Sending cloudlet 195 to VM #15
0.1: Broker_0: Sending cloudlet 196 to VM #16
0.1: Broker_0: Sending cloudlet 197 to VM #17
0.1: Broker_0: Sending cloudlet 198 to VM #18
0.1: Broker_0: Sending cloudlet 199 to VM #19
0.1: Broker_0: Sending cloudlet 200 to VM #20
0.1: Broker_0: Sending cloudlet 201 to VM #21
0.1: Broker_0: Sending cloudlet 202 to VM #22
0.1: Broker_0: Sending cloudlet 203 to VM #23
0.1: Broker_0: Sending cloudlet 204 to VM #24
0.1: Broker_0: Sending cloudlet 205 to VM #25
0.1: Broker_0: Sending cloudlet 206 to VM #26
0.1: Broker_0: Sending cloudlet 207 to VM #27
0.1: Broker_0: Sending cloudlet 208 to VM #28
0.1: Broker_0: Sending cloudlet 209 to VM #29
0.1: Broker_0: Sending cloudlet 210 to VM #0
0.1: Broker_0: Sending cloudlet 211 to VM #1
0.1: Broker_0: Sending cloudlet 212 to VM #2
0.1: Broker_0: Sending cloudlet 213 to VM #3
0.1: Broker_0: Sending cloudlet 214 to VM #4
0.1: Broker_0: Sending cloudlet 215 to VM #5
0.1: Broker_0: Sending cloudlet 216 to VM #6
0.1: Broker_0: Sending cloudlet 217 to VM #7
0.1: Broker_0: Sending cloudlet 218 to VM #8
0.1: Broker_0: Sending cloudlet 219 to VM #9
0.1: Broker_0: Sending cloudlet 220 to VM #10
0.1: Broker_0: Sending cloudlet 221 to VM #11
0.1: Broker_0: Sending cloudlet 222 to VM #12
0.1: Broker_0: Sending cloudlet 223 to VM #13
0.1: Broker_0: Sending cloudlet 224 to VM #14
0.1: Broker_0: Sending cloudlet 225 to VM #15
0.1: Broker_0: Sending cloudlet 226 to VM #16
0.1: Broker_0: Sending cloudlet 227 to VM #17
0.1: Broker_0: Sending cloudlet 228 to VM #18
0.1: Broker_0: Sending cloudlet 229 to VM #19
0.1: Broker_0: Sending cloudlet 230 to VM #20
0.1: Broker_0: Sending cloudlet 231 to VM #21
0.1: Broker_0: Sending cloudlet 232 to VM #22
0.1: Broker_0: Sending cloudlet 233 to VM #23
0.1: Broker_0: Sending cloudlet 234 to VM #24
0.1: Broker_0: Sending cloudlet 235 to VM #25
0.1: Broker_0: Sending cloudlet 236 to VM #26
0.1: Broker_0: Sending cloudlet 237 to VM #27
0.1: Broker_0: Sending cloudlet 238 to VM #28

0.1: Broker_0: Sending cloudlet 239 to VM #29

0.1: Broker_0: Sending cloudlet 240 to VM #0

0.1: Broker_0: Sending cloudlet 241 to VM #1

0.1: Broker_0: Sending cloudlet 242 to VM #2

0.1: Broker_0: Sending cloudlet 243 to VM #3

0.1: Broker_0: Sending cloudlet 244 to VM #4

0.1: Broker_0: Sending cloudlet 245 to VM #5

0.1: Broker_0: Sending cloudlet 246 to VM #6

0.1: Broker_0: Sending cloudlet 247 to VM #7

0.1: Broker_0: Sending cloudlet 248 to VM #8

0.1: Broker_0: Sending cloudlet 249 to VM #9

0.1: Broker_0: Sending cloudlet 250 to VM #10

0.1: Broker_0: Sending cloudlet 251 to VM #11

0.1: Broker_0: Sending cloudlet 252 to VM #12

0.1: Broker_0: Sending cloudlet 253 to VM #13

0.1: Broker_0: Sending cloudlet 254 to VM #14

0.1: Broker_0: Sending cloudlet 255 to VM #15

0.1: Broker_0: Sending cloudlet 256 to VM #16

0.1: Broker_0: Sending cloudlet 257 to VM #17

0.1: Broker_0: Sending cloudlet 258 to VM #18

0.1: Broker_0: Sending cloudlet 259 to VM #19

0.1: Broker_0: Sending cloudlet 260 to VM #20

0.1: Broker_0: Sending cloudlet 261 to VM #21

0.1: Broker_0: Sending cloudlet 262 to VM #22

0.1: Broker_0: Sending cloudlet 263 to VM #23

0.1: Broker_0: Sending cloudlet 264 to VM #24

0.1: Broker_0: Sending cloudlet 265 to VM #25

0.1: Broker_0: Sending cloudlet 266 to VM #26

0.1: Broker_0: Sending cloudlet 267 to VM #27

0.1: Broker_0: Sending cloudlet 268 to VM #28

0.1: Broker_0: Sending cloudlet 269 to VM #29

0.1: Broker_0: Sending cloudlet 270 to VM #0

0.1: Broker_0: Sending cloudlet 271 to VM #1

0.1: Broker_0: Sending cloudlet 272 to VM #2

0.1: Broker_0: Sending cloudlet 273 to VM #3

0.1: Broker_0: Sending cloudlet 274 to VM #4

0.1: Broker_0: Sending cloudlet 275 to VM #5

0.1: Broker_0: Sending cloudlet 276 to VM #6

0.1: Broker_0: Sending cloudlet 277 to VM #7

0.1: Broker_0: Sending cloudlet 278 to VM #8

0.1: Broker_0: Sending cloudlet 279 to VM #9

0.1: Broker_0: Sending cloudlet 280 to VM #10

0.1: Broker_0: Sending cloudlet 281 to VM #11

0.1: Broker_0: Sending cloudlet 282 to VM #12

0.1: Broker_0: Sending cloudlet 283 to VM #13

0.1: Broker_0: Sending cloudlet 284 to VM #14

0.1: Broker_0: Sending cloudlet 285 to VM #15

0.1: Broker_0: Sending cloudlet 286 to VM #16

0.1: Broker_0: Sending cloudlet 287 to VM #17

0.1: Broker_0: Sending cloudlet 288 to VM #18

0.1: Broker_0: Sending cloudlet 289 to VM #19

0.1: Broker_0: Sending cloudlet 290 to VM #20

0.1: Broker_0: Sending cloudlet 291 to VM #21

0.1: Broker_0: Sending cloudlet 292 to VM #22

0.1: Broker_0: Sending cloudlet 293 to VM #23
0.1: Broker_0: Sending cloudlet 294 to VM #24
0.1: Broker_0: Sending cloudlet 295 to VM #25
0.1: Broker_0: Sending cloudlet 296 to VM #26
0.1: Broker_0: Sending cloudlet 297 to VM #27
0.1: Broker_0: Sending cloudlet 298 to VM #28
0.1: Broker_0: Sending cloudlet 299 to VM #29
0.1: Broker_0: Sending cloudlet 300 to VM #0
0.1: Broker_0: Sending cloudlet 301 to VM #1
0.1: Broker_0: Sending cloudlet 302 to VM #2
0.1: Broker_0: Sending cloudlet 303 to VM #3
0.1: Broker_0: Sending cloudlet 304 to VM #4
0.1: Broker_0: Sending cloudlet 305 to VM #5
0.1: Broker_0: Sending cloudlet 306 to VM #6
0.1: Broker_0: Sending cloudlet 307 to VM #7
0.1: Broker_0: Sending cloudlet 308 to VM #8
0.1: Broker_0: Sending cloudlet 309 to VM #9
0.1: Broker_0: Sending cloudlet 310 to VM #10
0.1: Broker_0: Sending cloudlet 311 to VM #11
0.1: Broker_0: Sending cloudlet 312 to VM #12
0.1: Broker_0: Sending cloudlet 313 to VM #13
0.1: Broker_0: Sending cloudlet 314 to VM #14
0.1: Broker_0: Sending cloudlet 315 to VM #15
0.1: Broker_0: Sending cloudlet 316 to VM #16
0.1: Broker_0: Sending cloudlet 317 to VM #17
0.1: Broker_0: Sending cloudlet 318 to VM #18
0.1: Broker_0: Sending cloudlet 319 to VM #19
0.1: Broker_0: Sending cloudlet 320 to VM #20
0.1: Broker_0: Sending cloudlet 321 to VM #21
0.1: Broker_0: Sending cloudlet 322 to VM #22
0.1: Broker_0: Sending cloudlet 323 to VM #23
0.1: Broker_0: Sending cloudlet 324 to VM #24
0.1: Broker_0: Sending cloudlet 325 to VM #25
0.1: Broker_0: Sending cloudlet 326 to VM #26
0.1: Broker_0: Sending cloudlet 327 to VM #27
0.1: Broker_0: Sending cloudlet 328 to VM #28
0.1: Broker_0: Sending cloudlet 329 to VM #29
0.1: Broker_0: Sending cloudlet 330 to VM #0
0.1: Broker_0: Sending cloudlet 331 to VM #1
0.1: Broker_0: Sending cloudlet 332 to VM #2
0.1: Broker_0: Sending cloudlet 333 to VM #3
0.1: Broker_0: Sending cloudlet 334 to VM #4
0.1: Broker_0: Sending cloudlet 335 to VM #5
0.1: Broker_0: Sending cloudlet 336 to VM #6
0.1: Broker_0: Sending cloudlet 337 to VM #7
0.1: Broker_0: Sending cloudlet 338 to VM #8
0.1: Broker_0: Sending cloudlet 339 to VM #9
0.1: Broker_0: Sending cloudlet 340 to VM #10
0.1: Broker_0: Sending cloudlet 341 to VM #11
0.1: Broker_0: Sending cloudlet 342 to VM #12
0.1: Broker_0: Sending cloudlet 343 to VM #13
0.1: Broker_0: Sending cloudlet 344 to VM #14
0.1: Broker_0: Sending cloudlet 345 to VM #15
0.1: Broker_0: Sending cloudlet 346 to VM #16

0.1: Broker_0: Sending cloudlet 347 to VM #17
0.1: Broker_0: Sending cloudlet 348 to VM #18
0.1: Broker_0: Sending cloudlet 349 to VM #19
0.1: Broker_0: Sending cloudlet 350 to VM #20
0.1: Broker_0: Sending cloudlet 351 to VM #21
0.1: Broker_0: Sending cloudlet 352 to VM #22
0.1: Broker_0: Sending cloudlet 353 to VM #23
0.1: Broker_0: Sending cloudlet 354 to VM #24
0.1: Broker_0: Sending cloudlet 355 to VM #25
0.1: Broker_0: Sending cloudlet 356 to VM #26
0.1: Broker_0: Sending cloudlet 357 to VM #27
0.1: Broker_0: Sending cloudlet 358 to VM #28
0.1: Broker_0: Sending cloudlet 359 to VM #29
0.1: Broker_0: Sending cloudlet 360 to VM #0
0.1: Broker_0: Sending cloudlet 361 to VM #1
0.1: Broker_0: Sending cloudlet 362 to VM #2
0.1: Broker_0: Sending cloudlet 363 to VM #3
0.1: Broker_0: Sending cloudlet 364 to VM #4
0.1: Broker_0: Sending cloudlet 365 to VM #5
0.1: Broker_0: Sending cloudlet 366 to VM #6
0.1: Broker_0: Sending cloudlet 367 to VM #7
0.1: Broker_0: Sending cloudlet 368 to VM #8
0.1: Broker_0: Sending cloudlet 369 to VM #9
0.1: Broker_0: Sending cloudlet 370 to VM #10
0.1: Broker_0: Sending cloudlet 371 to VM #11
0.1: Broker_0: Sending cloudlet 372 to VM #12
0.1: Broker_0: Sending cloudlet 373 to VM #13
0.1: Broker_0: Sending cloudlet 374 to VM #14
0.1: Broker_0: Sending cloudlet 375 to VM #15
0.1: Broker_0: Sending cloudlet 376 to VM #16
0.1: Broker_0: Sending cloudlet 377 to VM #17
0.1: Broker_0: Sending cloudlet 378 to VM #18
0.1: Broker_0: Sending cloudlet 379 to VM #19
0.1: Broker_0: Sending cloudlet 380 to VM #20
0.1: Broker_0: Sending cloudlet 381 to VM #21
0.1: Broker_0: Sending cloudlet 382 to VM #22
0.1: Broker_0: Sending cloudlet 383 to VM #23
0.1: Broker_0: Sending cloudlet 384 to VM #24
0.1: Broker_0: Sending cloudlet 385 to VM #25
0.1: Broker_0: Sending cloudlet 386 to VM #26
0.1: Broker_0: Sending cloudlet 387 to VM #27
0.1: Broker_0: Sending cloudlet 388 to VM #28
0.1: Broker_0: Sending cloudlet 389 to VM #29
0.1: Broker_0: Sending cloudlet 390 to VM #0
0.1: Broker_0: Sending cloudlet 391 to VM #1
0.1: Broker_0: Sending cloudlet 392 to VM #2
0.1: Broker_0: Sending cloudlet 393 to VM #3
0.1: Broker_0: Sending cloudlet 394 to VM #4
0.1: Broker_0: Sending cloudlet 395 to VM #5
0.1: Broker_0: Sending cloudlet 396 to VM #6
0.1: Broker_0: Sending cloudlet 397 to VM #7
0.1: Broker_0: Sending cloudlet 398 to VM #8
0.1: Broker_0: Sending cloudlet 399 to VM #9
0.1: Broker_0: Sending cloudlet 400 to VM #10

0.1: Broker_0: Sending cloudlet 401 to VM #11
0.1: Broker_0: Sending cloudlet 402 to VM #12
0.1: Broker_0: Sending cloudlet 403 to VM #13
0.1: Broker_0: Sending cloudlet 404 to VM #14
0.1: Broker_0: Sending cloudlet 405 to VM #15
0.1: Broker_0: Sending cloudlet 406 to VM #16
0.1: Broker_0: Sending cloudlet 407 to VM #17
0.1: Broker_0: Sending cloudlet 408 to VM #18
0.1: Broker_0: Sending cloudlet 409 to VM #19
0.1: Broker_0: Sending cloudlet 410 to VM #20
0.1: Broker_0: Sending cloudlet 411 to VM #21
0.1: Broker_0: Sending cloudlet 412 to VM #22
0.1: Broker_0: Sending cloudlet 413 to VM #23
0.1: Broker_0: Sending cloudlet 414 to VM #24
0.1: Broker_0: Sending cloudlet 415 to VM #25
0.1: Broker_0: Sending cloudlet 416 to VM #26
0.1: Broker_0: Sending cloudlet 417 to VM #27
0.1: Broker_0: Sending cloudlet 418 to VM #28
0.1: Broker_0: Sending cloudlet 419 to VM #29
0.1: Broker_0: Sending cloudlet 420 to VM #0
0.1: Broker_0: Sending cloudlet 421 to VM #1
0.1: Broker_0: Sending cloudlet 422 to VM #2
0.1: Broker_0: Sending cloudlet 423 to VM #3
0.1: Broker_0: Sending cloudlet 424 to VM #4
0.1: Broker_0: Sending cloudlet 425 to VM #5
0.1: Broker_0: Sending cloudlet 426 to VM #6
0.1: Broker_0: Sending cloudlet 427 to VM #7
0.1: Broker_0: Sending cloudlet 428 to VM #8
0.1: Broker_0: Sending cloudlet 429 to VM #9
0.1: Broker_0: Sending cloudlet 430 to VM #10
0.1: Broker_0: Sending cloudlet 431 to VM #11
0.1: Broker_0: Sending cloudlet 432 to VM #12
0.1: Broker_0: Sending cloudlet 433 to VM #13
0.1: Broker_0: Sending cloudlet 434 to VM #14
0.1: Broker_0: Sending cloudlet 435 to VM #15
0.1: Broker_0: Sending cloudlet 436 to VM #16
0.1: Broker_0: Sending cloudlet 437 to VM #17
0.1: Broker_0: Sending cloudlet 438 to VM #18
0.1: Broker_0: Sending cloudlet 439 to VM #19
0.1: Broker_0: Sending cloudlet 440 to VM #20
0.1: Broker_0: Sending cloudlet 441 to VM #21
0.1: Broker_0: Sending cloudlet 442 to VM #22
0.1: Broker_0: Sending cloudlet 443 to VM #23
0.1: Broker_0: Sending cloudlet 444 to VM #24
0.1: Broker_0: Sending cloudlet 445 to VM #25
0.1: Broker_0: Sending cloudlet 446 to VM #26
0.1: Broker_0: Sending cloudlet 447 to VM #27
0.1: Broker_0: Sending cloudlet 448 to VM #28
0.1: Broker_0: Sending cloudlet 449 to VM #29
0.1: Broker_0: Sending cloudlet 450 to VM #0
0.1: Broker_0: Sending cloudlet 451 to VM #1
0.1: Broker_0: Sending cloudlet 452 to VM #2
0.1: Broker_0: Sending cloudlet 453 to VM #3
0.1: Broker_0: Sending cloudlet 454 to VM #4

