

MEU

جامعة الشرق الأوسط
MIDDLE EAST UNIVERSITY
Amman - Jordan عمان - الأردن

Towards Achieving, Self-Load Balancing In Autonomic Overlay Networks.

A Thesis Submitted in Partial Fulfillment of the Requirements of
the Master Degree in Computer Science

By

Mwafaq Salim Al-Zboon

Main Supervisor

Dr. Hussein H. Owaied

Co-Supervisor

Dr. Ibrahim Al-Oqily

Faculty of Information Technology

Middle East University

Amman-Jordan

January, 2012

تفويض خطي

أنا الطالب موفق سالم عطا الزبون أفوض جامعة الشرق الأوسط للدراسات العليا بتزويد نسخ من رسالتي ورقيا وإلكترونيا للمكتبات، أو المنظمات، أو الهيئات والمؤسسات المعنية بالأبحاث والدراسات العلمية عند طلبها.

الاسم: موفق سالم عطا الزبون

التاريخ: شباط 2012

التوقيع: 

Authorization Statement

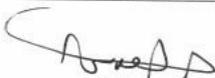
I, Mwafaq Salim Al-Zboon, authorize the Middle East University to supply a copy of my thesis to libraries, establishments, or individuals upon their request.

Signature: 

Date: February 2012.

Middle East University**Examination Committee Decision**

This to certify that the thesis entitled "Towards Achieving, Self-Load Balancing in Autonomic Overlay Networks" was successfully defended and approved on January 18th 2012

Examination Committee Members	Signature
Dr. Hussein H. Owaied Associate Professor, Department of Computer Science (Middle East University)	
Dr. Ibrahim Z. R. Al-Oqily Associate Professor, Department of Computer Science (The Hashemite University)	
Dr. Oleg Victorov Associate Professor, Department of Computer Science (Middle East University)	
Dr. Saad Bani Mohammad Associate Professor, Department of Computer Science (Al al-Bayt University)	

Declaration Statement

I do hereby declare that the present research work has been carried out by me under the main supervision of: Dr. Hussein H. Owaied and Co-supervision of: Dr. Ibrahim Al-Oqily and that work has not been submitted elsewhere for any other degree, fellowship or any other similar title.

Signature: 

Date: February 2012.

Mwafaq Salim Al-Zboon

Department of computer information system

Faculty of Information Technology

Middle East University

Dedication

To my dear parents and specially my mother for her love, care and support.

To my dear wife for her support, love and patience throughout my work.

Also, to my children- Lujin, Faisal and Zaid and my brothers and sisters.

To all teachers, friends and those who helped and encouraged me throughout my work.

For all of these, I dedicate this thesis supplicating Allah for benefit and success.

Acknowledgments

First and foremost, Thanks be to Allah for my life, you made my life bountiful, may your name be exalted, honored, and glorified.

A journey is easier when travelled together; interdependence is certainly more valuable than independence. This thesis is the result of 13 months of work whereby, I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

The development of this thesis was a time of personal growth and that development did not always take place without pain.

I would like to thank all those who supported me. Especially, I wish to thank supervisor Dr. Hussein H. Owaied for all his advice and support during this studies. Special thanks to Dr. Ibrahim Al-Oqily for being a great Co-supervisor. He spent a lot of time helping me to complete this work. I am very much grateful to Dr. Saad Bani Mohammad and Dr. Mohammad Malkawi and Dr. Khalid Sarayrah. My thanks to Mr. Ahmed shtnawi and all my friends for various help. Finally, I wish to thank my family for their unflagging love and support throughout my life. I thank all my brothers and sisters who were with me all the time. My deepest gratitude goes to my wife. This thesis was simply impossible to complete without her. I am indebted to my mother and father for their inspiration and support.

Table of Contents

Chapter 1	Introduction	
1.1.	Introduction	1
1.2.	Problem Definition	2
1.3.	Objectives	4
1.4.	Motivation	4
1.5.	Contributions	5
1.6.	Study Boundaries	6
1.7.	Thesis Structure	7
Chapter 2	Background	
2.1.	Overview.....	8
2.2.	Autonomic Computing (AC).....	8
2.2.1.	Autonomic Service Specific Overlay Networks (A-SSON).....	10
2.2.2.	Autonomic Overlays (AO) self-composition.....	11
2.2.3.	Quadtree.....	15
2.2.4.	Morton Ordering.....	18
2.3.	Load Balance.....	19
2.3.1.	Self Load Balancing.....	20
2.3.2.	Analytical Load Balancing Algorithms.....	21
2.3.3.	Types of load balancing and policies	25
2.3.3.1	Load Balancing Types (LBT).....	26
2.3.3.2	Load balancing policies	30

Chapter 3	Related work	
3.1	Load Balance.....	32
3.2	Load Balancing in different domains.....	32
3.2.1.	Load Balancing In Grid Networks.....	32
3.2.2.	Load Balancing in Wireless networks.....	38
3.2.3.	Load Balancing in Overlay networks.....	42
3.3.	The Implementation of DQT for Networks	43
3.3.1	Distributed Quadtree for Spatial Querying in WSNs.....	44
3.3.2	A QT-Based Data Dissemination Pr. for WSNs with mobile sinks	48
Chapter 4	Design of Algorithms and Methods Used	
4.1.	Overview.....	52
4.2.	The Strategy of Designing a Self-Load Balancing Scheme.....	53
4.2.1.	Scheme Formulation.....	54
4.2.2.	Computing the Processing speed.....	55
4.2.3.	MPs Power Calculation.....	56
4.2.4.	Standardize the parameters Measurements.....	57
4.2.5.	Local knowledge calculation.....	57
4.2.6.	The main formula.....	58
4.2.7.	Formula for each type.....	58
4.2.8.	Proof.....	59
4.2.9.	Encoding the resources in the topology of the overlay.....	61
4.2.10.	Selection of powerful MPs.....	62

Chapter 5	Proposed self load balance scheme	
5.1.	Introduction	63
5.2.	Partitioning the geographical location of the networks.....	63
5.3.	Building the DQT.....	64
5.3.1.	Building the DQT dependent only on local knowledge.....	64
5.3.2.	Node Type.....	65
5.3.3.	Indexing the geographical location Spatial Indexing	65
5.3.4.	Parent Detection.....	69
5.3.5.	My Parent Detection.....	69
5.3.6.	Parent level Detection.....	70
5.3.7.	Children Detection.....	71
5.3.8.	Root Detection.....	72
5.3.9.	Brothers Root Detection.....	72
5.3.10.	Calculating the power.....	73
5.3.11.	Routing Algorithm.....	74
5.3.12.	Joining the Overlay.....	78
5.3.13.	Leaving the Overlay.....	78
5.4.	A Self-Load Balancing.....	87
5.4.1.	The Procedure Power Percentage.....	87
5.4.2.	The Procedure number of incoming edges.....	87
5.4.3.	The Procedure Joining incoming edges.....	88
5.4.4.	Procedure Assign Job.....	89
5.4.5.	Procedure Delete Edge.....	90

5.4.6.	Procedure Add New Edge.....	90
Chapter 6	Experimental Evaluation	
6.1.	Network simulation.....	91
6.2.	The Simulation Tool (J-Sim Simulator)	91
6.3.	J-Sim for Network Simulation.....	93
6.4.	Justification of the Method of Study.....	93
6.5.	Experiment One: Distributed Quadtree (DQT).....	95
6.5.1.	The DQT Building Message for Each Level.....	95
6.5.2.	The Stretch	97
6.5.3.	Response Time	99
6.5.4.	Success Rate	100
6.6.	Experiment Two: Joined Node.....	101
6.6.1	The Network Load.....	101
6.6.2	Response Time.....	102
6.7.	Experiment Three: Left Node.....	103
6.7.1.	The Network Load.....	103
6.7.2.	Response Time.....	104
6.8.	Experiment Four The Self-Load Balancing.....	105
6.8.1	Number of job equal 100 jobs.....	105
6.8.2	Number of job equal 200 jobs.....	106
6.8.3	Number of job equal 300 jobs.....	107
6.8.4	Number of job equal 400 jobs.....	108
6.8.5	Number of job equal 500 jobs.....	109
6.8.6	Number of job equal 600 jobs.....	110

Chapter 7	Verification & Validation of designing a self load Balancing scheme	
7.1.	Overview.....	111
7.2.	Verification of the Self-Load Balancing Based Algorithm.....	113
7.2.1.	Verification of the Building DQT.....	113
7.2.2.	Verification of the Self-Load Balancing.....	114
7.3.	Validation of the self-load balancing algorithm.....	116
7.3.1	Validating The DQT Building Message for Each Level.....	116
7.3.2	Validating The Stretch And Response Time.....	117
7.3.3	Validating algorithm for Joined and leave Node.....	118
7.4.	Discussion and Results.....	119
Chapter 8	Conclusion and Future Work	
8.1.	Conclusion	121
8.2.	Future Work	121
References		122

List of Figures:

Figure No.	Figure Name	Page
Figure 1.1	MPs at certain area in the network are overloaded while other MPs are less overloaded	3
Figure 2.1	Functional details of an autonomic manager	10
Figure 2.2	Autonomic Overlay Architecture	12
Figure 2.3	The relationships between SAM and SSON-AM	14
Figure 2.4	Quad Tree space partitioning	16
Figure 3.1	Estimation and Status Exchange intervals	37
Figure 3.2	DQT Structure and Construction	45
Figure 3.3	Different direction	46
Figure 3.4	Node addressing and tree structures	46
Figure 3.5	(a) Sensor network space N partitioning. (b) QT representation	50
Figure 5.1	Procedure Indexing	66
Figure 5.2	Procedure ParentDetection	69
Figure 5.3	Procedure MyParentDetection	70
Figure 5.4	Procedure ParentLevelDetection	71
Figure 5.5	Procedure ChildrenDetection	71
Figure 5.6	Procedure RootDetection	72
Figure 5.7	Procedure BrotherRootDetection	73
Figure 5.8	Procedure CalculatePower	73

Figure 5.9	Procedure INFOMessage	74
Figure 5.10	Procedure ListMessage	75
Figure 5.11	Procedure WhereMyParentQuery	75
Figure 5.12	Procedure SummeryMessage	76
Figure 5.13	Procedure TimeOutMessage	76
Figure 5.14	Procedure NumbersOfNodes	77
Figure 5.15	Procedure ParentsMessage	77
Figure 5.16	Procedure JoiningOverlay	78
Figure 5.17	Procedure SelectiveParentLeave	79
Figure 5.18	Procedure SelectiveChildLeave	79
Figure 5.19	Procedure ForceParentLeave	80
Figure 5.20	Procedure ForceChildLeave	80
Figure 5.21	show our network as a two dimensional array	81
Figure 5.22	Shows our partitioning and indexing procedures in first level and its root	81
Figure 5.23	Shows our partitioning and indexing procedures in second level	82
Figure 5.24	Shows our partitioning and indexing procedures in third level	82
Figure 5.25	Shows the upper left (NW) section	83
Figure 5.26	Shows the upper left (NW) section, and the quadtree representation	83
Figure 5.27	Shows the upper right (NE) section	84
Figure 5.28	Shows the upper right (NE) section, and quadtree representation	84

Figure 5.29	Shows the lower left (SW) section	85
Figure 5.30	Shows the lower left (SW) section, and quadtree representation	85
Figure 5.31	Shows the lower left (SE) section	86
Figure 5.32	Shows the lower right (SE) section, and quadtree representation	86
Figure 5.33	Procedure Power Percentage	87
Figure 5.34	Procedure InCom Edges	88
Figure 5.35	Procedure Join InCom Edges	88
Figure 5.36	Procedure Assign Job	89
Figure 5.37	Procedure Delet Edge	90
Figure 5.38	Procedure Add Edge	90
Figure 6.1	Distributed Quadtree building message	96
Figure 6.2 a	Direct Hop Stretch	97
Figure 6.2 b	DQT Hop Stretch	98
Figure 6.3	Response Time	99
Figure 6.4	Success rate	100
Figure 6.5	Network load	101
Figure 6.6	Response Time	102
Figure 6.7	Network load	103
Figure 6.8	Response Time	104
Figure 6.9	Percentage usage of MPs when number of jobs equals 100 jobs	105
Figure 6.10	Percentage usage of MPs when number of jobs equals 200 jobs	106

Figure 6.11	Percentage usage of MPs when number of jobs equals 300 jobs	107
Figure 6.12	Percentage usage of MPs when number of jobs equals 400 jobs	108
Figure 6.13	Percentage usage of MPs when number of jobs equals 500 jobs	109
Figure 6.14	Percentage usage of MPs when number of jobs equals 600 jobs	110

List of Tables:

Table No.	Table Name	Page
Table 2.1	Performance Analysis of Load Balancing Algorithms	22
Table 2.2	Qualitative Parametric Comparison of Load Balancing Algorithms	24
Table 4.1	MP resources and the percentage value for each service provide	56
Table 4.2	MP resource and the percentage value for each service provide	56
Table 5.1	The original node distribution in a grid	66
Table 5.2	Represents the Morton Order after apply it over table 5.1	68
Table 7.1	The number of queerer node, direct hop, DQT hop and response time of the simulation model	117
Table 7.2	The number of Join, Left node, network load and response time of the simulation model	118

List of Abbreviation

AC: Autonomic Computing

ACS: Ambient Control Space

ALB: Autonomic Load Balancing

ANs: Ambient Networks

AO: Autonomic Overlays

AFWBM: Autonomic Flowing Water Balancing Method

ARMS: Agent-based Resource Management System

ART: Average Response Time

AT: Application Tools

AVI: Audio Video Interleave

BON: Balanced Overlay Networks

CN: Coordinator Nodes

CSIS: Common Service Information System

CSMA: Carrier-Sense Multiple Access protocol

DQT: Distributed Quad Tree

DLB: Dynamic Load Balancing

EE: Evaluation Engine

E-UTRAN: Evolved Universal Terrestrial Radio Access

FTP: File Transfer Protocol

GIC: Grid Information Center

LB: Load Balancing

LBA: Load Balancing on Arrival

LBT: Load Balancing Types

LCA: linked Cluster Algorithm

LTE: Long Term Evolution

MAC: Medium Access Control protocol

MACA: Multiple Access with Collision Avoidance protocol

MC: Media Client

MCLB: Multi hop Clustering Algorithm for Load Balancing

MPs: Media Ports

MS: Media Server

NE: North East

NW: North West

ON: Overlay Networks

ONAMs: Overlay Network Autonomic Managers

OSL: Overlay Support Layer

PID: Physical ID

QT: Quad Tree

QoS: Quality of Service

RAN: Radio Access Networks

RW: Random Walk

RT: Resource Tools

RTS/CTS: Request To Send/Clear To Send

SAM: System Autonomic Managers

SLB: Static Load Balancing

SSON-AM: Service Specific Overlay Networks Autonomic Manager

SSONs: Service Specific Overlay Networks

SE: South East

SW: South West

TDMA: Time Division Multiple Access

TTFB: Time To First Byte

VID: Virtual ID

WN: Worker Nodes

WSN: Wireless Sensor Networks

Abstract

Services Specific Overlay Networks are virtual networks built on top of the physical computer network to meet the users' specific requirements. They are basically used to deliver multimedia content from a streaming media server to the user. A central component to this kind of networks is the media ports. They are network side functions that offer extra tasks such as catching, synchronization, and adaptation for the multimedia content.

Having a specific overlay network for each user implies that huge number of overlay networks will coexist. This could lead to competition on the media ports. In addition to that, users may join and leave the network which will render managing this huge number of networks a complex task.

In this thesis, we propose a self-load balancing scheme for service specific overlay networks. It is intended to balance the loads between media ports, which will fairly distribute tasks between media ports to increase its efficiency. The proposed solution builds a quadtree overlay, quantifies the media ports' resources, and encodes the quantified values into an incoming edges overlay. Self-load balancing is then achieved by sending tasks to the highest in degree media ports.

الخلاصة

الشبكات الفوقية هي شبكات خياليه، افتراضية يتم بناؤها فوق شبكات الحاسوب المادية المنشأة مسبقاً. تم بناء نوع خاص من هذه الشبكات ليوافق متطلبات محددة ومعينه من قبل المستخدم وتسمى هذه الشبكات بشبكة الخدمة المحددة او المخصصة الفوقية (SSONs).

الجزء الرئيس في الشبكة الفوقية هو الـ (MP). الـ (MP) هي اجزاء وظيفية في الشبكة حيث انها تضيف وظائف جديده للشبكة مثل التخزين و التزامن و التكيف. ان ادارة وتحقيق توازن الاحمال في هذه البيئه يعتبر تحدياً. وذلك لوجود كم هائل من شبكات الخدمة المخصصة الفوقية وتنوعها فكل واحدة من هذه الشبكات هائلة العدد ومخصصة لمستخدم واحد فقط. في مثل هذه الظروف ينشأ تنافس حتمي على المصادر الاساسية لمراكز الخدمة مما يؤدي الى توزيع غير متوازن للاحمال.

زيادة على ذلك فان التكنولوجيا الحديثة تواجه تحدي وهو زيادة التعقيد والتكلفة وتغاير خواص البنية التحتية. وبالرغم من هذا فان الاستراتيجيات التقليدية لا تلبى ولا تتغلب على حاجات التكنولوجيا الحالية. في هذه الاطروحة نقترح خطة جديده وحديثة لتحقيق التوازن الذاتي في الشبكات الفوقية ذات الاستقلال الذاتي .

الخطة المقترحة تمتاز بتركيبية طبقية تحقق الديناميكيه وتوازن ذاتي للاحمال بالارتكاز على الشجرة الرباعية الموزعه (DQT) وهذه الخطة تحمل المواصفات التالية : الطبقيه والتي لا تبالي باختلافات البنية التحتية وتدعم زيادة ونقص حجم الشبكة، ومستقلة عن اي بناء مادي من عمارة الشبكات وتحقق التوازن الذاتي بالاعتماد على المعلومات المتوافرة لديها فقط في كل (MP).

تم فحص والتأكد من الخطة المقترحة لتحقيق التوازن الذاتي في الشبكات الفوقية ذات الاستقلال الذاتي من خلال المحاكاه وتبين بانها تتمتع بديناميكية عالية وبنية تنظيمية ذاتيه لكل من الشبكة الفوقية للـ (DQT and MPs) . وكلاهما ابدو سيمات رائعة في مواجهة الفشل وفي عملية تحقيق التوازن على مختلف الـ (MPs). علماً بان التجارب التي تم اجراؤها اخذت بعين الاعتبار تنوع المصادر الحاسوبية المتوافرة والطلبات المتزايدة في تنفيذ الاعمال .

Chapter One

1

Introduction

1.1 Introduction

Autonomic Computing (AC), where technology manages technology, was motivated by the increasing of technology complexity, the increased size of computing infrastructure, and the ballooning maintenance costs of infrastructure, and the shortages of skilled labor (IBM Corporation, 2006). Overlay networks face the same challenges. They are growing so fast, they are being deployed on the fly without special equipments, they are being used to solve problems in network routing, and they are being used to realize services that can't be implemented otherwise (Khalid, Haye, Khan, & Shamail, 2009), (IBM Corporation, 2006) (Al-Oqily, & Karmouch, 2008).

Also the expression outgrowth of networks and services has lead to new complex mediums. To survive with this convolution, IBM Corporation proposed AC in (IBM Corporation, 2006). It permits systems to run and control themselves as an alternative of relying on IT specialists. Overlay networks are receiving a great concentration due to the reliable and effective services that they provide. One type of overlay networks is being designed to meet user's requirements. It is called Service Specific Overlay Networks (SSONs). It is a service definite in the wisdom that it is tailored to a definite user for a definite type of service. It has been proposed for multimedia delivery sessions (Al-Oqily, and Karmouch, 2008). To connect the gap between clients and the network and to be able to make available seamless services, the SSONs use network side functions called Media Ports (MPs). MPs make available value further functionality to the overlay such as media caching, media synchronization, media adaptations and routing. With the augmented number of mobile clients and services, SSONs management is becoming more complex and hard to achieve using traditional methods (Khalid, Haye, Khan, and Shamail, 2009), (IBM Corporation, 2006) (Al-Oqily, and Karmouch, 2008).

Runs and controls SSONs involve creating, adapting and terminating them. Since a huge number of them may coexist in the network, creating them and assigning network resources such as MPs are not a trouble-free mission.

Al-Oqily, Karmouch, and Glitho, (2008) proposed Autonomic Overlay (AO) to solve the management complexity and to handle with the augmented claim of creating and deploying new services. Composition and self-organizing schemes have also been proposed by (Al-Oqily, Karmouch, 2008) to create and maintain SSONs in such an autonomic situation.

1.2 Problem Definition

Users in certain locality (domain) usually tend to request the same kind of services. In other words, if they are interested in watching a certain movie or video clip from a certain streaming video server, another close group might be interested in the same video as well. Since those users are close to each other and based on the locality scheme in creating SSONs, the same set of MPs is being reused each time. Based on this scenario, the following problems have been identified:

1. The miss distribution of tasks among MPs is a major cause for network inefficiency (Al-Oqily, Karmouch, 2008).
2. Exploiting local knowledge only when searching for MPs. Such way could result in using a subset of the available MPs and ignoring the rest.
3. The network environment is dynamic; users may leave and join at any time thus achieving load balancing is challenging.
4. A set of MPs is overloaded while another set is idle. This is a load-balancing problem.