0.1: Broker_0: Sending cloudlet 455 to VM #5

0.1: Broker_0: Sending cloudlet 456 to VM #6

0.1: Broker_0: Sending cloudlet 457 to VM #7

0.1: Broker_0: Sending cloudlet 458 to VM #8

0.1: Broker_0: Sending cloudlet 459 to VM #9

0.1: Broker_0: Sending cloudlet 460 to VM #10

0.1: Broker_0: Sending cloudlet 461 to VM #11

0.1: Broker_0: Sending cloudlet 462 to VM #12

0.1: Broker_0: Sending cloudlet 463 to VM #13

0.1: Broker_0: Sending cloudlet 464 to VM #14

0.1: Broker_0: Sending cloudlet 465 to VM #15

0.1: Broker_0: Sending cloudlet 466 to VM #16

0.1: Broker_0: Sending cloudlet 467 to VM #17

0.1: Broker_0: Sending cloudlet 468 to VM #18

0.1: Broker_0: Sending cloudlet 469 to VM #19

0.1: Broker_0: Sending cloudlet 470 to VM #20

0.1: Broker_0: Sending cloudlet 471 to VM #21

0.1: Broker_0: Sending cloudlet 472 to VM #22

0.1: Broker_0: Sending cloudlet 473 to VM #23

0.1: Broker_0: Sending cloudlet 474 to VM #24

0.1: Broker_0: Sending cloudlet 475 to VM #25

0.1: Broker_0: Sending cloudlet 476 to VM #26

0.1: Broker_0: Sending cloudlet 477 to VM #27

0.1: Broker_0: Sending cloudlet 478 to VM #28

0.1: Broker_0: Sending cloudlet 479 to VM #29

0.1: Broker_0: Sending cloudlet 480 to VM #0

0.1: Broker_0: Sending cloudlet 481 to VM #1

0.1: Broker_0: Sending cloudlet 482 to VM #2

0.1: Broker_0: Sending cloudlet 483 to VM #3

0.1: Broker_0: Sending cloudlet 484 to VM #4

0.1: Broker_0: Sending cloudlet 485 to VM #5

0.1: Broker_0: Sending cloudlet 486 to VM #6

0.1: Broker_0: Sending cloudlet 487 to VM #7

0.1: Broker_0: Sending cloudlet 488 to VM #8

0.1: Broker_0: Sending cloudlet 489 to VM #9

0.1: Broker_0: Sending cloudlet 490 to VM #10

0.1: Broker_0: Sending cloudlet 491 to VM #11

0.1: Broker_0: Sending cloudlet 492 to VM #12

0.1: Broker_0: Sending cloudlet 493 to VM #13

0.1: Broker_0: Sending cloudlet 494 to VM #14

0.1: Broker_0: Sending cloudlet 495 to VM #15

0.1: Broker_0: Sending cloudlet 496 to VM #16

0.1: Broker_0: Sending cloudlet 497 to VM #17

0.1: Broker_0: Sending cloudlet 498 to VM #18

0.1: Broker_0: Sending cloudlet 499 to VM #19

2.0 ---> Host org.cloudbus.cloudsim.Host@4554617c FAILURE...---------->1

3.0 ---> Host org.cloudbus.cloudsim.Host@74a14482 FAILURE...---------->7

4.0 ---> Host org.cloudbus.cloudsim.Host@1540e19d FAILURE...---------->2

4.0 ---> Host org.cloudbus.cloudsim.Host@677327b6 FAILURE...---------->4

4.0 ---> Host org.cloudbus.cloudsim.Host@14ae5a5 FAILURE...---------->8

6.0 ---> Host org.cloudbus.cloudsim.Host@7f31245a FAILURE...---------->3

7.0 ---> Host org.cloudbus.cloudsim.Host@6d6f6e28 FAILURE...---------->6

9.0 ---> Host org.cloudbus.cloudsim.Host@135fbaa4 FAILURE...---------->0

9.0 ---> Host org.cloudbus.cloudsim.Host@45ee12a7 FAILURE...---------->5

10.0 ---> Host org.cloudbus.cloudsim.Host@330bedb4 FAILURE...---------->9

Simulation: No more future events

CloudInformationService: Notify all CloudSim entities for shutting down.

Datacenter_0 is shutting down...

HostFaultInjection0: is shutting down...

HostFaultInjection1: is shutting down...

HostFaultInjection2: is shutting down...

HostFaultInjection3: is shutting down...

HostFaultInjection4: is shutting down...

HostFaultInjection5: is shutting down...

HostFaultInjection6: is shutting down...

HostFaultInjection7: is shutting down...

HostFaultInjection8: is shutting down...

HostFaultInjection9: is shutting down...

Broker_0 is shutting down...

Simulation completed.

Simulation completed.

========== OUTPUT ==========

Cloudlet ID STATUS Data center ID VM ID Time Start Time Finish Time

========== OUTPUT ==========

0 [org.cloudbus.cloudsim.Vm@12a3a380, org.cloudbus.cloudsim.Vm@29453f44, org.cloudbus.cloudsim.Vm@5cad8086, org.cloudbus.cloudsim.Vm@6e0be858]

1 [org.cloudbus.cloudsim.Vm@61bbe9ba, org.cloudbus.cloudsim.Vm@610455d6]

2 [org.cloudbus.cloudsim.Vm@511d50c0, org.cloudbus.cloudsim.Vm@60e53b93, org.cloudbus.cloudsim.Vm@5e2de80c, org.cloudbus.cloudsim.Vm@1d44bcfa]

3 [org.cloudbus.cloudsim.Vm@266474c2, org.cloudbus.cloudsim.Vm@6f94fa3e]

4 [org.cloudbus.cloudsim.Vm@5e481248, org.cloudbus.cloudsim.Vm@66d3c617, org.cloudbus.cloudsim.Vm@63947c6b, org.cloudbus.cloudsim.Vm@2b193f2d]

5 [org.cloudbus.cloudsim.Vm@355da254, org.cloudbus.cloudsim.Vm@4dc63996]

6 [org.cloudbus.cloudsim.Vm@d716361, org.cloudbus.cloudsim.Vm@6ff3c5b5, org.cloudbus.cloudsim.Vm@3764951d, org.cloudbus.cloudsim.Vm@4b1210ee]

7 [org.cloudbus.cloudsim.Vm@4d7e1886, org.cloudbus.cloudsim.Vm@3cd1a2f1]

8 [org.cloudbus.cloudsim.Vm@2f0e140b, org.cloudbus.cloudsim.Vm@7440e464, org.cloudbus.cloudsim.Vm@49476842, org.cloudbus.cloudsim.Vm@78308db1]

9 [org.cloudbus.cloudsim.Vm@27c170f0, org.cloudbus.cloudsim.Vm@5451c3a8]

0 [org.cloudbus.cloudsim.Vm@12a3a380, org.cloudbus.cloudsim.Vm@29453f44, org.cloudbus.cloudsim.Vm@5cad8086, org.cloudbus.cloudsim.Vm@6e0be858]

1 [org.cloudbus.cloudsim.Vm@61bbe9ba, org.cloudbus.cloudsim.Vm@610455d6]

2 [org.cloudbus.cloudsim.Vm@511d50c0, org.cloudbus.cloudsim.Vm@60e53b93, org.cloudbus.cloudsim.Vm@5e2de80c, org.cloudbus.cloudsim.Vm@1d44bcfa]

3 [org.cloudbus.cloudsim.Vm@266474c2, org.cloudbus.cloudsim.Vm@6f94fa3e]

4 [org.cloudbus.cloudsim.Vm@5e481248, org.cloudbus.cloudsim.Vm@66d3c617, org.cloudbus.cloudsim.Vm@63947c6b, org.cloudbus.cloudsim.Vm@2b193f2d]

5 [org.cloudbus.cloudsim.Vm@355da254, org.cloudbus.cloudsim.Vm@4dc63996]

6 [org.cloudbus.cloudsim.Vm@d716361, org.cloudbus.cloudsim.Vm@6ff3c5b5, org.cloudbus.cloudsim.Vm@3764951d, org.cloudbus.cloudsim.Vm@4b1210ee]

7 [org.cloudbus.cloudsim.Vm@4d7e1886, org.cloudbus.cloudsim.Vm@3cd1a2f1]

8 [org.cloudbus.cloudsim.Vm@2f0e140b, org.cloudbus.cloudsim.Vm@7440e464, org.cloudbus.cloudsim.Vm@49476842, org.cloudbus.cloudsim.Vm@78308db1]

9 [org.cloudbus.cloudsim.Vm@27c170f0, org.cloudbus.cloudsim.Vm@5451c3a8]

Scenario with faultInjection finished!

[INFO] --

[INFO] BUILD SUCCESS

[INFO] --

[INFO] Total time: 1.521s

[INFO] Finished at: Tue Jun 20 16:37:41 EAT 2017

[INFO] Final Memory: 6M/123M

[INFO] --

Appendix C: Sample java code for simulator (scenario 1 simulation 2: node management)

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * Title:        CloudSim Toolkit
 * Description:  CloudSim (Cloud Simulation) Toolkit for Modeling and Simulation
```

```
 *              of Clouds
 * Licence:      GPL - http://www.gnu.org/copyleft/gpl.html
 *
 * Copyright (c) 2009, The University of Melbourne, Australia
 */

/** This simulation is for Scenario 1 Simulation 2 of Msagha J Mbogholi Thesis
 * entitled: A MODEL DRIVEN APPROACH TO RELATING AVAILABILITY MECHANISMS TO OUTAGE CAUSES IN CLOUD COMPUTING
 * Scenario is based on Node Management
 * In this scenario we have now allowed VM migration while also injecting random node failures:
 * The simulation run shows that while cloudlet were sent to the datacenter and random nodes failed
 * cloudlet execution still occurred.
 */
package org.cloudbus.cloudsim.examples;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter;
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
```

```
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.UtilizationModel;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerTimeShared;
import org.cloudbus.cloudsim.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.lists.HostList;
import org.cloudbus.cloudsim.lists.VmList;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;


/**
 * In this scenario we attempt to fail several hosts using 10 hosts, 5 users
 * and 30 VMs; 500 cloudlets are created and sent to the datacenter
 *
 * The scenario further adds an event that allows us to view the host IDs and the hosts themselves as 2 separate
outputs
 */
public class Scenario7 {
```

```
    /** The cloudlet list. */
    private static List<Cloudlet> cloudletList;

/** The host list */
private static List<Host> hostList;

/** The host list with hosts removed */
private static List<Host> newhostList;

    /** The vmList. */
    private static List<Vm> vmList;

    private static List<Vm> createVM(int userId, int vms, int idShift) {
            //Creates a container to store VMs. This list is passed to the broker later
            LinkedList<Vm> list = new LinkedList<Vm>();

            //VM Parameters
            long size = 10000; //image size (MB)
            int ram = 512; //vm memory (MB)
            int mips = 250;
            long bw = 1000;
            int pesNumber = 1; //number of cpus
            String vmm = "Xen"; //VMM name

            //create VMs
            Vm[] vm = new Vm[vms];
```

```
		for(int i=0;i<vms;i++){
			vm[i] = new Vm(idShift + i, userId, mips, pesNumber, ram, bw, size, vmm, new CloudletSchedulerTimeShared());
			list.add(vm[i]);
		}

		return list;
	}

	private static List<Cloudlet> createCloudlet(int userId, int cloudlets, int idShift){
		// Creates a container to store Cloudlets
		LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();

		//cloudlet parameters
		long length = 40000;
		long fileSize = 300;
		long outputSize = 300;
		int pesNumber = 1;
		UtilizationModel utilizationModel = new UtilizationModelFull();

		Cloudlet[] cloudlet = new Cloudlet[cloudlets];

		for(int i=0;i<cloudlets;i++){
			cloudlet[i] = new Cloudlet(idShift + i, length, pesNumber, fileSize, outputSize, utilizationModel, utilizationModel, utilizationModel);
			// setting the owner of these Cloudlets
			cloudlet[i].setUserId(userId);
			list.add(cloudlet[i]);
```

```
                }

                return list;
        }

//Create the host list

public <T extends Host> List<T> getHostList() {
                return (List<T>) hostList;




}


        /////////////////////////// STATIC METHODS ///////////////////////

        /**
         * Creates main() to run this scenario
         */
        public static void main(String[] args) {
                Log.printLine("Starting Scenario7...");

                try {
                        // First step: Initialize the CloudSim package. It should be called
                        // before creating any entities.
                        int num_user = 5;   // number of grid users
```

```
                Calendar calendar = Calendar.getInstance();
                boolean trace_flag = false;  // mean trace events

                // Initialize the CloudSim library
                CloudSim.init(num_user, calendar, trace_flag);



                    // Second step: Create Datacenters
                //Datacenters are the resource providers in CloudSim. We need at list one of them to run a CloudSim simulation
                @SuppressWarnings("unused")
                Datacenter datacenter0 = createDatacenter("Datacenter_0");

        for (Host host : datacenter0.getHostList()) {

                HostFaultInjection hostFaultInjection = new HostFaultInjection("HostFaultInjection" + host.getId());
        hostFaultInjection.setHost(host);



        }


                  //Use only one datacenter for this scenario

                //Third step: Create Broker
```

```
                DatacenterBroker broker = createBroker("Broker_0");
                int brokerId = broker.getId();

                //Fourth step: Create VMs and Cloudlets and send them to broker
                vmList = createVM(brokerId, 30, 0); //creating 50 vms
                cloudletList = createCloudlet(brokerId, 500, 0); // creating 500 cloudlets
//hostList created here

                broker.submitVmList(vmList);
                broker.submitCloudletList(cloudletList);


                // Fifth step: Starts the simulation
                CloudSim.startSimulation();



                // Final step: Print results when simulation is over
                List<Cloudlet> newList = broker.getCloudletReceivedList();

 List<Host> newList2 = datacenter0.getHostList();
newList2.addAll(datacenter0.getHostList());
```

```
        CloudSim.stopSimulation();


         printCloudletList(newList);
         printHostList(newList2);



                Log.printLine("Scenario with faultInjection finished!");
            }
            catch (Exception e)
            {
                e.printStackTrace();
                Log.printLine("The simulation has been terminated due to an unexpected error");
            }
}

private static Datacenter createDatacenter(String name){

            // Here are the steps needed to create a PowerDatacenter:
            // 1. We need to create a list to store one or more
            //    Machines
            ArrayList<Host> hostList = new ArrayList<Host>();

            // 2. A Machine contains one or more PEs or CPUs/Cores. Therefore, should
            //    create a list to store these PEs before creating
            //    a Machine.
            List<Pe> peList1 = new ArrayList<Pe>();
```

```
int mips = 1000;

// 3. Create PEs and add these into the list.
//for a quad-core machine, a list of 4 PEs is required:
peList1.add(new Pe(0, new PeProvisionerSimple(mips))); // need to store Pe id and MIPS Rating
peList1.add(new Pe(1, new PeProvisionerSimple(mips)));
peList1.add(new Pe(2, new PeProvisionerSimple(mips)));
peList1.add(new Pe(3, new PeProvisionerSimple(mips)));

//Another list, for a dual-core machine
List<Pe> peList2 = new ArrayList<Pe>();

peList2.add(new Pe(0, new PeProvisionerSimple(mips)));
peList2.add(new Pe(1, new PeProvisionerSimple(mips)));