MPs have their own resource limitations they can leave and join the network as they are owned by the network provider. In a dynamic network, a single set of MPs may not be able to cope with the ever increased users requirements. Moreover, users at a certain time may be interested in the same type of the services but each one of them has different device and service requirements. As shown in Figure 1.1, this will result in overloaded MPs while the others are less loaded which has a clear negative impact on the network performance and the Quality of Service (QoS). Therefore, it is essential to device ways where load balancing is achieved between the available MPs.

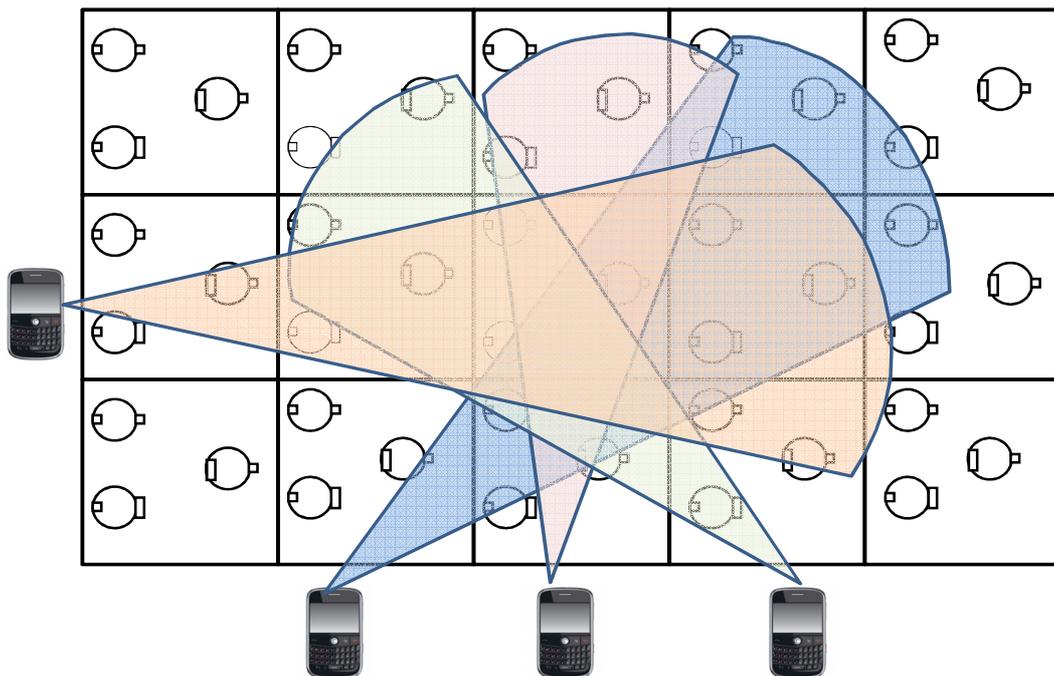


Figure 1.1 MPs at certain area in the network are overloaded while other MPs are less overloaded.

1.3 Objectives

This thesis has the following objectives:

1. Review the state of the art in AC, SSONs, load balancing, and self-load balancing.
2. Deeply study the network environment in order to identify the different parameters that can affect self-load balancing.
3. Design a distributed algorithm that can achieve self-load balancing between MPs.
4. Maintain the users' requested Quality of Service (QoS), and efficiently distribute resources between SSONs to increase and maintain network performance.

1.4 Motivation

Overlay networks are getting great attention due to their flexibility, ability to provide new services, and for their low cost, as they do not require the installation of new devices or equipments. This, in addition to the increased development of mobile applications and the ever-increasing demand on new and novel services designed specifically to meet users' requirements has led to the introduction of SSONs.

MPs have their own resource limitations they can leave and join the network, as the network provider owns them. In a dynamic network, a single set of MPs may not be able to cope with the ever-increased users requirements. Moreover, users at a certain time may be interested in the same type of the services but each one of them has

different device and service requirements. This will result in overloaded MPs while the others are less loaded which has a clear negative impact on the network performance and the service QoS. Network providers install MPs and wish to maximize their profit, while service providers wish to reduce the cost of using MPs and to satisfy users. These are two conflicting goals. Load balancing seems to bring them into an equilibrium state. In one hand, load balancing will utilize MPs efficiently and will distribute the load between them which leaves the network environment stable and problem free, on the other hand, with load-balanced services users' satisfaction can be achieved through providing services that satisfy their requirements. Therefore, it is essential to devise ways with which load balancing can be achieved.

1.5 Contributions

The contributions of this thesis are as follow:

1. A Self-Load Balancing scheme for MPs is presented and discussed. The presented scheme uses an indexing method to build a hierarchical structure known as a distributed quad tree. It encodes the available resources in each MP and presents it as overly incoming edges.
2. A formula to quantify MPs power, i.e the amount of available resources in the MP. It takes into consideration the MPs types and their relation to the MP internal resources.
3. Building an overlay network that maps MPs power for all MPs into incoming edges to these MPs thus facilitating the process of distributing tasks between MPs by following the edges that point to less used MPs.

1.6 Study Boundaries

This work considers a network environment that is distributed, intelligent and autonomic. Where the concept of distributed implies this existent of no global entity. Furthermore, the concept of AC implies that each autonomic entity (node, computer, or MP) is self-managed. This can be interpreted, as "there is no authority higher than the autonomic entity". Besides, it is based on the AONs and focuses on the SSONs as proposed for multimedia delivery sessions. In addition, the main form of communications between each autonomic entity such as MPs is only via messaging (Khalid, Haye, Khan, & Shamail, 2009), (IBM Corporation, 2006) & (Al-Oqily, & Karmouch, 2008).

1.7 Thesis Structure

This thesis includes eight chapters; the preceding chapter gives an introduction about this thesis. Chapter two presented the background regarding AC, A-SSON and the realization of AO then, will shed light in the field of self-load balancing, distributed load balancing, and finally talking about Quad Tree (QT) and Distributed Quad Tree (DQT).

Chapter three presents literature survey and related work for the thesis, showing the related work regarding load balancing in different domains and DQT implementations. Chapter four introduces the methodology used through this thesis including the Comprehensive Literature Survey, a design of a self-load balancing scheme. However, chapter five goes within the implementation of the proposed methodology of the self load balance scheme. Chapter six discusses experimental evaluation of the ability of building self-load balancing scheme for (AO) networks. Chapter seven discusses verification and validation of the designing of a self load Balancing scheme. Chapter eight has conclusions and the future work of this thesis. The last chapter illustrates the references. Algorithms were defined and written using the standard programming code (pseudo code), and were tested and proved using java simulator.

Chapter Two

2

Background

2.1 Overview

This chapter presents the necessary background information that is needed to better understand the types of applications that are targeted and their working environment. In addition to that, it presents all the necessary concepts and terminology required.

2.2 Autonomic Computing (AC)

This section presents a brief review of Autonomic Computing (AC), Autonomic Service Specific Overlay Networks (A-SSON), Autonomic Overlays (AO) self-composition, Quadtree, and Morton Ordering.

IBM Corporation claimed that (IBM Corporation, 2006) the current technology faces the challenge of increased complexity, cost, and heterogeneity. Communications and software technologies are growing rapidly; though, the scale & complexity have grown as well. The growth in system/application development, configuration, and management have begun to overcome existing tools and methodologies, which are rapidly making systems/apps fragile, unmanageable, and insecure. Therefore, there is a great demand to change the ways in which these systems are managed as proposed by (Al-Oqily, Karmouch, & Glitho, 2008). IBM Corporation in 2001 introduced the concept of AC to overcome this complexity. It is inspired from the human biological system. IBM envisioned a computing environment with the capability to manage itself and dynamically adapt to change in accordance with business policies and objectives through a set of self-managing functions such as self-configuring, self-healing, self-optimizing, and self-protecting.

To this end an architectural blueprint for AC has been proposed by IBM and has been revised in 2002 by (Lohman, & Lightstone 2002), 2003 by (Chess, & Kephart, 2003), 2004 by (Hariri, & Parashar, 2004), 2005 by (Berk, Cybenko, & Roblee, 2005), and finally in 2006 by (IBM Corporation, 2006). The blueprint defines concepts and constructs for building self-managing abilities into modern computer systems as well as architectural building blocks of these abilities. The goal of AC is thus to manage complexity (technology manages technology), to reduce cost of ownership (automation reduces human involvement/error) and to enhance other software qualities by (Khalid, Haye, Khan, & Shamail, 2009). AC systems are used to automate the management of a resource (Hardware or software) such as storage, server, network, etc. The resource is monitored for significant events and controlled accordingly. An interface is used for sensing a change in the monitored resources and another is used to enforce a behavior for managed resources to react automatically and manage the targeted environment with minimal human intervention as proposed by (IBM Corporation, 2006), (Hariri, & Parashar, 2004), (Berk, Cybenko, & Roblee, 2005) [2,3,29]. For the management continuity, the four phases control loop as shown in Figure 2.1 is introduced, these are:

1. Monitor: the monitor function provides the mechanisms that collect, aggregate, filter and report details (such as metrics and topologies) collected from a managed resource.
2. Analyze: the analyze function provides the mechanisms that correlate and model complex situations (e.g., time-series forecasting and queuing models).
3. Plan: the plan function provides the mechanisms that construct the actions needed to achieve goals and objectives; it uses policy information to guide its work.

4. Execute: the execute function provides the mechanisms that control the execution of a plan with considerations for dynamic updates.

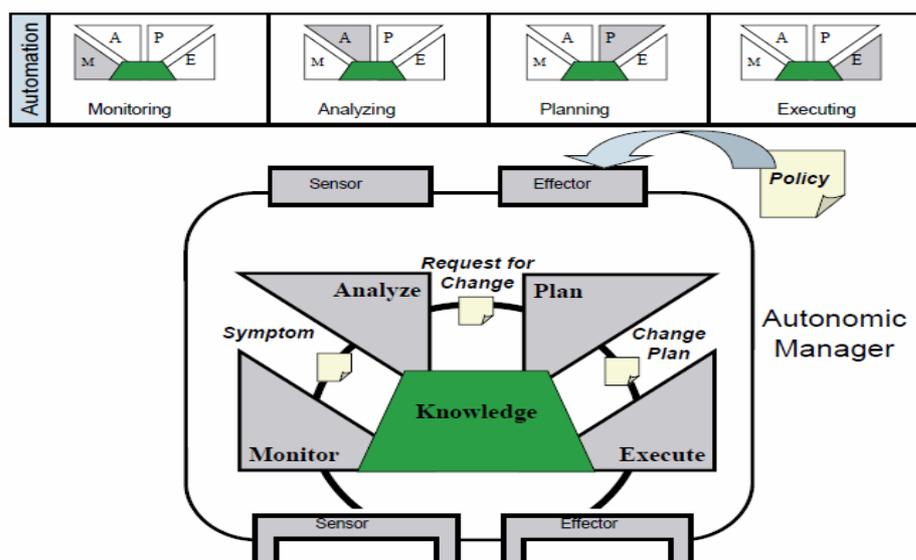


Figure 2.1 Functional details of an autonomic manager (IBM Corporation, 2006).