//4. Create Hosts with its id and list of PEs and add them to the list of machines
int hostId=0;
int ram = 16384; //host memory (MB)
long storage = 1000000; //host storage
int bw = 10000;

hostList.add(
        new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
```

```
                storage,
                peList1,
                new VmSchedulerSpaceShared(peList1)
            )
        ); // This is our first machine

        hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerSpaceShared(peList2)
            )
        ); // Second machine

hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
```

```
                peList1,
                new VmSchedulerSpaceShared(peList1)
            )
        ); // Third machine

hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerSpaceShared(peList2)
            )
        ); // Fourth machine


hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
```

```
                peList1,
                new VmSchedulerSpaceShared(peList1)
            )
        ); // Fifth machine

hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList2,
                new VmSchedulerSpaceShared(peList2)
            )
        ); // Sixth machine

hostId++;

        hostList.add(
            new Host(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerSimple(bw),
                storage,
                peList1,
```

```
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Seventh machine


hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Eigth machine

hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList1,
```

```
                        new VmSchedulerSpaceShared(peList1)
                )
        ); // Ninth machine


hostId++;

        hostList.add(
                new Host(
                        hostId,
                        new RamProvisionerSimple(ram),
                        new BwProvisionerSimple(bw),
                        storage,
                        peList2,
                        new VmSchedulerSpaceShared(peList2)
                )
        ); // Tenth machine



        // 5. Create a DatacenterCharacteristics object that stores the
        //    properties of a data center: architecture, OS, list of
        //    Machines, allocation policy: time- or space-shared, time zone
        //    and its price (G$/Pe time unit).
        String arch = "x86";      // system architecture
        String os = "Linux";          // operating system
        String vmm = "Xen";
```

```
                double time_zone = 10.0;         // time zone this resource located
                double cost = 3.0;               // the cost of using processing in this resource
                double costPerMem = 0.05;                // the cost of using memory in this resource
                double costPerStorage = 0.1;           // the cost of using storage in this resource
                double costPerBw = 0.1;                  // the cost of using bw in this resource
                LinkedList<Storage> storageList = new LinkedList<Storage>();           //we are not adding SAN
devices by now

                DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
            arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);



                // 6. Finally, we need to create a PowerDatacenter object.
                Datacenter datacenter = null;
                try {
                        datacenter = new Datacenter(name, characteristics, new
VmAllocationPolicySimple(hostList), storageList, 0);
                } catch (Exception e) {
                        e.printStackTrace();
                }

                return datacenter;
        }

        //We strongly encourage users to develop their own broker policies, to submit vms and cloudlets according
        //to the specific rules of the simulated scenario
        private static DatacenterBroker createBroker(String name){
```

```
		DatacenterBroker broker = null;
		try {
			broker = new DatacenterBroker(name);
		} catch (Exception e) {
			e.printStackTrace();
			return null;
		}
	 return broker;
	}



	/**
	 * Prints the Cloudlet objects
	 * @param list  list of Cloudlets
	 */
	private static void printCloudletList(List<Cloudlet> list) {
		int size = list.size();
		Cloudlet cloudlet;

		String indent = "    ";
		Log.printLine();
		Log.printLine("========== OUTPUT ==========");
		Log.printLine("Cloudlet ID" + indent + "STATUS" + indent +
				"Data center ID" + indent + "VM ID" + indent + indent +  indent + "Time" + indent + "Start Time" + indent + "Finish Time");

		DecimalFormat dft = new DecimalFormat("###.##");
		for (int i = 0; i < size; i++) {
```

```
                cloudlet = list.get(i);
                Log.print(indent + cloudlet.getCloudletId() + indent + indent);
            //cloudlet execution was successful
                if (cloudlet.getCloudletStatus() == Cloudlet.SUCCESS){
                    Log.print("SUCCESS");

                    Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                            indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                            indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                }

            else if (cloudlet.getCloudletStatus() == Cloudlet.FAILED){
                    Log.print("FAILED!");

                    Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                            indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                            indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                }

            else if (cloudlet.getCloudletStatus() == Cloudlet.FAILED_RESOURCE_UNAVAILABLE){
                    Log.print("NO RESOURCDES AVAILABLE!");

                    Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
```

```
                                                    indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                    indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                        }

                else if (cloudlet.getCloudletStatus() == Cloudlet.INEXEC){
                                Log.print("CLOUDLETS IN EXECUTION!");

                                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                                    indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                    indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                        }

                else if (cloudlet.getCloudletStatus() == Cloudlet.CANCELED){
                                Log.printLine("CANCELLED!");

                                Log.printLine( indent + indent + cloudlet.getResourceId() + indent + indent +
indent + cloudlet.getVmId() +
                                                    indent + indent + indent +
dft.format(cloudlet.getActualCPUTime()) +
                                                    indent + indent + dft.format(cloudlet.getExecStartTime())+
indent + indent + indent + dft.format(cloudlet.getFinishTime()));
                        }

                else Log.print("Status of the cloudlet is unkown");
                    }
```

```
    }


//print the host list


private static void printHostList(List<Host> list) {
                int size = list.size();
                Host host;


                String indent = "    ";
                Log.printLine();
                Log.printLine("========== OUTPUT ==========");
                Log.printLine("Host ID" + indent + indent +  "VM Id" + indent);


                DecimalFormat dft = new DecimalFormat("###.##");
                for (int i = 0; i < size; i++) {
                        host = list.get(i);
            Log.printLine(host.getId() + indent + indent + host.getVmsMigratingIn());




                        }
                }


 public static class HostFaultInjection extends SimEntity {
```

```
                /**
 * 1 means that the host failure is true and 0 otherwise
 */
private static final int HOST_FAILURE = 1;
private LinkedList<Cloudlet> cloudletList;
private Host host;
private Vm vm;
private Cloudlet cloudlet;
private DatacenterBroker broker;
private Datacenter datacenter;




                public HostFaultInjection(String name) {
                        super(name);
                }


                 @Override
public void startEntity() {
   int delay = delayRandomly(2800);
```

```
    Log.printLine(getName() + " is starting...");
    schedule(getId(), delay, HOST_FAILURE);



}

@Override
public void processEvent(SimEvent ev) {

    switch (ev.getTag()) {
        case HOST_FAILURE:
            host.setFailed(true); // set to true

            if (host.isFailed()) {
                Log.printLine(CloudSim.clock() + " ---> Host " + host + " FAILURE..." + "---------->" + host.getId());
                Log.printLine(host.getVmList());
                Log.printLine();

                for (Vm vm : host.getVmList()) {

                  vm.setInMigration(true);
                  this.host.reallocateMigratingInVms();
                  this.host.updateVmsProcessing(CloudSim.clock());
```

```
        }

    }

    break;

default:
    Log.printLine(getName() + ": unknown event type");
    break;
```

```
    }

}

@Override
public void shutdownEntity() {
    Log.printLine(getName() + ": is shutting down...");
}

/**
 * The value of the delay will be generated within that range (0 -
 * MAX_TIME_SIMULATION).
 *
 * @param max_simulation represents the max time for simulation.
 * @return
 */
public int delayRandomly(int max_simulation) {
    return 1 + (int) (Math.random() * max_simulation);
}

/**
 * @return the host
 */
public Host getHost() {
    return host;
}
```

```
    /**
     * @param host the host to set
     */
    public void setHost(Host host) {
        this.host = host;
    }
}


}
```

Appendix D: Sample output from simulator (simulation 1 scenario 1: node management)

```
cd D:\Msagha D on Nettop\PhD\cloudsim-3.0.3; "JAVA_HOME=C:\\Program Files\\Java\\jdk1.8.0_20" cmd /c
"\"\"C:\\Program Files\\NetBeans 8.1\\java\\maven\\bin\\mvn.bat\" -Dexec.args=\"-classpath %classpath
```

org.cloudbus.cloudsim.examples.Scenario7\" -Dexec.executable=java -Dexec.classpathScope=runtime -Dmaven.ext.class.path=\"C:\\Program Files\\NetBeans 8.1\\java\\maven-nblib\\netbeans-eventspy.jar\" -Dfile.encoding=UTF-8 org.codehaus.mojo:exec-maven-plugin:1.2.1:exec\""

Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Compile on Save turned on) will be used instead of their jar artifacts.

[INFO] Scanning for projects...

[INFO]

[INFO] --

[INFO] Building cloudsim-toolkit 2.1

[INFO] --

[INFO]

[INFO] --- exec-maven-plugin:1.2.1:exec (default-cli) @ cloudsim-toolkit ---

Starting Scenario7...

Initialising...

Starting CloudSim version 3.0

Datacenter_0 is starting...

HostFaultInjection0 is starting...

HostFaultInjection1 is starting...

HostFaultInjection2 is starting...

HostFaultInjection3 is starting...

HostFaultInjection4 is starting...

HostFaultInjection5 is starting...

HostFaultInjection6 is starting...

HostFaultInjection7 is starting...

HostFaultInjection8 is starting...

HostFaultInjection9 is starting...

Broker_0 is starting...

Entities started.

0.0: Broker_0: Cloud Resource List received with 1 resource(s)

0.0: Broker_0: Trying to Create VM #0 in Datacenter_0

0.0: Broker_0: Trying to Create VM #1 in Datacenter_0

0.0: Broker_0: Trying to Create VM #2 in Datacenter_0

0.0: Broker_0: Trying to Create VM #3 in Datacenter_0

0.0: Broker_0: Trying to Create VM #4 in Datacenter_0

0.0: Broker_0: Trying to Create VM #5 in Datacenter_0

0.0: Broker_0: Trying to Create VM #6 in Datacenter_0

0.0: Broker_0: Trying to Create VM #7 in Datacenter_0

0.0: Broker_0: Trying to Create VM #8 in Datacenter_0

0.0: Broker_0: Trying to Create VM #9 in Datacenter_0

0.0: Broker_0: Trying to Create VM #10 in Datacenter_0

0.0: Broker_0: Trying to Create VM #11 in Datacenter_0

0.0: Broker_0: Trying to Create VM #12 in Datacenter_0

0.0: Broker_0: Trying to Create VM #13 in Datacenter_0

0.0: Broker_0: Trying to Create VM #14 in Datacenter_0

0.0: Broker_0: Trying to Create VM #15 in Datacenter_0

0.0: Broker_0: Trying to Create VM #16 in Datacenter_0

0.0: Broker_0: Trying to Create VM #17 in Datacenter_0

0.0: Broker_0: Trying to Create VM #18 in Datacenter_0

0.0: Broker_0: Trying to Create VM #19 in Datacenter_0

0.0: Broker_0: Trying to Create VM #20 in Datacenter_0

0.0: Broker_0: Trying to Create VM #21 in Datacenter_0

0.0: Broker_0: Trying to Create VM #22 in Datacenter_0

0.0: Broker_0: Trying to Create VM #23 in Datacenter_0

0.0: Broker_0: Trying to Create VM #24 in Datacenter_0

0.0: Broker_0: Trying to Create VM #25 in Datacenter_0

0.0: Broker_0: Trying to Create VM #26 in Datacenter_0

0.0: Broker_0: Trying to Create VM #27 in Datacenter_0

0.0: Broker_0: Trying to Create VM #28 in Datacenter_0

0.0: Broker_0: Trying to Create VM #29 in Datacenter_0

0.1: Broker_0: VM #0 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #1 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #2 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #3 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #4 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #5 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #6 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #7 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #8 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #9 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #10 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #11 has been created in Datacenter #2, Host #1

0.1: Broker_0: VM #12 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #13 has been created in Datacenter #2, Host #3

0.1: Broker_0: VM #14 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #15 has been created in Datacenter #2, Host #5

0.1: Broker_0: VM #16 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #17 has been created in Datacenter #2, Host #7

0.1: Broker_0: VM #18 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #19 has been created in Datacenter #2, Host #9

0.1: Broker_0: VM #20 has been created in Datacenter #2, Host #0

0.1: Broker_0: VM #21 has been created in Datacenter #2, Host #1

0.1: Broker_0: VM #22 has been created in Datacenter #2, Host #2

0.1: Broker_0: VM #23 has been created in Datacenter #2, Host #3

0.1: Broker_0: VM #24 has been created in Datacenter #2, Host #4

0.1: Broker_0: VM #25 has been created in Datacenter #2, Host #5

0.1: Broker_0: VM #26 has been created in Datacenter #2, Host #6

0.1: Broker_0: VM #27 has been created in Datacenter #2, Host #7

0.1: Broker_0: VM #28 has been created in Datacenter #2, Host #8

0.1: Broker_0: VM #29 has been created in Datacenter #2, Host #9

0.1: Broker_0: Sending cloudlet 0 to VM #0

0.1: Broker_0: Sending cloudlet 1 to VM #1

0.1: Broker_0: Sending cloudlet 2 to VM #2

0.1: Broker_0: Sending cloudlet 3 to VM #3

0.1: Broker_0: Sending cloudlet 4 to VM #4
0.1: Broker_0: Sending cloudlet 5 to VM #5
0.1: Broker_0: Sending cloudlet 6 to VM #6
0.1: Broker_0: Sending cloudlet 7 to VM #7
0.1: Broker_0: Sending cloudlet 8 to VM #8
0.1: Broker_0: Sending cloudlet 9 to VM #9
0.1: Broker_0: Sending cloudlet 10 to VM #10
0.1: Broker_0: Sending cloudlet 11 to VM #11
0.1: Broker_0: Sending cloudlet 12 to VM #12
0.1: Broker_0: Sending cloudlet 13 to VM #13
0.1: Broker_0: Sending cloudlet 14 to VM #14
0.1: Broker_0: Sending cloudlet 15 to VM #15
0.1: Broker_0: Sending cloudlet 16 to VM #16
0.1: Broker_0: Sending cloudlet 17 to VM #17
0.1: Broker_0: Sending cloudlet 18 to VM #18
0.1: Broker_0: Sending cloudlet 19 to VM #19
0.1: Broker_0: Sending cloudlet 20 to VM #20
0.1: Broker_0: Sending cloudlet 21 to VM #21
0.1: Broker_0: Sending cloudlet 22 to VM #22
0.1: Broker_0: Sending cloudlet 23 to VM #23
0.1: Broker_0: Sending cloudlet 24 to VM #24
0.1: Broker_0: Sending cloudlet 25 to VM #25
0.1: Broker_0: Sending cloudlet 26 to VM #26
0.1: Broker_0: Sending cloudlet 27 to VM #27
0.1: Broker_0: Sending cloudlet 28 to VM #28
0.1: Broker_0: Sending cloudlet 29 to VM #29
0.1: Broker_0: Sending cloudlet 30 to VM #0
0.1: Broker_0: Sending cloudlet 31 to VM #1
0.1: Broker_0: Sending cloudlet 32 to VM #2
0.1: Broker_0: Sending cloudlet 33 to VM #3
0.1: Broker_0: Sending cloudlet 34 to VM #4
0.1: Broker_0: Sending cloudlet 35 to VM #5
0.1: Broker_0: Sending cloudlet 36 to VM #6
0.1: Broker_0: Sending cloudlet 37 to VM #7
0.1: Broker_0: Sending cloudlet 38 to VM #8
0.1: Broker_0: Sending cloudlet 39 to VM #9
0.1: Broker_0: Sending cloudlet 40 to VM #10
0.1: Broker_0: Sending cloudlet 41 to VM #11
0.1: Broker_0: Sending cloudlet 42 to VM #12
0.1: Broker_0: Sending cloudlet 43 to VM #13
0.1: Broker_0: Sending cloudlet 44 to VM #14
0.1: Broker_0: Sending cloudlet 45 to VM #15
0.1: Broker_0: Sending cloudlet 46 to VM #16
0.1: Broker_0: Sending cloudlet 47 to VM #17
0.1: Broker_0: Sending cloudlet 48 to VM #18
0.1: Broker_0: Sending cloudlet 49 to VM #19
0.1: Broker_0: Sending cloudlet 50 to VM #20
0.1: Broker_0: Sending cloudlet 51 to VM #21
0.1: Broker_0: Sending cloudlet 52 to VM #22
0.1: Broker_0: Sending cloudlet 53 to VM #23
0.1: Broker_0: Sending cloudlet 54 to VM #24
0.1: Broker_0: Sending cloudlet 55 to VM #25
0.1: Broker_0: Sending cloudlet 56 to VM #26
0.1: Broker_0: Sending cloudlet 57 to VM #27

0.1: Broker_0: Sending cloudlet 58 to VM #28
0.1: Broker_0: Sending cloudlet 59 to VM #29
0.1: Broker_0: Sending cloudlet 60 to VM #0
0.1: Broker_0: Sending cloudlet 61 to VM #1
0.1: Broker_0: Sending cloudlet 62 to VM #2
0.1: Broker_0: Sending cloudlet 63 to VM #3
0.1: Broker_0: Sending cloudlet 64 to VM #4
0.1: Broker_0: Sending cloudlet 65 to VM #5
0.1: Broker_0: Sending cloudlet 66 to VM #6
0.1: Broker_0: Sending cloudlet 67 to VM #7
0.1: Broker_0: Sending cloudlet 68 to VM #8
0.1: Broker_0: Sending cloudlet 69 to VM #9
0.1: Broker_0: Sending cloudlet 70 to VM #10
0.1: Broker_0: Sending cloudlet 71 to VM #11
0.1: Broker_0: Sending cloudlet 72 to VM #12
0.1: Broker_0: Sending cloudlet 73 to VM #13
0.1: Broker_0: Sending cloudlet 74 to VM #14
0.1: Broker_0: Sending cloudlet 75 to VM #15
0.1: Broker_0: Sending cloudlet 76 to VM #16
0.1: Broker_0: Sending cloudlet 77 to VM #17
0.1: Broker_0: Sending cloudlet 78 to VM #18
0.1: Broker_0: Sending cloudlet 79 to VM #19
0.1: Broker_0: Sending cloudlet 80 to VM #20
0.1: Broker_0: Sending cloudlet 81 to VM #21
0.1: Broker_0: Sending cloudlet 82 to VM #22
0.1: Broker_0: Sending cloudlet 83 to VM #23
0.1: Broker_0: Sending cloudlet 84 to VM #24
0.1: Broker_0: Sending cloudlet 85 to VM #25
0.1: Broker_0: Sending cloudlet 86 to VM #26
0.1: Broker_0: Sending cloudlet 87 to VM #27
0.1: Broker_0: Sending cloudlet 88 to VM #28
0.1: Broker_0: Sending cloudlet 89 to VM #29
0.1: Broker_0: Sending cloudlet 90 to VM #0
0.1: Broker_0: Sending cloudlet 91 to VM #1
0.1: Broker_0: Sending cloudlet 92 to VM #2
0.1: Broker_0: Sending cloudlet 93 to VM #3
0.1: Broker_0: Sending cloudlet 94 to VM #4
0.1: Broker_0: Sending cloudlet 95 to VM #5
0.1: Broker_0: Sending cloudlet 96 to VM #6
0.1: Broker_0: Sending cloudlet 97 to VM #7
0.1: Broker_0: Sending cloudlet 98 to VM #8
0.1: Broker_0: Sending cloudlet 99 to VM #9
0.1: Broker_0: Sending cloudlet 100 to VM #10
0.1: Broker_0: Sending cloudlet 101 to VM #11
0.1: Broker_0: Sending cloudlet 102 to VM #12
0.1: Broker_0: Sending cloudlet 103 to VM #13
0.1: Broker_0: Sending cloudlet 104 to VM #14
0.1: Broker_0: Sending cloudlet 105 to VM #15
0.1: Broker_0: Sending cloudlet 106 to VM #16
0.1: Broker_0: Sending cloudlet 107 to VM #17
0.1: Broker_0: Sending cloudlet 108 to VM #18
0.1: Broker_0: Sending cloudlet 109 to VM #19
0.1: Broker_0: Sending cloudlet 110 to VM #20
0.1: Broker_0: Sending cloudlet 111 to VM #21

0.1: Broker_0: Sending cloudlet 112 to VM #22
0.1: Broker_0: Sending cloudlet 113 to VM #23
0.1: Broker_0: Sending cloudlet 114 to VM #24
0.1: Broker_0: Sending cloudlet 115 to VM #25
0.1: Broker_0: Sending cloudlet 116 to VM #26
0.1: Broker_0: Sending cloudlet 117 to VM #27
0.1: Broker_0: Sending cloudlet 118 to VM #28
0.1: Broker_0: Sending cloudlet 119 to VM #29
0.1: Broker_0: Sending cloudlet 120 to VM #0
0.1: Broker_0: Sending cloudlet 121 to VM #1
0.1: Broker_0: Sending cloudlet 122 to VM #2
0.1: Broker_0: Sending cloudlet 123 to VM #3
0.1: Broker_0: Sending cloudlet 124 to VM #4
0.1: Broker_0: Sending cloudlet 125 to VM #5
0.1: Broker_0: Sending cloudlet 126 to VM #6
0.1: Broker_0: Sending cloudlet 127 to VM #7
0.1: Broker_0: Sending cloudlet 128 to VM #8
0.1: Broker_0: Sending cloudlet 129 to VM #9
0.1: Broker_0: Sending cloudlet 130 to VM #10
0.1: Broker_0: Sending cloudlet 131 to VM #11
0.1: Broker_0: Sending cloudlet 132 to VM #12
0.1: Broker_0: Sending cloudlet 133 to VM #13
0.1: Broker_0: Sending cloudlet 134 to VM #14
0.1: Broker_0: Sending cloudlet 135 to VM #15
0.1: Broker_0: Sending cloudlet 136 to VM #16
0.1: Broker_0: Sending cloudlet 137 to VM #17
0.1: Broker_0: Sending cloudlet 138 to VM #18
0.1: Broker_0: Sending cloudlet 139 to VM #19
0.1: Broker_0: Sending cloudlet 140 to VM #20
0.1: Broker_0: Sending cloudlet 141 to VM #21
0.1: Broker_0: Sending cloudlet 142 to VM #22
0.1: Broker_0: Sending cloudlet 143 to VM #23
0.1: Broker_0: Sending cloudlet 144 to VM #24
0.1: Broker_0: Sending cloudlet 145 to VM #25
0.1: Broker_0: Sending cloudlet 146 to VM #26
0.1: Broker_0: Sending cloudlet 147 to VM #27
0.1: Broker_0: Sending cloudlet 148 to VM #28
0.1: Broker_0: Sending cloudlet 149 to VM #29
0.1: Broker_0: Sending cloudlet 150 to VM #0
0.1: Broker_0: Sending cloudlet 151 to VM #1
0.1: Broker_0: Sending cloudlet 152 to VM #2
0.1: Broker_0: Sending cloudlet 153 to VM #3
0.1: Broker_0: Sending cloudlet 154 to VM #4
0.1: Broker_0: Sending cloudlet 155 to VM #5
0.1: Broker_0: Sending cloudlet 156 to VM #6
0.1: Broker_0: Sending cloudlet 157 to VM #7
0.1: Broker_0: Sending cloudlet 158 to VM #8
0.1: Broker_0: Sending cloudlet 159 to VM #9
0.1: Broker_0: Sending cloudlet 160 to VM #10
0.1: Broker_0: Sending cloudlet 161 to VM #11
0.1: Broker_0: Sending cloudlet 162 to VM #12
0.1: Broker_0: Sending cloudlet 163 to VM #13
0.1: Broker_0: Sending cloudlet 164 to VM #14
0.1: Broker_0: Sending cloudlet 165 to VM #15

0.1: Broker_0: Sending cloudlet 166 to VM #16
0.1: Broker_0: Sending cloudlet 167 to VM #17
0.1: Broker_0: Sending cloudlet 168 to VM #18
0.1: Broker_0: Sending cloudlet 169 to VM #19
0.1: Broker_0: Sending cloudlet 170 to VM #20
0.1: Broker_0: Sending cloudlet 171 to VM #21
0.1: Broker_0: Sending cloudlet 172 to VM #22
0.1: Broker_0: Sending cloudlet 173 to VM #23
0.1: Broker_0: Sending cloudlet 174 to VM #24
0.1: Broker_0: Sending cloudlet 175 to VM #25
0.1: Broker_0: Sending cloudlet 176 to VM #26
0.1: Broker_0: Sending cloudlet 177 to VM #27
0.1: Broker_0: Sending cloudlet 178 to VM #28
0.1: Broker_0: Sending cloudlet 179 to VM #29
0.1: Broker_0: Sending cloudlet 180 to VM #0
0.1: Broker_0: Sending cloudlet 181 to VM #1
0.1: Broker_0: Sending cloudlet 182 to VM #2
0.1: Broker_0: Sending cloudlet 183 to VM #3
0.1: Broker_0: Sending cloudlet 184 to VM #4
0.1: Broker_0: Sending cloudlet 185 to VM #5
0.1: Broker_0: Sending cloudlet 186 to VM #6
0.1: Broker_0: Sending cloudlet 187 to VM #7
0.1: Broker_0: Sending cloudlet 188 to VM #8
0.1: Broker_0: Sending cloudlet 189 to VM #9
0.1: Broker_0: Sending cloudlet 190 to VM #10
0.1: Broker_0: Sending cloudlet 191 to VM #11
0.1: Broker_0: Sending cloudlet 192 to VM #12
0.1: Broker_0: Sending cloudlet 193 to VM #13
0.1: Broker_0: Sending cloudlet 194 to VM #14
0.1: Broker_0: Sending cloudlet 195 to VM #15
0.1: Broker_0: Sending cloudlet 196 to VM #16
0.1: Broker_0: Sending cloudlet 197 to VM #17
0.1: Broker_0: Sending cloudlet 198 to VM #18
0.1: Broker_0: Sending cloudlet 199 to VM #19
0.1: Broker_0: Sending cloudlet 200 to VM #20
0.1: Broker_0: Sending cloudlet 201 to VM #21
0.1: Broker_0: Sending cloudlet 202 to VM #22
0.1: Broker_0: Sending cloudlet 203 to VM #23
0.1: Broker_0: Sending cloudlet 204 to VM #24
0.1: Broker_0: Sending cloudlet 205 to VM #25
0.1: Broker_0: Sending cloudlet 206 to VM #26
0.1: Broker_0: Sending cloudlet 207 to VM #27
0.1: Broker_0: Sending cloudlet 208 to VM #28
0.1: Broker_0: Sending cloudlet 209 to VM #29
0.1: Broker_0: Sending cloudlet 210 to VM #0
0.1: Broker_0: Sending cloudlet 211 to VM #1