2.2.1 Autonomic Service Specific Overlay Networks (A-SSON)

The SSONs are overlay networks built and setup for a single service. This overlay is usually customized to meet the user's request. Since different users may need different types of overlays, the setup process is pretty different for each user as proposed by (Al-Oqily, Karmouch, & Glitho, 2008). SSONs have been proposed in the context of the Ambient Networks (ANs) project (Abrahamsson, & Gunnar, 2004). The network environment is dynamically changing and heterogeneous as it consists of potentially a large number of independent, heterogeneous mobile nodes, with spontaneous topologies that can logically interact with each other to share a common

control space, known as the Ambient Control Space (ACS). ANs are also flexible i.e. they can compose and decompose dynamically and automatically, for supporting the deployment of cross-domain (new) services. Thus, the AN architecture must be sophisticatedly designed to support such high level of dynamicity, heterogeneity and flexibility. Thus, SSONs are used in ANs. They are created on-demand according to specific service requirements, they have to deliver, and to automatically adapt services to the dynamically changing user and network context.

Al-Oqily, Karmouch, & Glitho, (2008) proposed in AO draw upon IBM's vision and blueprint described earlier. In their proposal, overlays are viewed as a dynamic organization for self management in which self-interested nodes can join or leave according to their goals. Establishing SSONs involves Resource discovery to discover network side nodes that support the required media processing capabilities. An optimization criterion is needed to decide which nodes should be included in the overlay network, configure the selected overlay nodes; adapt the overlay to the changing network context, user, or service requirements, and join and leaving nodes.

2.2.2 Autonomic Overlays (AO) self-composition

The objective of the proposed architecture (see Figure 2.2) is to create AO that are driven by different levels of policies. Policies are generated at different levels of the autonomic management hierarchy and enforced on the fly. SSON construction uses network side functions, called Media Ports. MPs, thereby, provide the flexibility to modify the content and the services, such as caching, adaptation and synchronization. Every service consists of an allocation of resource amounts to perform a function. In AO, each step imposes a set of minimum requirements. Resource discovery scheme should be: distributed and not rely on a central entity, dynamic to cope with changing

network conditions, efficient in terms of response time which is the difference between the starting time of the query and the arrival time of the reply.

Moreover, message overhead which is the total number of generated messages. Should be accurate in terms of its success rate, which is defined as the number of requests that receives positive responses divided by the total number of queries. Optimization is mapped into a self-optimization that: selects resources based on an optimization criterion (such as delay, bandwidth, etc.). It should yield the cheapest overlay; overlay with the least number of hops, overlay that is load-balanced, low latency overlay network, and a high bandwidth overlay network. The configuration is mapped into a self-configuration and self-adaptation: Self-configuring SSONs dynamically configure themselves on the fly, they can adapt their overlay nodes immediately to the joining and leaving nodes, and to the changes in the network environment. Self-adapting SSONs self-tune their constituent resources dynamically to provide uninterrupted service.

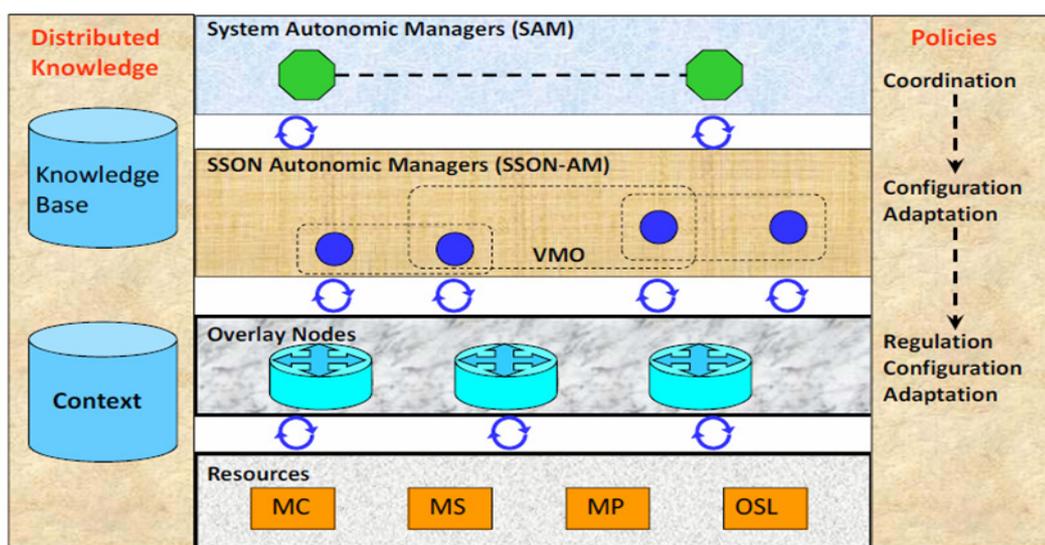


Figure 2.2 Autonomic Overlay Architecture (Al-Oqily, Karmouch, & Glitho, 2008)

As shown in Figure 2.2. The AO architecture is a layered architecture where the lowest layer contains the system resources that are needed for multimedia delivery sessions. MPs are special network side components that provide valuable functions such as special routing capabilities, caching, and adaptation. The Overlay Support Layer (OSL) receives packets from the network, sends them to the network, and forwards them on to the overlay.

The next layer contains the overlay nodes. The second layer Overlay nodes are physical Ambient Network nodes that have the necessary capabilities to become part of the SSON. They consist of a control plan and a user plan. The control plan is responsible for the creation, routing, adaptation, and termination of SSONs, while the user plan contains a set of managed resources. The self-management functions of overlay nodes are located in the control plan. The Ambient manageability interfaces are used by the self-managing functions to access and control the managed resources. The next layer SSON Autonomic Manager (SSON-AM) is responsible for tackling the complexity of overlay management; each SSON is managed by an SSON Autonomic Manager (SSON-AM) that dictates the service performance parameters. This ensures the self-load functions of the services. In addition to this, overlay nodes are made autonomic to self-manage their internal behavior and their interactions with other overlay nodes. So, in this thesis, be sure which system is widely performed. A single SSON-AM alone is only able to achieve self-management functions for the SSON that it manages. If a large number of SSONs in a given network with their autonomic managers are considered, it is observable that these SSONs are not really isolated. On

the one hand, each overlay node can be part of many SSONs if it offers more than one service or if it has enough resources to serve more than one session.

On the other hand, the SSONs' service paths may overlap, resulting in two or more SSONs sharing the same physical or logical link. This will lead to a competition between autonomic managers that are expected to provide the best achievable performance. Therefore, and in order to achieve a system wide autonomic behavior, the SSON-AMs need to coordinate their self-managing functions; this is achieved by using SAMs. The top layer System Autonomic Managers (SAM) manages the different SSON managers by providing them with high-level directives and goals. In other words, SAMs can manage one or more SSON-AMs. They pass the system high-level policies, such as load balancing policies, to the SSON-AMs as shown in Figure 2.3. A SSON-AM can manage one or more overlay nodes directly to achieve its goals and receive goal policies from the SAMs to decide the types of actions that should be taken for their managed resources. Therefore, the overlay nodes of a given SSON are viewed as its managed resources. In addition, they expose manageability interfaces to other autonomic managers, thus allowing SAMs to interact with them in much the same way that they interact with the Overlay Node AMs.

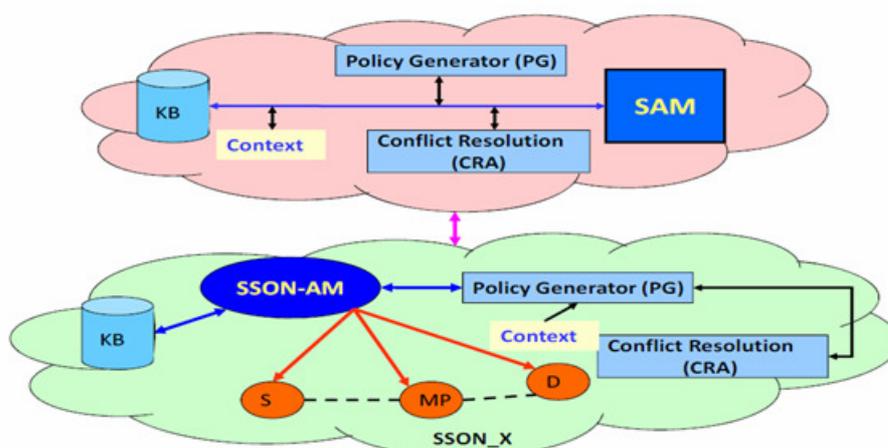


Figure 2.3 The relationships between SAM and SSON-AM (Al-Oqily, Karmouch, & Glitho, 2008).

2.2.3 Quadtree

In this section, will try to cover all aspects of quadtree starting by defining the quadtree. Next, we will go to state how it works. When reviewing the literature a different definition (illustrated below) was found:

A. A quad tree, occasionally quadtree, Q-tree or QT, is a computer science term that refers to a method of organizing data in four quadrants. Databases sometimes use quad trees to store and find their records. This type of organizational structure work especially well to find a particular bit or pixel in a two-dimensional image (Dooley, 2004), (Harwood, Samet & Tanin, 2005), (Gorman, Popinet, Rickard & Tolman, 2010).

B. The quad tree somewhat follows the tree data structure commonly used in the computer science. The normal tree data structure looks like an upside down tree, where a parent node at the top of the tree has one or more children nodes connected to it. Every other node on the tree has one parent node and can have any number of children nodes, including zero (Mir, & Ko, 2006), (Tayeb, Ulusoy, & Wolfson, 1998), (Schuster, & Katsaggelos, 1998).