0.1: Broker_0: Sending cloudlet 212 to VM #2
0.1: Broker_0: Sending cloudlet 213 to VM #3
0.1: Broker_0: Sending cloudlet 214 to VM #4
0.1: Broker_0: Sending cloudlet 215 to VM #5
0.1: Broker_0: Sending cloudlet 216 to VM #6
0.1: Broker_0: Sending cloudlet 217 to VM #7
0.1: Broker_0: Sending cloudlet 218 to VM #8
0.1: Broker_0: Sending cloudlet 219 to VM #9

0.1: Broker_0: Sending cloudlet 220 to VM #10
0.1: Broker_0: Sending cloudlet 221 to VM #11
0.1: Broker_0: Sending cloudlet 222 to VM #12
0.1: Broker_0: Sending cloudlet 223 to VM #13
0.1: Broker_0: Sending cloudlet 224 to VM #14
0.1: Broker_0: Sending cloudlet 225 to VM #15
0.1: Broker_0: Sending cloudlet 226 to VM #16
0.1: Broker_0: Sending cloudlet 227 to VM #17
0.1: Broker_0: Sending cloudlet 228 to VM #18
0.1: Broker_0: Sending cloudlet 229 to VM #19
0.1: Broker_0: Sending cloudlet 230 to VM #20
0.1: Broker_0: Sending cloudlet 231 to VM #21
0.1: Broker_0: Sending cloudlet 232 to VM #22
0.1: Broker_0: Sending cloudlet 233 to VM #23
0.1: Broker_0: Sending cloudlet 234 to VM #24
0.1: Broker_0: Sending cloudlet 235 to VM #25
0.1: Broker_0: Sending cloudlet 236 to VM #26
0.1: Broker_0: Sending cloudlet 237 to VM #27
0.1: Broker_0: Sending cloudlet 238 to VM #28
0.1: Broker_0: Sending cloudlet 239 to VM #29
0.1: Broker_0: Sending cloudlet 240 to VM #0
0.1: Broker_0: Sending cloudlet 241 to VM #1
0.1: Broker_0: Sending cloudlet 242 to VM #2
0.1: Broker_0: Sending cloudlet 243 to VM #3
0.1: Broker_0: Sending cloudlet 244 to VM #4
0.1: Broker_0: Sending cloudlet 245 to VM #5
0.1: Broker_0: Sending cloudlet 246 to VM #6
0.1: Broker_0: Sending cloudlet 247 to VM #7
0.1: Broker_0: Sending cloudlet 248 to VM #8
0.1: Broker_0: Sending cloudlet 249 to VM #9
0.1: Broker_0: Sending cloudlet 250 to VM #10
0.1: Broker_0: Sending cloudlet 251 to VM #11
0.1: Broker_0: Sending cloudlet 252 to VM #12
0.1: Broker_0: Sending cloudlet 253 to VM #13
0.1: Broker_0: Sending cloudlet 254 to VM #14
0.1: Broker_0: Sending cloudlet 255 to VM #15
0.1: Broker_0: Sending cloudlet 256 to VM #16
0.1: Broker_0: Sending cloudlet 257 to VM #17
0.1: Broker_0: Sending cloudlet 258 to VM #18
0.1: Broker_0: Sending cloudlet 259 to VM #19
0.1: Broker_0: Sending cloudlet 260 to VM #20
0.1: Broker_0: Sending cloudlet 261 to VM #21
0.1: Broker_0: Sending cloudlet 262 to VM #22
0.1: Broker_0: Sending cloudlet 263 to VM #23
0.1: Broker_0: Sending cloudlet 264 to VM #24
0.1: Broker_0: Sending cloudlet 265 to VM #25
0.1: Broker_0: Sending cloudlet 266 to VM #26
0.1: Broker_0: Sending cloudlet 267 to VM #27
0.1: Broker_0: Sending cloudlet 268 to VM #28
0.1: Broker_0: Sending cloudlet 269 to VM #29
0.1: Broker_0: Sending cloudlet 270 to VM #0
0.1: Broker_0: Sending cloudlet 271 to VM #1
0.1: Broker_0: Sending cloudlet 272 to VM #2
0.1: Broker_0: Sending cloudlet 273 to VM #3

0.1: Broker_0: Sending cloudlet 274 to VM #4
0.1: Broker_0: Sending cloudlet 275 to VM #5
0.1: Broker_0: Sending cloudlet 276 to VM #6
0.1: Broker_0: Sending cloudlet 277 to VM #7
0.1: Broker_0: Sending cloudlet 278 to VM #8
0.1: Broker_0: Sending cloudlet 279 to VM #9
0.1: Broker_0: Sending cloudlet 280 to VM #10
0.1: Broker_0: Sending cloudlet 281 to VM #11
0.1: Broker_0: Sending cloudlet 282 to VM #12
0.1: Broker_0: Sending cloudlet 283 to VM #13
0.1: Broker_0: Sending cloudlet 284 to VM #14
0.1: Broker_0: Sending cloudlet 285 to VM #15
0.1: Broker_0: Sending cloudlet 286 to VM #16
0.1: Broker_0: Sending cloudlet 287 to VM #17
0.1: Broker_0: Sending cloudlet 288 to VM #18
0.1: Broker_0: Sending cloudlet 289 to VM #19
0.1: Broker_0: Sending cloudlet 290 to VM #20
0.1: Broker_0: Sending cloudlet 291 to VM #21
0.1: Broker_0: Sending cloudlet 292 to VM #22
0.1: Broker_0: Sending cloudlet 293 to VM #23
0.1: Broker_0: Sending cloudlet 294 to VM #24
0.1: Broker_0: Sending cloudlet 295 to VM #25
0.1: Broker_0: Sending cloudlet 296 to VM #26
0.1: Broker_0: Sending cloudlet 297 to VM #27
0.1: Broker_0: Sending cloudlet 298 to VM #28
0.1: Broker_0: Sending cloudlet 299 to VM #29
0.1: Broker_0: Sending cloudlet 300 to VM #0
0.1: Broker_0: Sending cloudlet 301 to VM #1
0.1: Broker_0: Sending cloudlet 302 to VM #2
0.1: Broker_0: Sending cloudlet 303 to VM #3
0.1: Broker_0: Sending cloudlet 304 to VM #4
0.1: Broker_0: Sending cloudlet 305 to VM #5
0.1: Broker_0: Sending cloudlet 306 to VM #6
0.1: Broker_0: Sending cloudlet 307 to VM #7
0.1: Broker_0: Sending cloudlet 308 to VM #8
0.1: Broker_0: Sending cloudlet 309 to VM #9
0.1: Broker_0: Sending cloudlet 310 to VM #10
0.1: Broker_0: Sending cloudlet 311 to VM #11
0.1: Broker_0: Sending cloudlet 312 to VM #12
0.1: Broker_0: Sending cloudlet 313 to VM #13
0.1: Broker_0: Sending cloudlet 314 to VM #14
0.1: Broker_0: Sending cloudlet 315 to VM #15
0.1: Broker_0: Sending cloudlet 316 to VM #16
0.1: Broker_0: Sending cloudlet 317 to VM #17
0.1: Broker_0: Sending cloudlet 318 to VM #18
0.1: Broker_0: Sending cloudlet 319 to VM #19
0.1: Broker_0: Sending cloudlet 320 to VM #20
0.1: Broker_0: Sending cloudlet 321 to VM #21
0.1: Broker_0: Sending cloudlet 322 to VM #22
0.1: Broker_0: Sending cloudlet 323 to VM #23
0.1: Broker_0: Sending cloudlet 324 to VM #24
0.1: Broker_0: Sending cloudlet 325 to VM #25
0.1: Broker_0: Sending cloudlet 326 to VM #26
0.1: Broker_0: Sending cloudlet 327 to VM #27

0.1: Broker_0: Sending cloudlet 328 to VM #28
0.1: Broker_0: Sending cloudlet 329 to VM #29
0.1: Broker_0: Sending cloudlet 330 to VM #0
0.1: Broker_0: Sending cloudlet 331 to VM #1
0.1: Broker_0: Sending cloudlet 332 to VM #2
0.1: Broker_0: Sending cloudlet 333 to VM #3
0.1: Broker_0: Sending cloudlet 334 to VM #4
0.1: Broker_0: Sending cloudlet 335 to VM #5
0.1: Broker_0: Sending cloudlet 336 to VM #6
0.1: Broker_0: Sending cloudlet 337 to VM #7
0.1: Broker_0: Sending cloudlet 338 to VM #8
0.1: Broker_0: Sending cloudlet 339 to VM #9
0.1: Broker_0: Sending cloudlet 340 to VM #10
0.1: Broker_0: Sending cloudlet 341 to VM #11
0.1: Broker_0: Sending cloudlet 342 to VM #12
0.1: Broker_0: Sending cloudlet 343 to VM #13
0.1: Broker_0: Sending cloudlet 344 to VM #14
0.1: Broker_0: Sending cloudlet 345 to VM #15
0.1: Broker_0: Sending cloudlet 346 to VM #16
0.1: Broker_0: Sending cloudlet 347 to VM #17
0.1: Broker_0: Sending cloudlet 348 to VM #18
0.1: Broker_0: Sending cloudlet 349 to VM #19
0.1: Broker_0: Sending cloudlet 350 to VM #20
0.1: Broker_0: Sending cloudlet 351 to VM #21
0.1: Broker_0: Sending cloudlet 352 to VM #22
0.1: Broker_0: Sending cloudlet 353 to VM #23
0.1: Broker_0: Sending cloudlet 354 to VM #24
0.1: Broker_0: Sending cloudlet 355 to VM #25
0.1: Broker_0: Sending cloudlet 356 to VM #26
0.1: Broker_0: Sending cloudlet 357 to VM #27
0.1: Broker_0: Sending cloudlet 358 to VM #28
0.1: Broker_0: Sending cloudlet 359 to VM #29
0.1: Broker_0: Sending cloudlet 360 to VM #0
0.1: Broker_0: Sending cloudlet 361 to VM #1
0.1: Broker_0: Sending cloudlet 362 to VM #2
0.1: Broker_0: Sending cloudlet 363 to VM #3
0.1: Broker_0: Sending cloudlet 364 to VM #4
0.1: Broker_0: Sending cloudlet 365 to VM #5
0.1: Broker_0: Sending cloudlet 366 to VM #6
0.1: Broker_0: Sending cloudlet 367 to VM #7
0.1: Broker_0: Sending cloudlet 368 to VM #8
0.1: Broker_0: Sending cloudlet 369 to VM #9
0.1: Broker_0: Sending cloudlet 370 to VM #10
0.1: Broker_0: Sending cloudlet 371 to VM #11
0.1: Broker_0: Sending cloudlet 372 to VM #12
0.1: Broker_0: Sending cloudlet 373 to VM #13
0.1: Broker_0: Sending cloudlet 374 to VM #14
0.1: Broker_0: Sending cloudlet 375 to VM #15
0.1: Broker_0: Sending cloudlet 376 to VM #16
0.1: Broker_0: Sending cloudlet 377 to VM #17
0.1: Broker_0: Sending cloudlet 378 to VM #18
0.1: Broker_0: Sending cloudlet 379 to VM #19
0.1: Broker_0: Sending cloudlet 380 to VM #20
0.1: Broker_0: Sending cloudlet 381 to VM #21

0.1: Broker_0: Sending cloudlet 382 to VM #22
0.1: Broker_0: Sending cloudlet 383 to VM #23
0.1: Broker_0: Sending cloudlet 384 to VM #24
0.1: Broker_0: Sending cloudlet 385 to VM #25
0.1: Broker_0: Sending cloudlet 386 to VM #26
0.1: Broker_0: Sending cloudlet 387 to VM #27
0.1: Broker_0: Sending cloudlet 388 to VM #28
0.1: Broker_0: Sending cloudlet 389 to VM #29
0.1: Broker_0: Sending cloudlet 390 to VM #0
0.1: Broker_0: Sending cloudlet 391 to VM #1
0.1: Broker_0: Sending cloudlet 392 to VM #2
0.1: Broker_0: Sending cloudlet 393 to VM #3
0.1: Broker_0: Sending cloudlet 394 to VM #4
0.1: Broker_0: Sending cloudlet 395 to VM #5
0.1: Broker_0: Sending cloudlet 396 to VM #6
0.1: Broker_0: Sending cloudlet 397 to VM #7
0.1: Broker_0: Sending cloudlet 398 to VM #8
0.1: Broker_0: Sending cloudlet 399 to VM #9
0.1: Broker_0: Sending cloudlet 400 to VM #10
0.1: Broker_0: Sending cloudlet 401 to VM #11
0.1: Broker_0: Sending cloudlet 402 to VM #12
0.1: Broker_0: Sending cloudlet 403 to VM #13
0.1: Broker_0: Sending cloudlet 404 to VM #14
0.1: Broker_0: Sending cloudlet 405 to VM #15
0.1: Broker_0: Sending cloudlet 406 to VM #16
0.1: Broker_0: Sending cloudlet 407 to VM #17
0.1: Broker_0: Sending cloudlet 408 to VM #18
0.1: Broker_0: Sending cloudlet 409 to VM #19
0.1: Broker_0: Sending cloudlet 410 to VM #20
0.1: Broker_0: Sending cloudlet 411 to VM #21
0.1: Broker_0: Sending cloudlet 412 to VM #22
0.1: Broker_0: Sending cloudlet 413 to VM #23
0.1: Broker_0: Sending cloudlet 414 to VM #24
0.1: Broker_0: Sending cloudlet 415 to VM #25
0.1: Broker_0: Sending cloudlet 416 to VM #26
0.1: Broker_0: Sending cloudlet 417 to VM #27
0.1: Broker_0: Sending cloudlet 418 to VM #28
0.1: Broker_0: Sending cloudlet 419 to VM #29
0.1: Broker_0: Sending cloudlet 420 to VM #0
0.1: Broker_0: Sending cloudlet 421 to VM #1
0.1: Broker_0: Sending cloudlet 422 to VM #2
0.1: Broker_0: Sending cloudlet 423 to VM #3
0.1: Broker_0: Sending cloudlet 424 to VM #4
0.1: Broker_0: Sending cloudlet 425 to VM #5
0.1: Broker_0: Sending cloudlet 426 to VM #6
0.1: Broker_0: Sending cloudlet 427 to VM #7
0.1: Broker_0: Sending cloudlet 428 to VM #8
0.1: Broker_0: Sending cloudlet 429 to VM #9
0.1: Broker_0: Sending cloudlet 430 to VM #10
0.1: Broker_0: Sending cloudlet 431 to VM #11
0.1: Broker_0: Sending cloudlet 432 to VM #12
0.1: Broker_0: Sending cloudlet 433 to VM #13
0.1: Broker_0: Sending cloudlet 434 to VM #14
0.1: Broker_0: Sending cloudlet 435 to VM #15

0.1: Broker_0: Sending cloudlet 436 to VM #16
0.1: Broker_0: Sending cloudlet 437 to VM #17
0.1: Broker_0: Sending cloudlet 438 to VM #18
0.1: Broker_0: Sending cloudlet 439 to VM #19
0.1: Broker_0: Sending cloudlet 440 to VM #20
0.1: Broker_0: Sending cloudlet 441 to VM #21
0.1: Broker_0: Sending cloudlet 442 to VM #22
0.1: Broker_0: Sending cloudlet 443 to VM #23
0.1: Broker_0: Sending cloudlet 444 to VM #24
0.1: Broker_0: Sending cloudlet 445 to VM #25
0.1: Broker_0: Sending cloudlet 446 to VM #26
0.1: Broker_0: Sending cloudlet 447 to VM #27
0.1: Broker_0: Sending cloudlet 448 to VM #28
0.1: Broker_0: Sending cloudlet 449 to VM #29
0.1: Broker_0: Sending cloudlet 450 to VM #0
0.1: Broker_0: Sending cloudlet 451 to VM #1
0.1: Broker_0: Sending cloudlet 452 to VM #2
0.1: Broker_0: Sending cloudlet 453 to VM #3
0.1: Broker_0: Sending cloudlet 454 to VM #4
0.1: Broker_0: Sending cloudlet 455 to VM #5
0.1: Broker_0: Sending cloudlet 456 to VM #6
0.1: Broker_0: Sending cloudlet 457 to VM #7
0.1: Broker_0: Sending cloudlet 458 to VM #8
0.1: Broker_0: Sending cloudlet 459 to VM #9
0.1: Broker_0: Sending cloudlet 460 to VM #10
0.1: Broker_0: Sending cloudlet 461 to VM #11
0.1: Broker_0: Sending cloudlet 462 to VM #12
0.1: Broker_0: Sending cloudlet 463 to VM #13
0.1: Broker_0: Sending cloudlet 464 to VM #14
0.1: Broker_0: Sending cloudlet 465 to VM #15
0.1: Broker_0: Sending cloudlet 466 to VM #16
0.1: Broker_0: Sending cloudlet 467 to VM #17
0.1: Broker_0: Sending cloudlet 468 to VM #18
0.1: Broker_0: Sending cloudlet 469 to VM #19
0.1: Broker_0: Sending cloudlet 470 to VM #20
0.1: Broker_0: Sending cloudlet 471 to VM #21
0.1: Broker_0: Sending cloudlet 472 to VM #22
0.1: Broker_0: Sending cloudlet 473 to VM #23
0.1: Broker_0: Sending cloudlet 474 to VM #24
0.1: Broker_0: Sending cloudlet 475 to VM #25
0.1: Broker_0: Sending cloudlet 476 to VM #26
0.1: Broker_0: Sending cloudlet 477 to VM #27
0.1: Broker_0: Sending cloudlet 478 to VM #28
0.1: Broker_0: Sending cloudlet 479 to VM #29
0.1: Broker_0: Sending cloudlet 480 to VM #0
0.1: Broker_0: Sending cloudlet 481 to VM #1
0.1: Broker_0: Sending cloudlet 482 to VM #2
0.1: Broker_0: Sending cloudlet 483 to VM #3
0.1: Broker_0: Sending cloudlet 484 to VM #4
0.1: Broker_0: Sending cloudlet 485 to VM #5
0.1: Broker_0: Sending cloudlet 486 to VM #6
0.1: Broker_0: Sending cloudlet 487 to VM #7
0.1: Broker_0: Sending cloudlet 488 to VM #8
0.1: Broker_0: Sending cloudlet 489 to VM #9

0.1: Broker_0: Sending cloudlet 490 to VM #10

0.1: Broker_0: Sending cloudlet 491 to VM #11

0.1: Broker_0: Sending cloudlet 492 to VM #12

0.1: Broker_0: Sending cloudlet 493 to VM #13

0.1: Broker_0: Sending cloudlet 494 to VM #14

0.1: Broker_0: Sending cloudlet 495 to VM #15

0.1: Broker_0: Sending cloudlet 496 to VM #16

0.1: Broker_0: Sending cloudlet 497 to VM #17

0.1: Broker_0: Sending cloudlet 498 to VM #18

0.1: Broker_0: Sending cloudlet 499 to VM #19

19.0 ---> Host org.cloudbus.cloudsim.Host@4554617c FAILURE...---------->8

[org.cloudbus.cloudsim.Vm@74a14482, org.cloudbus.cloudsim.Vm@1540e19d, org.cloudbus.cloudsim.Vm@677327b6, org.cloudbus.cloudsim.Vm@14ae5a5]

238.0 ---> Host org.cloudbus.cloudsim.Host@7f31245a FAILURE...---------->7

[org.cloudbus.cloudsim.Vm@6d6f6e28, org.cloudbus.cloudsim.Vm@135fbaa4]

747.0 ---> Host org.cloudbus.cloudsim.Host@45ee12a7 FAILURE...---------->6

[org.cloudbus.cloudsim.Vm@330bedb4, org.cloudbus.cloudsim.Vm@2503dbd3, org.cloudbus.cloudsim.Vm@4b67cf4d, org.cloudbus.cloudsim.Vm@7ea987ac]

1285.0 ---> Host org.cloudbus.cloudsim.Host@12a3a380 FAILURE...---------->3

[org.cloudbus.cloudsim.Vm@29453f44, org.cloudbus.cloudsim.Vm@5cad8086]

1532.0 ---> Host org.cloudbus.cloudsim.Host@6e0be858 FAILURE...---------->9

[org.cloudbus.cloudsim.Vm@61bbe9ba, org.cloudbus.cloudsim.Vm@610455d6]

1770.0 ---> Host org.cloudbus.cloudsim.Host@511d50c0 FAILURE...---------->4

[org.cloudbus.cloudsim.Vm@60e53b93, org.cloudbus.cloudsim.Vm@5e2de80c, org.cloudbus.cloudsim.Vm@1d44bcfa, org.cloudbus.cloudsim.Vm@266474c2]

1862.0 ---> Host org.cloudbus.cloudsim.Host@6f94fa3e FAILURE...---------->1

[org.cloudbus.cloudsim.Vm@5e481248, org.cloudbus.cloudsim.Vm@66d3c617]

1963.0 ---> Host org.cloudbus.cloudsim.Host@63947c6b FAILURE...---------->5

[org.cloudbus.cloudsim.Vm@2b193f2d, org.cloudbus.cloudsim.Vm@355da254]

2307.0 ---> Host
org.cloudbus.cloudsim.Host@4dc63996
FAILURE...---------->0

[org.cloudbus.cloudsim.Vm@d716361,
org.cloudbus.cloudsim.Vm@6ff3c5b5,
org.cloudbus.cloudsim.Vm@3764951d,
org.cloudbus.cloudsim.Vm@4b1210ee]

2344.0 ---> Host
org.cloudbus.cloudsim.Host@4d7e1886
FAILURE...---------->2

[org.cloudbus.cloudsim.Vm@3cd1a2f1,
org.cloudbus.cloudsim.Vm@2f0e140b,
org.cloudbus.cloudsim.Vm@7440e464,
org.cloudbus.cloudsim.Vm@49476842]

2560.1: Broker_0: Cloudlet 20 received

2560.1: Broker_0: Cloudlet 50 received

2560.1: Broker_0: Cloudlet 80 received

2560.1: Broker_0: Cloudlet 110 received

2560.1: Broker_0: Cloudlet 140 received

2560.1: Broker_0: Cloudlet 170 received

2560.1: Broker_0: Cloudlet 200 received

2560.1: Broker_0: Cloudlet 230 received

2560.1: Broker_0: Cloudlet 260 received

2560.1: Broker_0: Cloudlet 290 received

2560.1: Broker_0: Cloudlet 320 received

2560.1: Broker_0: Cloudlet 350 received

2560.1: Broker_0: Cloudlet 380 received

2560.1: Broker_0: Cloudlet 410 received

2560.1: Broker_0: Cloudlet 440 received

2560.1: Broker_0: Cloudlet 470 received

2560.1: Broker_0: Cloudlet 21 received

2560.1: Broker_0: Cloudlet 51 received

2560.1: Broker_0: Cloudlet 81 received

2560.1: Broker_0: Cloudlet 111 received

2560.1: Broker_0: Cloudlet 141 received

2560.1: Broker_0: Cloudlet 171 received

2560.1: Broker_0: Cloudlet 201 received

2560.1: Broker_0: Cloudlet 231 received

2560.1: Broker_0: Cloudlet 261 received

2560.1: Broker_0: Cloudlet 291 received

2560.1: Broker_0: Cloudlet 321 received

2560.1: Broker_0: Cloudlet 351 received

2560.1: Broker_0: Cloudlet 381 received

2560.1: Broker_0: Cloudlet 411 received

2560.1: Broker_0: Cloudlet 441 received

2560.1: Broker_0: Cloudlet 471 received

2560.1: Broker_0: Cloudlet 22 received

2560.1: Broker_0: Cloudlet 52 received

2560.1: Broker_0: Cloudlet 82 received

2560.1: Broker_0: Cloudlet 112 received

2560.1: Broker_0: Cloudlet 142 received

2560.1: Broker_0: Cloudlet 172 received

2560.1: Broker_0: Cloudlet 202 received
2560.1: Broker_0: Cloudlet 232 received
2560.1: Broker_0: Cloudlet 262 received
2560.1: Broker_0: Cloudlet 292 received
2560.1: Broker_0: Cloudlet 322 received
2560.1: Broker_0: Cloudlet 352 received
2560.1: Broker_0: Cloudlet 382 received
2560.1: Broker_0: Cloudlet 412 received
2560.1: Broker_0: Cloudlet 442 received
2560.1: Broker_0: Cloudlet 472 received
2560.1: Broker_0: Cloudlet 23 received
2560.1: Broker_0: Cloudlet 53 received
2560.1: Broker_0: Cloudlet 83 received
2560.1: Broker_0: Cloudlet 113 received
2560.1: Broker_0: Cloudlet 143 received
2560.1: Broker_0: Cloudlet 173 received
2560.1: Broker_0: Cloudlet 203 received
2560.1: Broker_0: Cloudlet 233 received
2560.1: Broker_0: Cloudlet 263 received
2560.1: Broker_0: Cloudlet 293 received
2560.1: Broker_0: Cloudlet 323 received
2560.1: Broker_0: Cloudlet 353 received
2560.1: Broker_0: Cloudlet 383 received
2560.1: Broker_0: Cloudlet 413 received
2560.1: Broker_0: Cloudlet 443 received
2560.1: Broker_0: Cloudlet 473 received
2560.1: Broker_0: Cloudlet 24 received
2560.1: Broker_0: Cloudlet 54 received
2560.1: Broker_0: Cloudlet 84 received
2560.1: Broker_0: Cloudlet 114 received
2560.1: Broker_0: Cloudlet 144 received
2560.1: Broker_0: Cloudlet 174 received
2560.1: Broker_0: Cloudlet 204 received
2560.1: Broker_0: Cloudlet 234 received
2560.1: Broker_0: Cloudlet 264 received
2560.1: Broker_0: Cloudlet 294 received
2560.1: Broker_0: Cloudlet 324 received
2560.1: Broker_0: Cloudlet 354 received
2560.1: Broker_0: Cloudlet 384 received
2560.1: Broker_0: Cloudlet 414 received
2560.1: Broker_0: Cloudlet 444 received
2560.1: Broker_0: Cloudlet 474 received
2560.1: Broker_0: Cloudlet 25 received
2560.1: Broker_0: Cloudlet 55 received
2560.1: Broker_0: Cloudlet 85 received
2560.1: Broker_0: Cloudlet 115 received
2560.1: Broker_0: Cloudlet 145 received
2560.1: Broker_0: Cloudlet 175 received
2560.1: Broker_0: Cloudlet 205 received
2560.1: Broker_0: Cloudlet 235 received
2560.1: Broker_0: Cloudlet 265 received
2560.1: Broker_0: Cloudlet 295 received
2560.1: Broker_0: Cloudlet 325 received
2560.1: Broker_0: Cloudlet 355 received

2560.1: Broker_0: Cloudlet 385 received
2560.1: Broker_0: Cloudlet 415 received
2560.1: Broker_0: Cloudlet 445 received
2560.1: Broker_0: Cloudlet 475 received
2560.1: Broker_0: Cloudlet 26 received
2560.1: Broker_0: Cloudlet 56 received
2560.1: Broker_0: Cloudlet 86 received
2560.1: Broker_0: Cloudlet 116 received
2560.1: Broker_0: Cloudlet 146 received
2560.1: Broker_0: Cloudlet 176 received
2560.1: Broker_0: Cloudlet 206 received
2560.1: Broker_0: Cloudlet 236 received
2560.1: Broker_0: Cloudlet 266 received
2560.1: Broker_0: Cloudlet 296 received
2560.1: Broker_0: Cloudlet 326 received
2560.1: Broker_0: Cloudlet 356 received
2560.1: Broker_0: Cloudlet 386 received
2560.1: Broker_0: Cloudlet 416 received
2560.1: Broker_0: Cloudlet 446 received
2560.1: Broker_0: Cloudlet 476 received
2560.1: Broker_0: Cloudlet 27 received
2560.1: Broker_0: Cloudlet 57 received
2560.1: Broker_0: Cloudlet 87 received
2560.1: Broker_0: Cloudlet 117 received
2560.1: Broker_0: Cloudlet 147 received
2560.1: Broker_0: Cloudlet 177 received
2560.1: Broker_0: Cloudlet 207 received
2560.1: Broker_0: Cloudlet 237 received
2560.1: Broker_0: Cloudlet 267 received
2560.1: Broker_0: Cloudlet 297 received
2560.1: Broker_0: Cloudlet 327 received
2560.1: Broker_0: Cloudlet 357 received
2560.1: Broker_0: Cloudlet 387 received
2560.1: Broker_0: Cloudlet 417 received
2560.1: Broker_0: Cloudlet 447 received
2560.1: Broker_0: Cloudlet 477 received
2560.1: Broker_0: Cloudlet 28 received
2560.1: Broker_0: Cloudlet 58 received
2560.1: Broker_0: Cloudlet 88 received
2560.1: Broker_0: Cloudlet 118 received
2560.1: Broker_0: Cloudlet 148 received
2560.1: Broker_0: Cloudlet 178 received
2560.1: Broker_0: Cloudlet 208 received
2560.1: Broker_0: Cloudlet 238 received
2560.1: Broker_0: Cloudlet 268 received
2560.1: Broker_0: Cloudlet 298 received
2560.1: Broker_0: Cloudlet 328 received
2560.1: Broker_0: Cloudlet 358 received
2560.1: Broker_0: Cloudlet 388 received
2560.1: Broker_0: Cloudlet 418 received
2560.1: Broker_0: Cloudlet 448 received
2560.1: Broker_0: Cloudlet 478 received
2560.1: Broker_0: Cloudlet 29 received
2560.1: Broker_0: Cloudlet 59 received

2560.1: Broker_0: Cloudlet 89 received
2560.1: Broker_0: Cloudlet 119 received
2560.1: Broker_0: Cloudlet 149 received
2560.1: Broker_0: Cloudlet 179 received
2560.1: Broker_0: Cloudlet 209 received
2560.1: Broker_0: Cloudlet 239 received
2560.1: Broker_0: Cloudlet 269 received
2560.1: Broker_0: Cloudlet 299 received
2560.1: Broker_0: Cloudlet 329 received
2560.1: Broker_0: Cloudlet 359 received
2560.1: Broker_0: Cloudlet 389 received
2560.1: Broker_0: Cloudlet 419 received
2560.1: Broker_0: Cloudlet 449 received
2560.1: Broker_0: Cloudlet 479 received
2720.036: Broker_0: Cloudlet 0 received
2720.036: Broker_0: Cloudlet 30 received
2720.036: Broker_0: Cloudlet 60 received
2720.036: Broker_0: Cloudlet 90 received
2720.036: Broker_0: Cloudlet 120 received
2720.036: Broker_0: Cloudlet 150 received
2720.036: Broker_0: Cloudlet 180 received
2720.036: Broker_0: Cloudlet 210 received
2720.036: Broker_0: Cloudlet 240 received
2720.036: Broker_0: Cloudlet 270 received
2720.036: Broker_0: Cloudlet 300 received
2720.036: Broker_0: Cloudlet 330 received
2720.036: Broker_0: Cloudlet 360 received
2720.036: Broker_0: Cloudlet 390 received
2720.036: Broker_0: Cloudlet 420 received
2720.036: Broker_0: Cloudlet 450 received
2720.036: Broker_0: Cloudlet 480 received
2720.036: Broker_0: Cloudlet 5 received
2720.036: Broker_0: Cloudlet 35 received
2720.036: Broker_0: Cloudlet 65 received
2720.036: Broker_0: Cloudlet 95 received
2720.036: Broker_0: Cloudlet 125 received
2720.036: Broker_0: Cloudlet 155 received
2720.036: Broker_0: Cloudlet 185 received
2720.036: Broker_0: Cloudlet 215 received
2720.036: Broker_0: Cloudlet 245 received
2720.036: Broker_0: Cloudlet 275 received
2720.036: Broker_0: Cloudlet 305 received
2720.036: Broker_0: Cloudlet 335 received
2720.036: Broker_0: Cloudlet 365 received
2720.036: Broker_0: Cloudlet 395 received
2720.036: Broker_0: Cloudlet 425 received
2720.036: Broker_0: Cloudlet 455 received
2720.036: Broker_0: Cloudlet 485 received
2720.036: Broker_0: Cloudlet 10 received
2720.036: Broker_0: Cloudlet 40 received
2720.036: Broker_0: Cloudlet 70 received
2720.036: Broker_0: Cloudlet 100 received
2720.036: Broker_0: Cloudlet 130 received
2720.036: Broker_0: Cloudlet 160 received

2720.036: Broker_0: Cloudlet 190 received
2720.036: Broker_0: Cloudlet 220 received
2720.036: Broker_0: Cloudlet 250 received
2720.036: Broker_0: Cloudlet 280 received
2720.036: Broker_0: Cloudlet 310 received
2720.036: Broker_0: Cloudlet 340 received
2720.036: Broker_0: Cloudlet 370 received
2720.036: Broker_0: Cloudlet 400 received
2720.036: Broker_0: Cloudlet 430 received
2720.036: Broker_0: Cloudlet 460 received
2720.036: Broker_0: Cloudlet 490 received
2720.036: Broker_0: Cloudlet 11 received
2720.036: Broker_0: Cloudlet 41 received
2720.036: Broker_0: Cloudlet 71 received
2720.036: Broker_0: Cloudlet 101 received
2720.036: Broker_0: Cloudlet 131 received
2720.036: Broker_0: Cloudlet 161 received
2720.036: Broker_0: Cloudlet 191 received
2720.036: Broker_0: Cloudlet 221 received
2720.036: Broker_0: Cloudlet 251 received
2720.036: Broker_0: Cloudlet 281 received
2720.036: Broker_0: Cloudlet 311 received
2720.036: Broker_0: Cloudlet 341 received
2720.036: Broker_0: Cloudlet 371 received
2720.036: Broker_0: Cloudlet 401 received
2720.036: Broker_0: Cloudlet 431 received
2720.036: Broker_0: Cloudlet 461 received
2720.036: Broker_0: Cloudlet 491 received
2720.036: Broker_0: Cloudlet 1 received
2720.036: Broker_0: Cloudlet 31 received
2720.036: Broker_0: Cloudlet 61 received
2720.036: Broker_0: Cloudlet 91 received
2720.036: Broker_0: Cloudlet 121 received
2720.036: Broker_0: Cloudlet 151 received
2720.036: Broker_0: Cloudlet 181 received
2720.036: Broker_0: Cloudlet 211 received
2720.036: Broker_0: Cloudlet 241 received
2720.036: Broker_0: Cloudlet 271 received
2720.036: Broker_0: Cloudlet 301 received
2720.036: Broker_0: Cloudlet 331 received
2720.036: Broker_0: Cloudlet 361 received
2720.036: Broker_0: Cloudlet 391 received
2720.036: Broker_0: Cloudlet 421 received
2720.036: Broker_0: Cloudlet 451 received
2720.036: Broker_0: Cloudlet 481 received
2720.036: Broker_0: Cloudlet 6 received
2720.036: Broker_0: Cloudlet 36 received
2720.036: Broker_0: Cloudlet 66 received
2720.036: Broker_0: Cloudlet 96 received
2720.036: Broker_0: Cloudlet 126 received
2720.036: Broker_0: Cloudlet 156 received
2720.036: Broker_0: Cloudlet 186 received
2720.036: Broker_0: Cloudlet 216 received
2720.036: Broker_0: Cloudlet 246 received

2720.036: Broker_0: Cloudlet 276 received
2720.036: Broker_0: Cloudlet 306 received
2720.036: Broker_0: Cloudlet 336 received
2720.036: Broker_0: Cloudlet 366 received
2720.036: Broker_0: Cloudlet 396 received
2720.036: Broker_0: Cloudlet 426 received
2720.036: Broker_0: Cloudlet 456 received
2720.036: Broker_0: Cloudlet 486 received
2720.036: Broker_0: Cloudlet 12 received
2720.036: Broker_0: Cloudlet 42 received
2720.036: Broker_0: Cloudlet 72 received
2720.036: Broker_0: Cloudlet 102 received
2720.036: Broker_0: Cloudlet 132 received
2720.036: Broker_0: Cloudlet 162 received
2720.036: Broker_0: Cloudlet 192 received
2720.036: Broker_0: Cloudlet 222 received
2720.036: Broker_0: Cloudlet 252 received
2720.036: Broker_0: Cloudlet 282 received
2720.036: Broker_0: Cloudlet 312 received
2720.036: Broker_0: Cloudlet 342 received
2720.036: Broker_0: Cloudlet 372 received
2720.036: Broker_0: Cloudlet 402 received
2720.036: Broker_0: Cloudlet 432 received
2720.036: Broker_0: Cloudlet 462 received
2720.036: Broker_0: Cloudlet 492 received
2720.036: Broker_0: Cloudlet 13 received
2720.036: Broker_0: Cloudlet 43 received
2720.036: Broker_0: Cloudlet 73 received
2720.036: Broker_0: Cloudlet 103 received
2720.036: Broker_0: Cloudlet 133 received
2720.036: Broker_0: Cloudlet 163 received
2720.036: Broker_0: Cloudlet 193 received
2720.036: Broker_0: Cloudlet 223 received
2720.036: Broker_0: Cloudlet 253 received
2720.036: Broker_0: Cloudlet 283 received
2720.036: Broker_0: Cloudlet 313 received
2720.036: Broker_0: Cloudlet 343 received
2720.036: Broker_0: Cloudlet 373 received
2720.036: Broker_0: Cloudlet 403 received
2720.036: Broker_0: Cloudlet 433 received
2720.036: Broker_0: Cloudlet 463 received
2720.036: Broker_0: Cloudlet 493 received
2720.036: Broker_0: Cloudlet 2 received
2720.036: Broker_0: Cloudlet 32 received
2720.036: Broker_0: Cloudlet 62 received
2720.036: Broker_0: Cloudlet 92 received
2720.036: Broker_0: Cloudlet 122 received
2720.036: Broker_0: Cloudlet 152 received
2720.036: Broker_0: Cloudlet 182 received
2720.036: Broker_0: Cloudlet 212 received
2720.036: Broker_0: Cloudlet 242 received
2720.036: Broker_0: Cloudlet 272 received
2720.036: Broker_0: Cloudlet 302 received
2720.036: Broker_0: Cloudlet 332 received

2720.036: Broker_0: Cloudlet 362 received
2720.036: Broker_0: Cloudlet 392 received
2720.036: Broker_0: Cloudlet 422 received
2720.036: Broker_0: Cloudlet 452 received
2720.036: Broker_0: Cloudlet 482 received
2720.036: Broker_0: Cloudlet 7 received
2720.036: Broker_0: Cloudlet 37 received
2720.036: Broker_0: Cloudlet 67 received
2720.036: Broker_0: Cloudlet 97 received
2720.036: Broker_0: Cloudlet 127 received
2720.036: Broker_0: Cloudlet 157 received
2720.036: Broker_0: Cloudlet 187 received
2720.036: Broker_0: Cloudlet 217 received
2720.036: Broker_0: Cloudlet 247 received
2720.036: Broker_0: Cloudlet 277 received
2720.036: Broker_0: Cloudlet 307 received
2720.036: Broker_0: Cloudlet 337 received
2720.036: Broker_0: Cloudlet 367 received
2720.036: Broker_0: Cloudlet 397 received
2720.036: Broker_0: Cloudlet 427 received
2720.036: Broker_0: Cloudlet 457 received
2720.036: Broker_0: Cloudlet 487 received
2720.036: Broker_0: Cloudlet 14 received
2720.036: Broker_0: Cloudlet 44 received
2720.036: Broker_0: Cloudlet 74 received
2720.036: Broker_0: Cloudlet 104 received
2720.036: Broker_0: Cloudlet 134 received
2720.036: Broker_0: Cloudlet 164 received
2720.036: Broker_0: Cloudlet 194 received
2720.036: Broker_0: Cloudlet 224 received