C. A quadtree is a type of tree structure in which each node has up to four children. Quadtrees are commonly used to divide 2D spaces into smaller areas. They are similar to octrees. The advantage of using quadtrees, like other trees, is that it can be quickly searched. For instance, a tree storing sixteen pieces can be searched in only two search

iterations. A tree storing 64 pieces of data can be searched in only three iterations (Ang, & Samet, 1989), (Oliver, & Wiseman, 1983), (Samet, & Shaffer, 1986).

D. Quadrees are a well-established technique in computer graphics and computer visions for representing a 2D shape. A quadtree is a tree of data structure with each tree node having up to four children. A quadtree node usually represents a square, which can be subdivided into four other squares, which cover the same area. Each leaf node is marked as being part of the object or not (Manolopoulos, Tzouramanis, & Vassilakopoulos 2000), (Dehne, Ferreira, & Rau-chaplin 1991), (Mazumder, 1987).

Now, how the quadtree works will be explained. A quadtree, like other tree structures, has three main components. The first is a "root" or parent node, which represents the head of the tree. Child nodes are all below this root node. Each child has exactly one parent, except for the tree's root node. A "leaf" is a child node, which has no other children. Leaves are usually where the data is stored. Quadtrees are a helpful data structure for dividing a 2D area into smaller pieces as illustrated in figure 2.4.

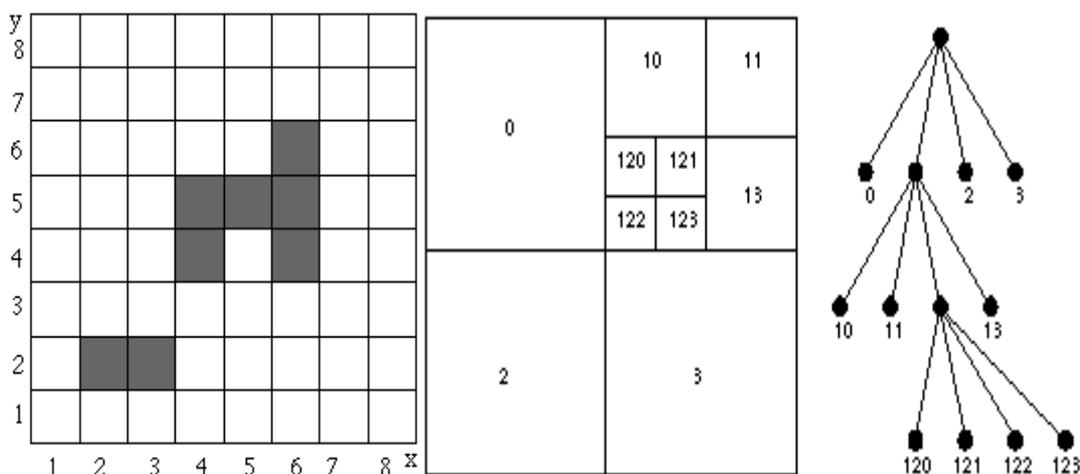


Figure 2.4 Quad Tree space partitioning

In the figure 2.4, the "root" node represents the entire area of the tree. Underneath that node is four smaller areas, zero, one, two, and three. Below each of those nodes is there are four nodes that divide that space further. For instance, one has four child nodes, 10, 11, 12, and 13 and another example node 12 has four child nodes, 120, 121, 122, and 123.

Quadtrees are the majority frequently used to divide a two dimensional space by recursively subdividing it into four quadrants or regions. The regions may be square or rectangular, or may have random shapes. A quadtree is a tree data structure in which every node in the interior node has up to four children. This data structure was named a quadtree by Raphael Finkel and J.L. Bentley in 1974 (World News Website, 2011). A related partitioning is also known as a Q-tree. Each and every one forms Quadtrees share some ordinary features, they decompose space into adjustable cells; each cell has a maximum capability. When maximum capability is achieved, the cell splits, and the tree directory follows the spatial decomposition of the Quadtree (Aboulnaga, & Aref, 2001), (Harwood, Samet, & Tanin, 2005), (Eppstein, Goodrich, & Sun, 2005), (Lario, Antonijuan, & Pajarola, 2002), (Eisenstat, 2011).

2.2.4 Morton Ordering

Frens, & Wisey (1999) conclude that Morton order was introduced in 1966 by G. M. Morton. In the mathematical analysis and computer science, Z-order, Morton order, or Morton code is a space-filling curve which maps multidimensional data to one dimension while preserving locality of the data points (Alexander, Frens, Gu, & Wise, 2001).

To enable us to know and determine where the (X,Y) coordinate lie in the distributed quadtree or vice versa where a distributed quadtree indexes lie in coordinate space. To achieve this addressing scheme using the bit interleaving as follow:

1. take a cell's row and column numbers
 - A. e.g. row Y= 2, column X= 5
2. write the row and column numbers in binary notation, using bits
 - A. 2 = 010; 5 = 101
3. interleave the bits, starting at the left and working to the right, and taking a row bit first

0	1	1	0	0	1
---	---	---	---	---	---

The blue color and shading refer to the Y axis (2) and the black refer to the X (5) axis

4. The result is 011001, now replacing zero by 00, 1 by 01, 2 by 10, and 3 by 11.

2.3 Load Balance

In this section, survey and present Load Balancing for Self Load Balancing, Analytical Load Balancing Algorithm and Load Balancing types and policies are presented.

Various definitions of load balancing have been introduced. For example (Alakeel, 2010) define the load balancing as “the process of redistributing the work load among nodes of the distributed system to improve both resource utilization and job response time while also avoiding a situation where some nodes are heavily loaded while others are idle or doing little work”. Again (Alakeel, 2010) defines it as “the mechanism that enables jobs to move from one computer to another within the distributed system, this creates faster job service e.g., minimize job response time which is the difference between the starting time of the query and the arrival time of the reply. And enhances resource utilization”. There is a more interesting definition proposed by (Eager, Lazowski, & Zahorjan 1986) where they define it as "the process of roughly equalizing the workload among all nodes included in the distributed system. It strives to produce a global improvement in system performance". In this manner, load balancing goes one-step further than load sharing which only avoids having some nodes idle in the distributed system when other nodes have too much work. This definition better captures the essence of the problem trying to solve, thus will be adopted through this proposal. Load balancing has been proposed to solve problems in various disciplines. In the following sections, load balancing in Grid, wireless, and overlay networks is reviewed.

2.3.1 Self Load Balancing

While reviewing the literature, not a lot about self-load balancing was found for this era. For instance, a lot of work was based on central entity while others try to predicate the load status and not all of them cope with this environment.

Flatebo, Datta, & Bourgon, (1994) proposed a Self-stabilizing Load Balancing Algorithm. This algorithm cares for the receiving and completion of jobs as perturbations to the system. A long time ago, the system stabilizes and the local variables permit jobs to be thrown to the least loaded node. Once the least loaded node begins the job, its load will be increased which would yield the modification in the variables. The variables will dramatically update and reflect to be converging so that jobs can go ahead to a least loaded node. The proposed solution yields an extra overhead because of the message passing to update the different lists and variables.

Meng, Qiu., Zhang, & Zhang, (2010) proposed a design of distributed and Autonomic Load Balancing (ALB) for self-organization. Long Term Evolution (LTE). This proposed effort distributed Autonomic Flowing Water Balancing Method (AFWBM) which function can be described as monitoring, analyzing, optimization and implementation. For that, AFTWBM can detect their load conditions depending on self-monitoring actions. To attaining ALB for LTE Radio Access Networks (RAN) by employing AFWBM modules in Evolved Universal Terrestrial Radio Access (E-UTRAN) NodeB (eNBs), overload conditions can be detected by eNBs, then handover

hysteresis margin HOM will be adjusted and handover actions will be triggered to balance load. This work is restricted for a specific domain.

For that as declared in chapter one, the section on problem definition, the only key is to provide a self-load-balancing technique maybe by redesigning an existing one to cope with the restricted new network environment or by proposing a new one.

2.3.2 Analytical Load Balancing Algorithm

Performance Analysis of Load Balancing Algorithms has been proposed by (Sharma, Sharma, & Singh, 2008). They present the performance analysis of various load balancing algorithms based on different parameters, considering two typical load balancing approaches static and dynamic. The analysis shows that static and dynamic (both types of algorithm) can have advancements as well as weaknesses over each other. To decide which; type of algorithm to be implemented will relay on type of parallel applications to solve. The main reason for their work is to assist in proposing new algorithms in the future by studying the behavior of various accessible algorithms. The comparison of various load-balancing algorithms on behalf of the different parameters is shown in Table 2.1 below.

Table 2.1 Performance Analysis of Load Balancing Algorithms (Sharma, Sharma, & Singh, 2008).

Parameters	Round Robin	Random	Local Queue	Central Queue	Central Manager	Threshold
Overload Rejection	No	No	Yes	Yes	No	No
Fault Tolerant	No	No	Yes	Yes	Yes	No
Forecasting Accuracy	More	More	Less	Less	More	More
Stability	Large	Large	Small	Small	Large	Large
Centralized/Decentralized	D	D	D	C	C	D
Dynamic/Static	S	S	Dy	Dy	S	S
Cooperative	No	No	Yes	Yes	Yes	Yes
Process Migration	No	No	Yes	No	No	No
Resource Utilization	Less	Less	More	Less	Less	Less

Load balancing algorithms work on the belief that in any situation workload is assigned, during compile time or at runtime. The above comparison shows that static load balancing algorithms are more stable in contrast to dynamic and it is also simple to guess the behavior of static, but at the same time, dynamic distributed algorithms are always measured better than static algorithms.

Chhabra, & Singh, (2006) made a qualitative parametric comparison of load balancing algorithms in a parallel and distributed computing environment. Decreases in hardware costs and advances in computer networking technologies have led to increased interest in the use of large-scale parallel and distributed computing systems.

One of the biggest issues in such systems is the development of effective techniques/algorithms for the distribution of the processes/load of a parallel program on multiple hosts to achieve goal(s) such as:

1. Minimizing execution time.
2. Minimizing communication delays.
3. Maximizing resource utilization.
4. Maximizing throughput.

Researches using queuing analysis and assuming job arrivals following a Poisson pattern, have shown that in a multi-host system the probability of one of the hosts being idle while other host has multiple jobs queued up can be very high. Such imbalances in system load suggest that performance can be improved by either transferring jobs from the currently heavily loaded hosts to the lightly loaded ones or distributing load evenly/fairly among the hosts. The algorithms known as load balancing algorithms, help to achieve the above said goal(s). These algorithms come into two basic categories - static and dynamic. Whereas Static Load Balancing algorithms (SLB) take decisions regarding assignment of tasks to processors based on the average estimated values of process execution times and communication delays at compile time, Dynamic Load Balancing algorithms (DLB) are adaptive to changing situations and take decisions at run time.