2720.036: Broker_0: Cloudlet 254 received
2720.036: Broker_0: Cloudlet 284 received
2720.036: Broker_0: Cloudlet 314 received
2720.036: Broker_0: Cloudlet 344 received
2720.036: Broker_0: Cloudlet 374 received
2720.036: Broker_0: Cloudlet 404 received
2720.036: Broker_0: Cloudlet 434 received
2720.036: Broker_0: Cloudlet 464 received
2720.036: Broker_0: Cloudlet 494 received
2720.036: Broker_0: Cloudlet 15 received
2720.036: Broker_0: Cloudlet 45 received
2720.036: Broker_0: Cloudlet 75 received
2720.036: Broker_0: Cloudlet 105 received
2720.036: Broker_0: Cloudlet 135 received
2720.036: Broker_0: Cloudlet 165 received
2720.036: Broker_0: Cloudlet 195 received
2720.036: Broker_0: Cloudlet 225 received
2720.036: Broker_0: Cloudlet 255 received
2720.036: Broker_0: Cloudlet 285 received
2720.036: Broker_0: Cloudlet 315 received
2720.036: Broker_0: Cloudlet 345 received
2720.036: Broker_0: Cloudlet 375 received
2720.036: Broker_0: Cloudlet 405 received
2720.036: Broker_0: Cloudlet 435 received

2720.036: Broker_0: Cloudlet 465 received
2720.036: Broker_0: Cloudlet 495 received
2720.036: Broker_0: Cloudlet 3 received
2720.036: Broker_0: Cloudlet 33 received
2720.036: Broker_0: Cloudlet 63 received
2720.036: Broker_0: Cloudlet 93 received
2720.036: Broker_0: Cloudlet 123 received
2720.036: Broker_0: Cloudlet 153 received
2720.036: Broker_0: Cloudlet 183 received
2720.036: Broker_0: Cloudlet 213 received
2720.036: Broker_0: Cloudlet 243 received
2720.036: Broker_0: Cloudlet 273 received
2720.036: Broker_0: Cloudlet 303 received
2720.036: Broker_0: Cloudlet 333 received
2720.036: Broker_0: Cloudlet 363 received
2720.036: Broker_0: Cloudlet 393 received
2720.036: Broker_0: Cloudlet 423 received
2720.036: Broker_0: Cloudlet 453 received
2720.036: Broker_0: Cloudlet 483 received
2720.036: Broker_0: Cloudlet 8 received
2720.036: Broker_0: Cloudlet 38 received
2720.036: Broker_0: Cloudlet 68 received
2720.036: Broker_0: Cloudlet 98 received
2720.036: Broker_0: Cloudlet 128 received
2720.036: Broker_0: Cloudlet 158 received
2720.036: Broker_0: Cloudlet 188 received
2720.036: Broker_0: Cloudlet 218 received
2720.036: Broker_0: Cloudlet 248 received
2720.036: Broker_0: Cloudlet 278 received
2720.036: Broker_0: Cloudlet 308 received
2720.036: Broker_0: Cloudlet 338 received
2720.036: Broker_0: Cloudlet 368 received
2720.036: Broker_0: Cloudlet 398 received
2720.036: Broker_0: Cloudlet 428 received
2720.036: Broker_0: Cloudlet 458 received
2720.036: Broker_0: Cloudlet 488 received
2720.036: Broker_0: Cloudlet 16 received
2720.036: Broker_0: Cloudlet 46 received
2720.036: Broker_0: Cloudlet 76 received
2720.036: Broker_0: Cloudlet 106 received
2720.036: Broker_0: Cloudlet 136 received
2720.036: Broker_0: Cloudlet 166 received
2720.036: Broker_0: Cloudlet 196 received
2720.036: Broker_0: Cloudlet 226 received
2720.036: Broker_0: Cloudlet 256 received
2720.036: Broker_0: Cloudlet 286 received
2720.036: Broker_0: Cloudlet 316 received
2720.036: Broker_0: Cloudlet 346 received
2720.036: Broker_0: Cloudlet 376 received
2720.036: Broker_0: Cloudlet 406 received
2720.036: Broker_0: Cloudlet 436 received
2720.036: Broker_0: Cloudlet 466 received
2720.036: Broker_0: Cloudlet 496 received
2720.036: Broker_0: Cloudlet 17 received

2720.036: Broker_0: Cloudlet 47 received
2720.036: Broker_0: Cloudlet 77 received
2720.036: Broker_0: Cloudlet 107 received
2720.036: Broker_0: Cloudlet 137 received
2720.036: Broker_0: Cloudlet 167 received
2720.036: Broker_0: Cloudlet 197 received
2720.036: Broker_0: Cloudlet 227 received
2720.036: Broker_0: Cloudlet 257 received
2720.036: Broker_0: Cloudlet 287 received
2720.036: Broker_0: Cloudlet 317 received
2720.036: Broker_0: Cloudlet 347 received
2720.036: Broker_0: Cloudlet 377 received
2720.036: Broker_0: Cloudlet 407 received
2720.036: Broker_0: Cloudlet 437 received
2720.036: Broker_0: Cloudlet 467 received
2720.036: Broker_0: Cloudlet 497 received
2720.036: Broker_0: Cloudlet 4 received
2720.036: Broker_0: Cloudlet 34 received
2720.036: Broker_0: Cloudlet 64 received
2720.036: Broker_0: Cloudlet 94 received
2720.036: Broker_0: Cloudlet 124 received
2720.036: Broker_0: Cloudlet 154 received
2720.036: Broker_0: Cloudlet 184 received
2720.036: Broker_0: Cloudlet 214 received
2720.036: Broker_0: Cloudlet 244 received
2720.036: Broker_0: Cloudlet 274 received
2720.036: Broker_0: Cloudlet 304 received
2720.036: Broker_0: Cloudlet 334 received
2720.036: Broker_0: Cloudlet 364 received
2720.036: Broker_0: Cloudlet 394 received
2720.036: Broker_0: Cloudlet 424 received
2720.036: Broker_0: Cloudlet 454 received
2720.036: Broker_0: Cloudlet 484 received
2720.036: Broker_0: Cloudlet 9 received
2720.036: Broker_0: Cloudlet 39 received
2720.036: Broker_0: Cloudlet 69 received
2720.036: Broker_0: Cloudlet 99 received
2720.036: Broker_0: Cloudlet 129 received
2720.036: Broker_0: Cloudlet 159 received
2720.036: Broker_0: Cloudlet 189 received
2720.036: Broker_0: Cloudlet 219 received
2720.036: Broker_0: Cloudlet 249 received
2720.036: Broker_0: Cloudlet 279 received
2720.036: Broker_0: Cloudlet 309 received
2720.036: Broker_0: Cloudlet 339 received
2720.036: Broker_0: Cloudlet 369 received
2720.036: Broker_0: Cloudlet 399 received
2720.036: Broker_0: Cloudlet 429 received
2720.036: Broker_0: Cloudlet 459 received
2720.036: Broker_0: Cloudlet 489 received
2720.036: Broker_0: Cloudlet 18 received
2720.036: Broker_0: Cloudlet 48 received
2720.036: Broker_0: Cloudlet 78 received
2720.036: Broker_0: Cloudlet 108 received

2720.036: Broker_0: Cloudlet 138 received

2720.036: Broker_0: Cloudlet 168 received

2720.036: Broker_0: Cloudlet 198 received

2720.036: Broker_0: Cloudlet 228 received

2720.036: Broker_0: Cloudlet 258 received

2720.036: Broker_0: Cloudlet 288 received

2720.036: Broker_0: Cloudlet 318 received

2720.036: Broker_0: Cloudlet 348 received

2720.036: Broker_0: Cloudlet 378 received

2720.036: Broker_0: Cloudlet 408 received

2720.036: Broker_0: Cloudlet 438 received

2720.036: Broker_0: Cloudlet 468 received

2720.036: Broker_0: Cloudlet 498 received

2720.036: Broker_0: Cloudlet 19 received

2720.036: Broker_0: Cloudlet 49 received

2720.036: Broker_0: Cloudlet 79 received

2720.036: Broker_0: Cloudlet 109 received

2720.036: Broker_0: Cloudlet 139 received

2720.036: Broker_0: Cloudlet 169 received

2720.036: Broker_0: Cloudlet 199 received

2720.036: Broker_0: Cloudlet 229 received

2720.036: Broker_0: Cloudlet 259 received

2720.036: Broker_0: Cloudlet 289 received

2720.036: Broker_0: Cloudlet 319 received

2720.036: Broker_0: Cloudlet 349 received

2720.036: Broker_0: Cloudlet 379 received

2720.036: Broker_0: Cloudlet 409 received

2720.036: Broker_0: Cloudlet 439 received

2720.036: Broker_0: Cloudlet 469 received

2720.036: Broker_0: Cloudlet 499 received

2720.036: Broker_0: All Cloudlets executed. Finishing...

2720.036: Broker_0: Destroying VM #0

2720.036: Broker_0: Destroying VM #1

2720.036: Broker_0: Destroying VM #2

2720.036: Broker_0: Destroying VM #3

2720.036: Broker_0: Destroying VM #4

2720.036: Broker_0: Destroying VM #5

2720.036: Broker_0: Destroying VM #6

2720.036: Broker_0: Destroying VM #7

2720.036: Broker_0: Destroying VM #8

2720.036: Broker_0: Destroying VM #9

2720.036: Broker_0: Destroying VM #10

2720.036: Broker_0: Destroying VM #11

2720.036: Broker_0: Destroying VM #12

2720.036: Broker_0: Destroying VM #13

2720.036: Broker_0: Destroying VM #14

2720.036: Broker_0: Destroying VM #15

2720.036: Broker_0: Destroying VM #16

2720.036: Broker_0: Destroying VM #17

2720.036: Broker_0: Destroying VM #18

2720.036: Broker_0: Destroying VM #19

2720.036: Broker_0: Destroying VM #20

2720.036: Broker_0: Destroying VM #21

2720.036: Broker_0: Destroying VM #22

2720.036: Broker_0: Destroying VM #23

2720.036: Broker_0: Destroying VM #24

2720.036: Broker_0: Destroying VM #25

2720.036: Broker_0: Destroying VM #26

2720.036: Broker_0: Destroying VM #27

2720.036: Broker_0: Destroying VM #28

2720.036: Broker_0: Destroying VM #29

Broker_0 is shutting down...

Simulation: No more future events

CloudInformationService: Notify all CloudSim entities for shutting down.

Datacenter_0 is shutting down...

HostFaultInjection0: is shutting down...

HostFaultInjection1: is shutting down...

HostFaultInjection2: is shutting down...

HostFaultInjection3: is shutting down...

HostFaultInjection4: is shutting down...

HostFaultInjection5: is shutting down...

HostFaultInjection6: is shutting down...

HostFaultInjection7: is shutting down...

HostFaultInjection8: is shutting down...

HostFaultInjection9: is shutting down...

Broker_0 is shutting down...

Simulation completed.

Simulation completed.

========== OUTPUT ==========

Cloudlet ID	STATUS	Data center ID	VM ID	Time	Start Time	Finish Time
170	SUCCESS	2	20	2560	0.1	2560.1
20	SUCCESS	2	20	2560	0.1	2560.1
200	SUCCESS	2	20	2560	0.1	2560.1
50	SUCCESS	2	20	2560	0.1	2560.1
230	SUCCESS	2	20	2560	0.1	2560.1
80	SUCCESS	2	20	2560	0.1	2560.1
260	SUCCESS	2	20	2560	0.1	2560.1
110	SUCCESS	2	20	2560	0.1	2560.1
290	SUCCESS	2	20	2560	0.1	2560.1
140	SUCCESS	2	20	2560	0.1	2560.1
320	SUCCESS	2	20	2560	0.1	2560.1

350	SUCCESS	2	20	2560	381	SUCCESS	2	21	2560
0.1	2560.1				0.1	2560.1			
380	SUCCESS	2	20	2560	411	SUCCESS	2	21	2560
0.1	2560.1				0.1	2560.1			
410	SUCCESS	2	20	2560	441	SUCCESS	2	21	2560
0.1	2560.1				0.1	2560.1			
440	SUCCESS	2	20	2560	471	SUCCESS	2	21	2560
0.1	2560.1				0.1	2560.1			
470	SUCCESS	2	20	2560	22	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
21	SUCCESS	2	21	2560	52	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
51	SUCCESS	2	21	2560	82	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
81	SUCCESS	2	21	2560	112	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
111	SUCCESS	2	21	2560	142	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
141	SUCCESS	2	21	2560	172	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
171	SUCCESS	2	21	2560	202	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
201	SUCCESS	2	21	2560	232	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
231	SUCCESS	2	21	2560	262	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
261	SUCCESS	2	21	2560	292	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
291	SUCCESS	2	21	2560	322	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
321	SUCCESS	2	21	2560	352	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			
351	SUCCESS	2	21	2560	382	SUCCESS	2	22	2560
0.1	2560.1				0.1	2560.1			

412 SUCCESS 2 22 2560 0.1 2560.1
442 SUCCESS 2 22 2560 0.1 2560.1
472 SUCCESS 2 22 2560 0.1 2560.1
23 SUCCESS 2 23 2560 0.1 2560.1
53 SUCCESS 2 23 2560 0.1 2560.1
83 SUCCESS 2 23 2560 0.1 2560.1
113 SUCCESS 2 23 2560 0.1 2560.1
143 SUCCESS 2 23 2560 0.1 2560.1
173 SUCCESS 2 23 2560 0.1 2560.1
203 SUCCESS 2 23 2560 0.1 2560.1
233 SUCCESS 2 23 2560 0.1 2560.1
263 SUCCESS 2 23 2560 0.1 2560.1
293 SUCCESS 2 23 2560 0.1 2560.1
323 SUCCESS 2 23 2560 0.1 2560.1
353 SUCCESS 2 23 2560 0.1 2560.1
383 SUCCESS 2 23 2560 0.1 2560.1
413 SUCCESS 2 23 2560 0.1 2560.1
443 SUCCESS 2 23 2560 0.1 2560.1
473 SUCCESS 2 23 2560 0.1 2560.1
24 SUCCESS 2 24 2560 0.1 2560.1
54 SUCCESS 2 24 2560 0.1 2560.1
84 SUCCESS 2 24 2560 0.1 2560.1
114 SUCCESS 2 24 2560 0.1 2560.1
144 SUCCESS 2 24 2560 0.1 2560.1
174 SUCCESS 2 24 2560 0.1 2560.1
204 SUCCESS 2 24 2560 0.1 2560.1
234 SUCCESS 2 24 2560 0.1 2560.1
264 SUCCESS 2 24 2560 0.1 2560.1
294 SUCCESS 2 24 2560 0.1 2560.1
324 SUCCESS 2 24 2560 0.1 2560.1
354 SUCCESS 2 24 2560 0.1 2560.1
384 SUCCESS 2 24 2560 0.1 2560.1
414 SUCCESS 2 24 2560 0.1 2560.1
444 SUCCESS 2 24 2560 0.1 2560.1

474	SUCCESS	2	24	2560	26	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
25	SUCCESS	2	25	2560	56	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
55	SUCCESS	2	25	2560	86	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
85	SUCCESS	2	25	2560	116	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
115	SUCCESS	2	25	2560	146	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
145	SUCCESS	2	25	2560	176	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
175	SUCCESS	2	25	2560	206	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
205	SUCCESS	2	25	2560	236	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
235	SUCCESS	2	25	2560	266	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
265	SUCCESS	2	25	2560	296	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
295	SUCCESS	2	25	2560	326	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
325	SUCCESS	2	25	2560	356	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
355	SUCCESS	2	25	2560	386	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
385	SUCCESS	2	25	2560	416	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
415	SUCCESS	2	25	2560	446	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
445	SUCCESS	2	25	2560	476	SUCCESS	2	26	2560
0.1	2560.1				0.1	2560.1			
475	SUCCESS	2	25	2560	27	SUCCESS	2	27	2560
0.1	2560.1				0.1	2560.1			

57	SUCCESS	2	27	2560	88	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
87	SUCCESS	2	27	2560	118	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
117	SUCCESS	2	27	2560	148	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
147	SUCCESS	2	27	2560	178	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
177	SUCCESS	2	27	2560	208	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
207	SUCCESS	2	27	2560	238	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
237	SUCCESS	2	27	2560	268	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
267	SUCCESS	2	27	2560	298	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
297	SUCCESS	2	27	2560	328	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
327	SUCCESS	2	27	2560	358	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
357	SUCCESS	2	27	2560	388	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
387	SUCCESS	2	27	2560	418	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
417	SUCCESS	2	27	2560	448	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
447	SUCCESS	2	27	2560	478	SUCCESS	2	28	2560
0.1	2560.1				0.1	2560.1			
477	SUCCESS	2	27	2560	29	SUCCESS	2	29	2560
0.1	2560.1				0.1	2560.1			
28	SUCCESS	2	28	2560	59	SUCCESS	2	29	2560
0.1	2560.1				0.1	2560.1			
58	SUCCESS	2	28	2560	89	SUCCESS	2	29	2560
0.1	2560.1				0.1	2560.1			

119	SUCCESS	2	29	2560	120	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
149	SUCCESS	2	29	2560	150	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
179	SUCCESS	2	29	2560	180	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