The objective of their work is to identify qualitative parameters for the comparison of the above said algorithms. This comparison work in tabular form is shown in Table 2.2 below.

Table 2.2 Qualitative Parametric Comparison of Load Balancing Algorithms Chhabra, & Singh, (2006).

Load balancing Parameters	SLB Algorithms	DLB Algorithms
1.Nature	Static	Dynamic
2.Associated overhead	Lesser overhead	More overhead
3.Resource Utilization	Lesser Utilization	More Utilization
4.Processor Thrashing	No Thrashing	Substantial Thrashing
5.Preemptiveness	Non-preemptive	Preemptive and Non-preemptive
6.Predictability	More Predictable	Lesser predictable
7.Adaptability	Less adaptive	More Adaptive
8.Reliability	Less	More
9.Response Time	Less	More
10.Stability	More	Less
11.Other Issues	Determining process execution time at run time	Developing techniques to reduce comm. Overhead

2.3.3 Types of load balancing and policies

By means of the huge developments in computer machinery in addition to the ease of use of many distributed systems, the difficulty of load balancing in distributed systems has increased a superior awareness and significance (Casavant, & Kuhl, 1988), (Goscinski, 1991). As a result, a huge quantity plus diversity of investigation has been carried out in a challenge to resolve this trouble. Classification of load balancing algorithms in distributed systems are reported by (Wang, & Morris, 1985). Solutions to the load balancing problem are divided into two main approaches depending on whether a load balancing algorithm bases its decisions on the current state of the system or not: static and dynamic. As described Load balancing algorithms can be classified into two categories: static or dynamic. In static algorithms, the decisions related to load balance are made at compile time when resource requirements are estimated. Multicomputers with dynamic load balancing allocate or reallocate resources at runtime based on no prior task information, which may determine when and whose tasks can be migrated (Slimani, & Yagoubi, 2006), (Alakeel, 2010).

In the presents networks nowadays all systems have common factors such as heterogeneity, scalability, adaptability. Additionally, request load and resource administration are two vital purposes presented at the service level of the grid software infrastructure. Several load balancing types or strategies and algorithms have been proposed to improve the global throughput of these software situations; workloads have to be equally planned between the accessible resources. Most strategies were developed in a state of mind assuming homogeneous set of sites linked with homogeneous and high-speed networks.

2.3.3.1 Load Balancing Types (LBT)

There are different kinds of LBT that are used widely; Selecting an appropriate load balancing strategy permits CSIS to offer balancing service according to the load capacity of each server (Fuse source website, 2010), (Apache camel website, 2010).

1. Round Robin balancing: In a round-robin algorithm, the IP sprayer assigns the requests to a record of the servers on a rotating basis. The earliest request is allocated to a server selected randomly from the cluster, so that if an additional IP sprayer is concerned, not all the original requests depart to the same server. For the subsequent requests, the IP sprayer goes after the circular sort to forward the request. Formerly a server allocating a request; the server is going out to the end of the record. This remains the servers uniformly assigned. In computing, "round-robin" illustrates a technique of selecting a resource for a task from a list or record of accessible ones typically for the ideas of load balancing. Such that may be a distribution of incoming requests to an amount of processors, worker threads, or servers. At the same time, as the basic algorithm, the scheduler chooses a resource pointed to by a counter from a list, later than which the counter is incremented and if the ending is reached, returned to the start of the list. Round-robin selection has an optimistic attribute of avoiding starvation, since every resource will be sooner or later chosen by the scheduler, excluding may be not fitting for some applications where similarity is desirable, for instance when handing over a process to a CPU or in link aggregation. The services requested from a consumer are distributed to each server in a cluster in turn. This kind of equilibrium algorithm is appropriate for all the servers in a server cluster having the similar software and hardware configure; and the average resource practice of each request is relative to the same efficiency. This algorithm belongs to a static load balancing.

2. Weighted Round Robin balancing: According to unlike processing ability of each server, each server has a defined related weighted value in order to admit the service request of corresponding weighted value. This class of balancing algorithm can guarantee the high-performance server that can acquire more accesses, in the meantime avoid the server of low-performance from overloading, or go to overcapacity. This algorithm belongs to the static load balancing.

3. Random balancing: Distributing the requests from a network to a server in a server cluster is random. In a random allocation, the HTTP requests are allocated to any server chosen randomly amongst the cluster of servers. In such situation, individual of the servers might be allocated several additional requests to process, whilst the other servers are sitting idle or unused. However, on average, every server acquires its share of the load outstanding to the random selection. This algorithm belongs to the static load balancing.

4. Weighted Random balancing: This balancing algorithm is similar to Weighted Round Robin algorithm; nevertheless, the loads are distributed randomly depending to weighted value. Weighted Round Robin is a highly developed version of the round robin with the reason of reducing the shortages of the simple round robin algorithm. In situation of a weighted round-robin, individuals can allocate a weight to each server in the group so that if one server is skilled of handling two times as much load as the other, the powerful server obtains a weight of two. In such situations, the IP sprayer will assign two requests to the powerful server for each request assigned to the weaker one. This algorithm belongs to the static load balancing.

5. Flash DNS balancing: The service requests of client achieve IP address of the server from the first to the last determining domain name. Usually, by means of this balancing algorithm, the diverse load balancing equipment (DNS) dispersed in dissimilar geographical locations accept the same request of determining domain name from the same client. After that, the domain name is determined into IP address by different DNS and returns to the client. The client will admit the server's service of which IP address arrives first and ignores other's service of which IP address arrives behind schedule. This balancing strategy is appropriate for the situation of global load balancing, however, it is inappropriate for the local load balancing. This algorithm belongs to static load balancing.

6. Least Connection balancing: In attendance, there may be bigger varieties in serving time of each one request. If assuming the Round Robin or Random balancing algorithm, the quantity of service connection on every server might turn out to be more dissimilar alongside with time trailing off, and it will origin load balancing halt. Least Connection balancing exercises a list to record the quantity of connections of every server. When a fresh service request arrives, it will be distributed to the server of which connection number is the least. This type of balancing algorithm is appropriate for the request that requires long time service such as FTP service. This algorithm belongs to the static load balancing.

7. Response Time balancing or least response time Load balancer: Load balancing tools start sensing command or request such as Ping packet to every server in the server cluster, after that distribute service request to the server of which response time is the shortest. This Response Time balancing algorithm can find enough mirrors for the usability of servers, but the least time just capital the shortest communication time among load balancing equipment and server. When a Load balancer is configured to employ the least response time technique, it chooses the service with the least number of active connections and the least average response time. The response time also called Time to First Byte, or TTFB is the time period between sending a request packet to a server and receiving the first response packet back. This algorithm belongs to the dynamic load balancing.

The approaches of aforementioned load balancing exist not enough subjectively assigning and scheduling loads. Except “response time balancing”, aforementioned strategies are unidirectional, static and subjective, and they cannot reproduce the true load conditions of the server and load ability in time. Temporarily, they cannot notice server fault and recognize fault tolerance.

2.3.3.2 Load balancing policies

Load balancing algorithms can be defined by their implementation of the following policies or strategies (Karatza, 1994):

1. Information policy or strategy: identifies what workload information to be grouped, when it is to be grouped and from where it is to be grouped. Information strategy is the information heart of a dynamic load-balancing algorithm. It is in charge for afford location and transfer strategies at each node with the essential information required to build their load balancing decisions. A complicated information strategy maintains each and every one node of the distributed system updated on the universal system state but produces extra load traffic and thus enlarges the overhead generated by the algorithm. For that reason, there is a trade-off between the total of information swap and the occurrence of the swap of this information.
2. Transfer Strategy: taking into account that significant limitations such as job execution time, size, I/O, and memory necessities are not recognized until the job is carried out; selecting a job for load balancing is not a simple mission. Furthermore, a single advance has been attempted in order to treat with this absent piece of information. One advance to load balancing creates job transfer decisions separately of the job's properties. In this method, a job is transferred if the queue length at the local node goes beyond a definite threshold or else, the job is executed locally. Shortly it finds out the suitable epoch to start a load balancing operation.

3. Resource type policy: categorizes a resource as a server or receiver of tasks according to its availability condition.

4. Location Strategy: Solitary of the main decisions achieved by a load-balancing algorithm is the selection of a target node for a job transferred for load balancing. This decision symbolizes the sole purpose for load balancing: an over loaded node attempts to find a lightly loaded node to assist in performing a quantity of its jobs. This decision is carried out by the location strategy. The choice of a remote node is based on the current workload exists at that node. Shortly, Location policy utilizes the outcome of the resource type policy to discover an appropriate collaborator for a server or receiver.

5. Selection policy: describes the responsibilities that must be transferred from overloaded resources to most idle resources.

6. The achievement of a load-balancing algorithm relies on: the steadiness of the quantity of messages, little overhead, maintained environment, low down cost update of the workload and tiny mean response time, which is an important amount for a user (Badidi, 2000). It is also necessary to determine the communication rate persuaded by a load balancing function.

Chapter Three

3

Related work

3.1 Load Balance

In this section, a survey and the related work for Load Balancing in different domains and Distributed Quadtree Implementations are presented:

3.2 Load Balancing in different domains

While reviewing the literature, it was found that each type of load balancing algorithm is suitable for one domain and not applicable for others unless, if was adapted to compatible domains in Grid Networks, Wireless networks, and Overlay networks.

3.2.1 Load Balancing In Grid Network

A grid network is a type of computer network component of a number of computer systems linked in a grid topology. Grid computing offers a homogeneous interface to heterogeneous and bodily-distributed resources, each and every one connected over a high-speed network. Current Grid implementations are geared toward scientific projects, which require large amounts of compute, storage, and network resources. These operations remain fairly static over time and as such demand long-lived, photonic network connections by (De Leenheer, et al., 2005). The fundamental inspiration behind Grid computing is to interconnect and utilize available storage, processor, or memory subcomponents of distributed computing systems to work out larger problems more professionally.

The benefits of Grid computing are cost savings, enhanced business quickness by decreasing the time-to-market (delivering actual results), and improved group effort and sharing of resources among departments or institutions. Several economic and business features are causal to the heightened interest in the development and deployment of Grid computing. Based on the Internet and E-commerce, today's society is inundated with data. As the available data repository grows bigger and wider, the window of opportunity for capturing and translating the obtainable data into information shrinks quickly.