209	SUCCESS	2	29	2560	210	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
239	SUCCESS	2	29	2560	240	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
269	SUCCESS	2	29	2560	270	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
299	SUCCESS	2	29	2560	300	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
329	SUCCESS	2	29	2560	330	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
359	SUCCESS	2	29	2560	360	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
389	SUCCESS	2	29	2560	390	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
419	SUCCESS	2	29	2560	420	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
449	SUCCESS	2	29	2560	450	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
479	SUCCESS	2	29	2560	480	SUCCESS	2	0	2719.94
0.1	2560.1				0.1	2720.04			
0	SUCCESS	2	0	2719.94	5	SUCCESS	2	5	2719.94
0.1	2720.04				0.1	2720.04			
30	SUCCESS	2	0	2719.94	35	SUCCESS	2	5	2719.94
0.1	2720.04				0.1	2720.04			
60	SUCCESS	2	0	2719.94	65	SUCCESS	2	5	2719.94
0.1	2720.04				0.1	2720.04			
90	SUCCESS	2	0	2719.94	95	SUCCESS	2	5	2719.94
0.1	2720.04				0.1	2720.04			

125	SUCCESS	2	5	2719.94	130	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
155	SUCCESS	2	5	2719.94	160	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
185	SUCCESS	2	5	2719.94	190	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
215	SUCCESS	2	5	2719.94	220	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
245	SUCCESS	2	5	2719.94	250	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
275	SUCCESS	2	5	2719.94	280	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
305	SUCCESS	2	5	2719.94	310	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
335	SUCCESS	2	5	2719.94	340	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
365	SUCCESS	2	5	2719.94	370	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
395	SUCCESS	2	5	2719.94	400	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
425	SUCCESS	2	5	2719.94	430	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
455	SUCCESS	2	5	2719.94	460	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
485	SUCCESS	2	5	2719.94	490	SUCCESS	2	10	2719.94
0.1	2720.04				0.1	2720.04			
10	SUCCESS	2	10	2719.94	11	SUCCESS	2	11	2719.94
0.1	2720.04				0.1	2720.04			
40	SUCCESS	2	10	2719.94	41	SUCCESS	2	11	2719.94
0.1	2720.04				0.1	2720.04			
70	SUCCESS	2	10	2719.94	71	SUCCESS	2	11	2719.94
0.1	2720.04				0.1	2720.04			
100	SUCCESS	2	10	2719.94	101	SUCCESS	2	11	2719.94
0.1	2720.04				0.1	2720.04			

131	SUCCESS	2	11	2719.94	121	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
161	SUCCESS	2	11	2719.94	151	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
191	SUCCESS	2	11	2719.94	181	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
221	SUCCESS	2	11	2719.94	211	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
251	SUCCESS	2	11	2719.94	241	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
281	SUCCESS	2	11	2719.94	271	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
311	SUCCESS	2	11	2719.94	301	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
341	SUCCESS	2	11	2719.94	331	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
371	SUCCESS	2	11	2719.94	361	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
401	SUCCESS	2	11	2719.94	391	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
431	SUCCESS	2	11	2719.94	421	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
461	SUCCESS	2	11	2719.94	451	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
491	SUCCESS	2	11	2719.94	481	SUCCESS	2	1	2719.94
0.1	2720.04				0.1	2720.04			
1	SUCCESS	2	1	2719.94	6	SUCCESS	2	6	2719.94
0.1	2720.04				0.1	2720.04			
31	SUCCESS	2	1	2719.94	36	SUCCESS	2	6	2719.94
0.1	2720.04				0.1	2720.04			
61	SUCCESS	2	1	2719.94	66	SUCCESS	2	6	2719.94
0.1	2720.04				0.1	2720.04			
91	SUCCESS	2	1	2719.94	96	SUCCESS	2	6	2719.94
0.1	2720.04				0.1	2720.04			

126	SUCCESS	2	6	2719.94	132	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
156	SUCCESS	2	6	2719.94	162	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
186	SUCCESS	2	6	2719.94	192	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
216	SUCCESS	2	6	2719.94	222	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
246	SUCCESS	2	6	2719.94	252	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
276	SUCCESS	2	6	2719.94	282	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
306	SUCCESS	2	6	2719.94	312	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
336	SUCCESS	2	6	2719.94	342	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
366	SUCCESS	2	6	2719.94	372	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
396	SUCCESS	2	6	2719.94	402	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
426	SUCCESS	2	6	2719.94	432	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
456	SUCCESS	2	6	2719.94	462	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
486	SUCCESS	2	6	2719.94	492	SUCCESS	2	12	2719.94
0.1	2720.04				0.1	2720.04			
12	SUCCESS	2	12	2719.94	13	SUCCESS	2	13	2719.94
0.1	2720.04				0.1	2720.04			
42	SUCCESS	2	12	2719.94	43	SUCCESS	2	13	2719.94
0.1	2720.04				0.1	2720.04			
72	SUCCESS	2	12	2719.94	73	SUCCESS	2	13	2719.94
0.1	2720.04				0.1	2720.04			
102	SUCCESS	2	12	2719.94	103	SUCCESS	2	13	2719.94
0.1	2720.04				0.1	2720.04			

133	SUCCESS	2	13	2719.94	122	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
163	SUCCESS	2	13	2719.94	152	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
193	SUCCESS	2	13	2719.94	182	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
223	SUCCESS	2	13	2719.94	212	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
253	SUCCESS	2	13	2719.94	242	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
283	SUCCESS	2	13	2719.94	272	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
313	SUCCESS	2	13	2719.94	302	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
343	SUCCESS	2	13	2719.94	332	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
373	SUCCESS	2	13	2719.94	362	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
403	SUCCESS	2	13	2719.94	392	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
433	SUCCESS	2	13	2719.94	422	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
463	SUCCESS	2	13	2719.94	452	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
493	SUCCESS	2	13	2719.94	482	SUCCESS	2	2	2719.94
0.1	2720.04				0.1	2720.04			
2	SUCCESS	2	2	2719.94	7	SUCCESS	2	7	2719.94
0.1	2720.04				0.1	2720.04			
32	SUCCESS	2	2	2719.94	37	SUCCESS	2	7	2719.94
0.1	2720.04				0.1	2720.04			
62	SUCCESS	2	2	2719.94	67	SUCCESS	2	7	2719.94
0.1	2720.04				0.1	2720.04			
92	SUCCESS	2	2	2719.94	97	SUCCESS	2	7	2719.94
0.1	2720.04				0.1	2720.04			

127	SUCCESS	2	7	2719.94	134	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
157	SUCCESS	2	7	2719.94	164	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
187	SUCCESS	2	7	2719.94	194	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
217	SUCCESS	2	7	2719.94	224	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
247	SUCCESS	2	7	2719.94	254	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
277	SUCCESS	2	7	2719.94	284	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
307	SUCCESS	2	7	2719.94	314	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
337	SUCCESS	2	7	2719.94	344	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
367	SUCCESS	2	7	2719.94	374	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
397	SUCCESS	2	7	2719.94	404	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
427	SUCCESS	2	7	2719.94	434	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
457	SUCCESS	2	7	2719.94	464	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
487	SUCCESS	2	7	2719.94	494	SUCCESS	2	14	2719.94
0.1	2720.04				0.1	2720.04			
14	SUCCESS	2	14	2719.94	15	SUCCESS	2	15	2719.94
0.1	2720.04				0.1	2720.04			
44	SUCCESS	2	14	2719.94	45	SUCCESS	2	15	2719.94
0.1	2720.04				0.1	2720.04			
74	SUCCESS	2	14	2719.94	75	SUCCESS	2	15	2719.94
0.1	2720.04				0.1	2720.04			
104	SUCCESS	2	14	2719.94	105	SUCCESS	2	15	2719.94
0.1	2720.04				0.1	2720.04			

135	SUCCESS	2	15	2719.94	123	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
165	SUCCESS	2	15	2719.94	153	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
195	SUCCESS	2	15	2719.94	183	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
225	SUCCESS	2	15	2719.94	213	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
255	SUCCESS	2	15	2719.94	243	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
285	SUCCESS	2	15	2719.94	273	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
315	SUCCESS	2	15	2719.94	303	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
345	SUCCESS	2	15	2719.94	333	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
375	SUCCESS	2	15	2719.94	363	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
405	SUCCESS	2	15	2719.94	393	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
435	SUCCESS	2	15	2719.94	423	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
465	SUCCESS	2	15	2719.94	453	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
495	SUCCESS	2	15	2719.94	483	SUCCESS	2	3	2719.94
0.1	2720.04				0.1	2720.04			
3	SUCCESS	2	3	2719.94	8	SUCCESS	2	8	2719.94
0.1	2720.04				0.1	2720.04			
33	SUCCESS	2	3	2719.94	38	SUCCESS	2	8	2719.94
0.1	2720.04				0.1	2720.04			
63	SUCCESS	2	3	2719.94	68	SUCCESS	2	8	2719.94
0.1	2720.04				0.1	2720.04			
93	SUCCESS	2	3	2719.94	98	SUCCESS	2	8	2719.94
0.1	2720.04				0.1	2720.04			

128	SUCCESS	2	8	2719.94	136	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
158	SUCCESS	2	8	2719.94	166	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
188	SUCCESS	2	8	2719.94	196	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
218	SUCCESS	2	8	2719.94	226	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
248	SUCCESS	2	8	2719.94	256	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
278	SUCCESS	2	8	2719.94	286	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
308	SUCCESS	2	8	2719.94	316	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
338	SUCCESS	2	8	2719.94	346	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
368	SUCCESS	2	8	2719.94	376	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
398	SUCCESS	2	8	2719.94	406	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
428	SUCCESS	2	8	2719.94	436	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
458	SUCCESS	2	8	2719.94	466	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
488	SUCCESS	2	8	2719.94	496	SUCCESS	2	16	2719.94
0.1	2720.04				0.1	2720.04			
16	SUCCESS	2	16	2719.94	17	SUCCESS	2	17	2719.94
0.1	2720.04				0.1	2720.04			
46	SUCCESS	2	16	2719.94	47	SUCCESS	2	17	2719.94
0.1	2720.04				0.1	2720.04			
76	SUCCESS	2	16	2719.94	77	SUCCESS	2	17	2719.94
0.1	2720.04				0.1	2720.04			
106	SUCCESS	2	16	2719.94	107	SUCCESS	2	17	2719.94
0.1	2720.04				0.1	2720.04			

137	SUCCESS	2	17	2719.94	124	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
167	SUCCESS	2	17	2719.94	154	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
197	SUCCESS	2	17	2719.94	184	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
227	SUCCESS	2	17	2719.94	214	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
257	SUCCESS	2	17	2719.94	244	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
287	SUCCESS	2	17	2719.94	274	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
317	SUCCESS	2	17	2719.94	304	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
347	SUCCESS	2	17	2719.94	334	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
377	SUCCESS	2	17	2719.94	364	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
407	SUCCESS	2	17	2719.94	394	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
437	SUCCESS	2	17	2719.94	424	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
467	SUCCESS	2	17	2719.94	454	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
497	SUCCESS	2	17	2719.94	484	SUCCESS	2	4	2719.94
0.1	2720.04				0.1	2720.04			
4	SUCCESS	2	4	2719.94	9	SUCCESS	2	9	2719.94
0.1	2720.04				0.1	2720.04			
34	SUCCESS	2	4	2719.94	39	SUCCESS	2	9	2719.94
0.1	2720.04				0.1	2720.04			
64	SUCCESS	2	4	2719.94	69	SUCCESS	2	9	2719.94
0.1	2720.04				0.1	2720.04			
94	SUCCESS	2	4	2719.94	99	SUCCESS	2	9	2719.94
0.1	2720.04				0.1	2720.04			

129	SUCCESS	2	9	2719.94	138	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
159	SUCCESS	2	9	2719.94	168	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
189	SUCCESS	2	9	2719.94	198	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
219	SUCCESS	2	9	2719.94	228	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
249	SUCCESS	2	9	2719.94	258	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
279	SUCCESS	2	9	2719.94	288	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
309	SUCCESS	2	9	2719.94	318	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
339	SUCCESS	2	9	2719.94	348	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
369	SUCCESS	2	9	2719.94	378	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
399	SUCCESS	2	9	2719.94	408	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
429	SUCCESS	2	9	2719.94	438	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
459	SUCCESS	2	9	2719.94	468	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
489	SUCCESS	2	9	2719.94	498	SUCCESS	2	18	2719.94
0.1	2720.04				0.1	2720.04			
18	SUCCESS	2	18	2719.94	19	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
48	SUCCESS	2	18	2719.94	49	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
78	SUCCESS	2	18	2719.94	79	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
108	SUCCESS	2	18	2719.94	109	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			

139	SUCCESS	2	19	2719.94	349	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
169	SUCCESS	2	19	2719.94	379	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
199	SUCCESS	2	19	2719.94	409	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
229	SUCCESS	2	19	2719.94	439	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
259	SUCCESS	2	19	2719.94	469	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
289	SUCCESS	2	19	2719.94	499	SUCCESS	2	19	2719.94
0.1	2720.04				0.1	2720.04			
319	SUCCESS	2	19	2719.94					
0.1	2720.04								

========== OUTPUT ==========

Host ID	VM Id		
0	[]	0	[]
1	[]	1	[]
2	[]	2	[]
3	[]	3	[]
4	[]	4	[]
5	[]	5	[]
6	[]	6	[]
7	[]	7	[]
8	[]	8	[]
9	[]	9	[]

Scenario with faultInjection finished!

[INFO] --

[INFO] BUILD SUCCESS

[INFO] --

[INFO] Total time: 4.628s

[INFO] Finished at: Thu Jun 29 15:46:10 EAT 2017

[INFO] Final Memory: 7M/155M

[INFO] --