Singh, & Suri, (2010) proposed a dynamic load balancing algorithm for a decentralized grid model. The algorithm considers load index as a decision factor for scheduling of tasks within a cluster and among clusters (grid). The decentralized grid system model is a set of clusters; each cluster contains Coordinator Nodes (CN) together with the multiple Worker Nodes (WN) but they have different processing powers. The tasks that are generated by users are sent to the CN where a decentralized job scheduling approach is used. CN collect jobs from users in their cluster and place them in a global task set and then compute the utilization factor for its cluster. Once CN with high utilization factor receive a new job, the Grid Information Center (GIC) is consulted to provide an alternative cluster with low utilization factor to which the CN will transfer the new job. Therefore, the algorithm periodically collects load information of clusters and sends it to the GIC entity. Although job scheduling is decentralized, the GIC entity is a central entity thus represents a single point of failure to the system since load balancing has a great impact on resources' performance (Singh, & Suri, 2010).

Deldari, & Salehi, (2006) try to provide a more accurate load measurement/estimation method, which relies on the time needed for executing current jobs (instead of number of current jobs). Then they are proposing a new load balancing method, as an agent, based on this new measurement/estimation policy. As application performance prediction provides the important functionality that enables the grid load balancing capabilities. The agent is equipped with a performance prediction toolkit called PACE. The PACE toolkit is used to supply this ability for both the local schedulers and the grid agents. The main components of the PACE toolkit include Application Tools (AT), Resource Tools (RT), and an Evaluation Engine (EE). The PACE evaluation engine can mingle application and resource models at execution time to produce performance data e.g. total execution time. In the Agent-based Resource Management System (ARMS), agents obtain their resource capabilities using PACE and exchange them with their neighbors periodically. An agent advertises its load information only to its neighbors, for the purpose of scalability needed in the grid. It is possible to attach load characteristics of the nodes to this exchanging of information. Here, they consider the total execution time (gained through PACE evaluation engine in each node), average of the job arriving rate and job completion rate (considering the number of arrivals/completions in a certain fixed interval of time in each node) as the load information.

This information helps to provide a more accurate measurement as well as estimation for load as follows:

1. The agents estimate the current load of their neighbors.
3. Then the agent computes the average load on its neighboring agents. An agent calls itself “overloaded” if its load is greater than the average load of its neighbors.
3. Agents in the neighboring set, whose estimated load is less than the estimated average load by more than a threshold, form an active set.
4. The sender each time finds a member of the active set which has the most profit (result in less response time) sends to it jobs (extra load).

However, load can be spread to a large area after many steps of equalization over a period of time. It is probable that an under loaded agent situated in an active set of two or more overloaded agents simultaneously. In these circumstances, overloaded agents may send their extra load to the under loaded agent at the same time and make the under loaded agent, overloaded. Hence, this condition causes instability for the proposed method. To this end, they use a locking technique to avoid these situations. Therefore, each agent only sends its load information to one requester, and does not respond to any other agent at the same time. This continues until the agent is dismissed by the requester. The Grid computing environment is a collaboration of spread computer systems where customer jobs can be executed on any home or remote computer. Many troubles live in the grid environment.

In a computational Grid, as resources are in nature distributed and positioned at different sites, the job transfer time from one spot to another site is a very important factor for load balancing. In addition, the communication latency is very big for the WAN through which Grid resources are usually connected. Furthermore, due to network heterogeneity, the network bandwidth varies from one link to another. For this

reason, the job transfer cost cannot be ignored when making a job migration decision. In addition, since the resources are heterogeneous, jobs that have to be assigned to processors according to their performance are needed.

Saravanakumar, & Prathima, .(2010) proposed, adaptive, and decentralized load balancing algorithm for computational Grid environments, called the Load Balancing on Arrival (LBA). The processors that are directly connected to a processor constitute its buddy set. It is assumed that each processor has knowledge about its buddy processors and the communication latency between them, and load balancing is carried out within buddy sets only. It may be noted that two neighboring buddy sets may have a few processors common to each set and use three performance metrics of relevance at three different levels:

1. At the job level, they consider the ART of the jobs processed in the system as the performance metric. If N jobs are processed by the system (Saravanakumar, and Prathima, 2010), then

$$ART = \frac{1}{N \sum_{i=1}^M (\text{Finish } i - \text{Arrival } i)} \quad (1)$$

2. At the system level, they consider the total execution time as the performance metric to measure the algorithm's efficiency. It indicates the time at which all N jobs get executed.
3. At the processor level, they consider the resource utilization as the performance metric. It is the ratio between the processor's busy time to the sum of the processor's busy and idle time (Saravanakumar, & Prathima, 2010):

$$U_i = \frac{\text{Busy } i}{(\text{Busy } i + \text{Idle } i)} \quad (2)$$

Accordingly, the objective is to propose well-organized load balancing algorithms to reduce the ART of the jobs for computational Grid environments. This algorithm will influence load balancing by watchful estimation of the job arrival rates, CPU processing rates, and loads on the processor. Moreover, they take into account the resource heterogeneity, network heterogeneity, and job migration cost before a load balancing decision. The process of parameter estimation and the way in which load balancing is carried out is described below.

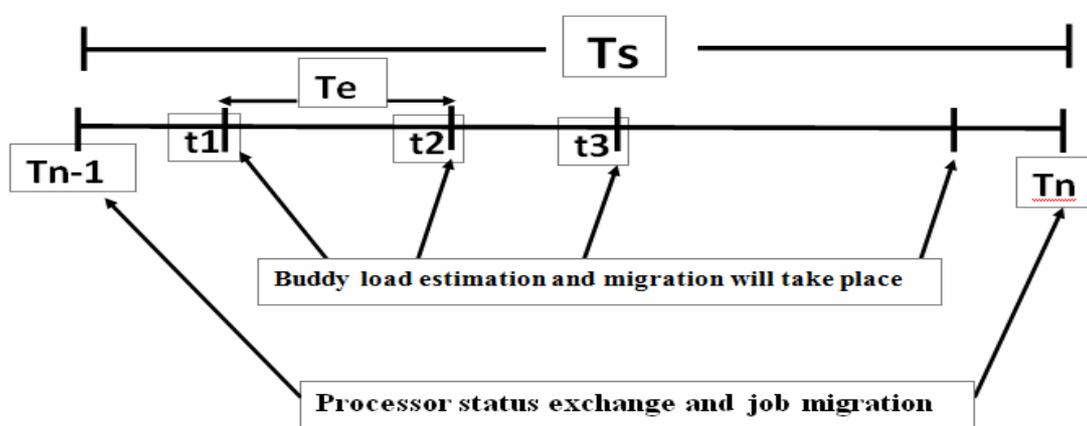


Figure 3.1 Estimation and Status Exchange intervals (Saravanakumar, & Prathima, 2010).

At each periodic interval of time, T_s called the status exchange interval; Each P_i in the system calculates its status parameters, as follows:

1. The estimated arrival rate.
2. The service rate.
3. Load on the processor.
4. Exchanges of its status information with the processors in its buddy set.

The instant at which this information exchange takes place is called a status exchange instant. In Figure 3.1, T_{n-1} and T_n represent the status exchange instant.

Each P_i calculates its status information at status exchange instant T_{n-1} . Each status exchange period is further divided into equal subintervals called estimation interval T_e . These points are known as estimation instants. In figure 3.1, t_1, t_2, \dots, t_{m-1} represent the estimation instants. Each P_i calculates the estimated load on its buddy processor P_k . The status exchange instants and the estimation instants together constitute the transfer instants. The decision to transfer jobs and actual transfer of jobs are done at transfer instants.

3.2.2 Load Balancing in Wireless network

Wireless networks refer to any type of a computer network that is wireless (without any type of wires), and is commonly associated with a telecommunications network whose interconnections between nodes are implemented without the use of wires. Wireless telecommunications networks are generally implemented with some type of remote information transmission systems. These systems use electromagnetic waves, such as radio waves, for the carrier and this implementation usually takes place at the physical level or "layer" of the network. In this context, Ad hoc networks are wireless, decentralized networks that consist of a set of identical nodes to form a network. Ad hoc is a Latin phrase, which, literally, means "For this". The network is ad hoc because it does not rely on a pre-existing infrastructure, Instead, all nodes almost identical in their capabilities move freely and independently and communicate with other nodes via wireless links, participate in routing by forwarding data for other nodes and so the determination of which nodes forward data is made dynamically based on the network connectivity.

Ad hoc network may be reasonably represented as a set of clusters by grouping together nodes that are in close proximity with one another. Such network consists of ordinary node and cluster heads (a leader). Clusterheads are ordinary nodes selected sometimes based on random algorithm to form the backbone of the wireless network, and have more responsibility to route packets or to distribute routing information or both within the same cluster or between cluster heads in another cluster. Nodes in ad hoc networks are mobile. For that powered by batteries, Communications or transmissions cause the batteries to be depleted. Therefore, the communications should be kept to a lower boundary to avoid a node dropping out of the network rashly. Because the clusterheads are involved in every communication, the battery's life depletes earlier than other battery's node in the clusters. Therefore, there is a need to distribute the load or to balance the load between other nodes as mentioned in (Amis, & Prakash, 2000).

It is assumed that the MAC layer will mask unidirectional links and pass only bidirectional links. Beacons could be used to determine the presence of neighboring nodes. The Multiple Access with Collision Avoidance (MACA) protocol utilizes a Request To Send/Clear To Send (RTS/CTS) handshaking to avoid collision between nodes.

Another type of wireless networks is the Wireless Sensor Networks (WSNs). It consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion, or pollutants. The developments of wireless sensor networks were motivated by military applications such as battlefield surveillance and are now used in many industrial and civilian application areas, including industrial process monitoring and control, machine health monitoring, environment and habitat monitoring, healthcare applications, home automation and traffic control.

Israr, & Awan, (2007) proposed a new cluster based routing algorithm that exploits the redundancy properties of the sensor networks in a try to address the usual problem of load balancing and energy efficiency in the WSNs. Any WSNs face challenges and issues in clustering such as:

1. Network deployment: Node deployment in WSNs is either fixed or random depending on the application.
2. Heterogeneous network: the WSNs are not always uniform. In some cases, a network is heterogeneous consisting of nodes with different energy levels.
3. Network scalability: When a WSN is deployed, some time new nodes need to be added to the network in order to cover more area or to prolong the lifetime of the current network.
4. Uniform energy consumption. Transmission in WSNs is more energy consuming compared to sensing. Therefore, the cluster heads which perform the function of transmitting the data to the base station consume more energy compared to the rest of the nodes.
5. Multi-hop or single hop communication: The communication model that a wireless sensor network uses is either single hop or multi hop.

6. Cluster Dynamics: Cluster dynamics means how the different parameters of the cluster are determined for example, the number of clusters in a particular network.

As thus, they propose Multi hop Clustering Algorithm for Load Balancing (MCLB) (Israr, & Awan, 2007). It consists of two distinct parts, the setup part and the steady part. During the setup part, Cluster Heads and temporary cluster heads are elected followed by the steady part. The steady part is the data transmission part and is longer than the setup part. In the setup part, the algorithm first filters all the nodes in the network of which area coverage is covered by its neighbors. Because of this operation, the network is divided into two layers the top layer and the bottom layer. The top layer comprises of nodes whose sensing area is completely covered by its neighbors along with cluster heads, whereas the bottom layer comprises of the rest of the network nodes. Because of operations, part of the algorithm is the same as that of each in which a set of cluster heads are chosen at random. These cluster heads then broadcast an advertisement message. Depending on the message strength, each node then decides to which cluster head it belongs. This part uses the CSMA MAC protocol and during this period all the nodes are listening. The selection of the cluster head is dependent on the probability. During each cycle, the cluster head selection is random and is dependent on the amount of energy a node has left and its probability of being not a cluster head during the last rounds. After this, the data transmission part starts. In this part, all nodes transmit data using TDMA based scheduling.

When all the nodes within the cluster finish sending data, the cluster head performs some computation on it and sends it to base station using multi hop communication involving temporary clusters and other clusters heads.

3.2.3 Load Balancing in Overlay network

Bridgewater, Boykin, & Roychowdhury, (2007) proposed a Balanced Overlay Networks (BON), a new decentralized load-balancing advance that codes the balancing algorithm in the developing construction of the graph that connects the resource-bearing nodes. A BON is scalable, self-organized, and relies only on local information to build job assignment decisions. New jobs are allocated to a node by a random walk on the graph which not only samples the graph preferentially but also selects the highest-degree node that was visited on the walk. Each node's idle resources are relative to its degree, so this approach works very fine when a network is not loaded further than its clipping point. When a BON is clipped, the relationship between load and in-degree breaks down, but the balancing performance remains quite good due to the so-called "power of two choices" in a ball-bin load balancing. Based on previous theoretical results and extensive simulation results, BON is seen to be efficient and practical. Added ongoing work on this difficulty includes geographical alertness extensions using more difficult walk objective functions. Lastly, it should be distinguished that this is only one probable way to code information about a network in its topology; other distributed algorithms may benefit from using a graph state to a favoritism node selection.

Dalal'ah (2006) proposed a sliding policy for load balancing. The policy grouping a definite number of neighboring nodes to execute load balancing. Upon the achievement of a certain period, the groups are to be span by changing each group one place to the right, therefore create different groups.

This policy (sort of clustering) not only reduces the load balancing overheads, but also could be utilized as a backbone by any load balancing policy. The proposed load balancing strategy always comes together, and tends to be in a steady state in an insignificant processing time. The load status and the locations of the nodes concerning the system's topology are irrelevant to the load balancing process. The new algorithm can be constantly applied to any distributed system, even if it is heavily loaded, since the rate of scheduling is very low due to the highly reduced number of communication. This is pulled off by dropping dramatically the overheads acquired from attached information tables, message passing, job thrashing, and response time. Two methods of grouping the nodes were introduced; the first is to group the nodes in couples while the other one is to group the nodes into triples. The overhead branch from computations is reduced dramatically in both methods. Therefore, the number of communication (message passing) is not any more an important issue, since it turns to be rigid with a small number of messages and when the utilization of the system is maximized. The proposed policies guaranteed the distributed system to be scalable.

3.3 The Implementation of DQT for Networks

DQT is simple infrastructure and stateless environment, the DQT demonstrates a charming flexibility to the appearance of node breakdowns.

3.3.1 DQT for Spatial Querying in WSNs

Demirbas, & Xuming, (2007) presented an in-network querying infrastructure, called a distributed quad-tree (DQT) structure, appropriate for employing in actual globe WSN deployments. The DQT convinces a distance-sensitive querying as well as a well-organized information storage in a network. The DQT building is local and does not need any communication.

Furthermore, due to its simple infrastructure and stateless environment, the DQT demonstrates a charming flexibility to the appearance of node breakdowns. The DQT is acquiescent to an organism extended to arbitrary and compound queries than the binary version “is there an event?” queries exist at this point. In that case, since the queries are arbitrary, the information advertisement cannot be hopeful of all queries, and only a review of sensor data should be accumulated for energy-efficiency purposes. As such, for a declaration of queries there may be several corresponding alternatives that need to be discovered excluding that they may not assure the query and may result in back-tracking and model-based query optimization techniques. The stateless nature of DQT formulates that it is flexible to topology alters. In fact, it may possibly expand DQT to give a location service for mobile ad hoc networks. The thought is to redo a query until it grabs up with the mobile objective. Even if a target node may shift during the query implementation and guides to a fail to see the query when invoked from this new location closer to the target node. It will have an improved chance to catch up to the target node due to the distance-sensitivity property in DQT.

In building his model, the author takes for granted that the WSN motes sit down on a two dimensional plan and their coordinates (X,Y) are made accessible to themselves. The network is separated into grid cells while inserting a DQT over the network. A level one box in DQT represents the minimum cell area in the DQT construction. The authors suppose that all the WSN motes inside a level 1 box are within one hop distance. According to his jargon, a mote refers to a physical MPs node, while a “node” refers to a virtual DQT node, such as level one box. The cost of querying an event is calculated as the number of hops passed through from the querying mote to a mote that holds an advertisement about the event.

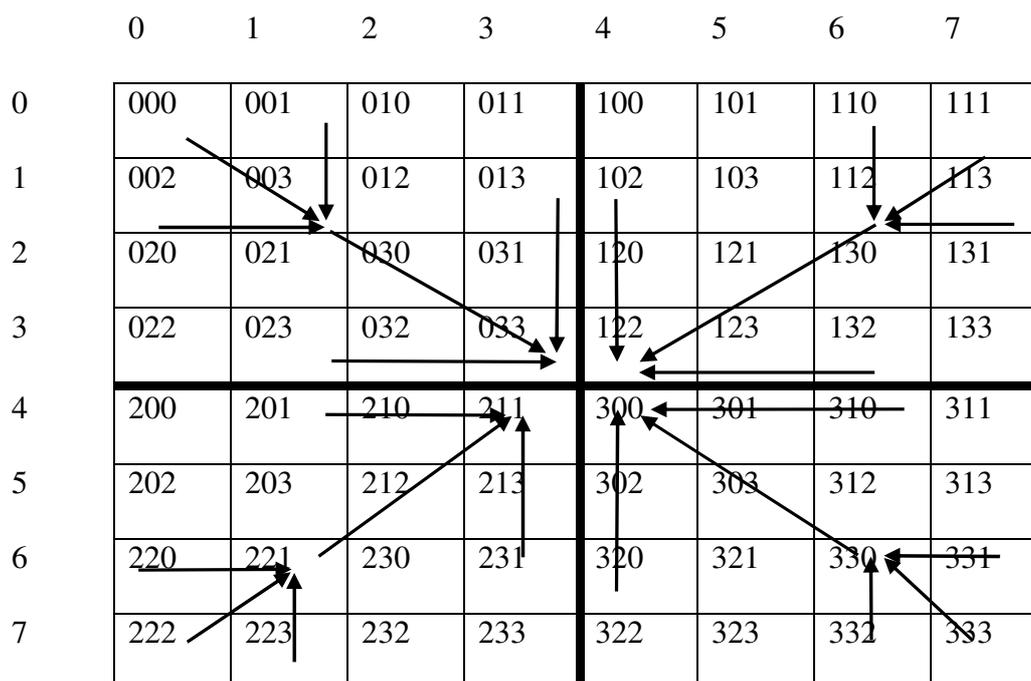


Figure 3.2 DQT Structure and Construction (Demirbas, & Xuming, 2007).

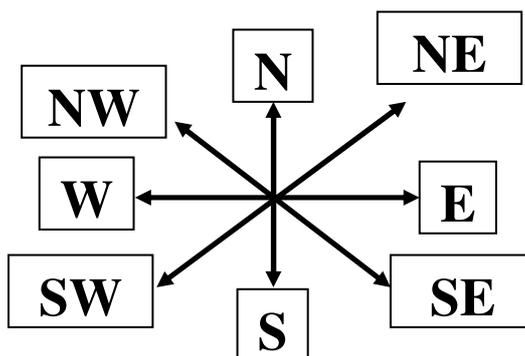


Figure 3.3 Different directions (Demirbas, & Xuming, 2007).

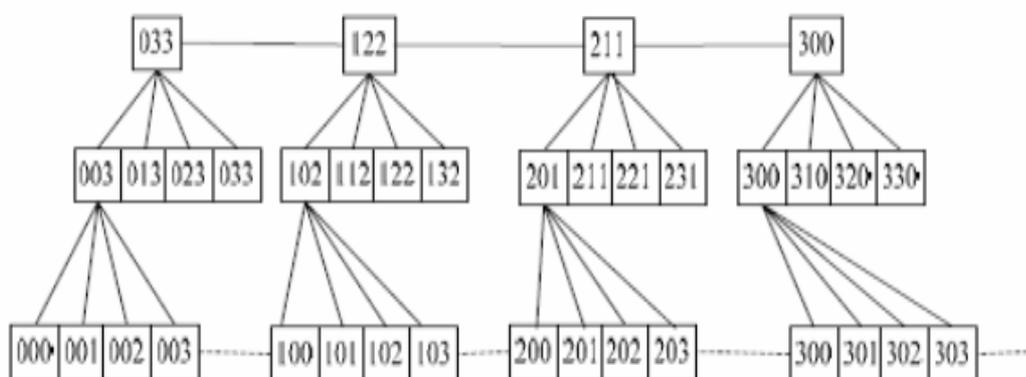


Figure 3.4 Node addressing and tree structures (Demirbas, & Xuming, 2007).

For building DQT, they utilize an encoding trick. In this encoding, every level 1 box in the construction is assigned an ID, which uniquely identifies a region. The length of the ID is equivalent and identical to the number of levels. Employ this addressing method to care for the location information of a node. Due to the technique building level 1 box, this scheme is self-governing of the number of nodes, but relies on the division levels. Figure 3.4 illustrates the addresses of the nodes in a region with three levels. In all level of partition, a node is assigned as a clusterhead node of the region. The clusterhead is always its own child in lower levels. The clusterhead at every level division is statically assigned to be the closest node to the geographic center place of the entire network. For

instance, in level one division, node 003 is selected as a clusterhead for 00 regions, because it is closer to the center than nodes 000, 001 and 003. In the same way, node 033 is selected as level two clusterhead, as it is closer to the center than level two nodes 003, 013, and 023. Therefore, the node closest to the center of the entire network in each sub partition is selected as the parent node of that sub partition. The advantage of such a choice is to pass up rearward links. For instance, in figure 3.4, node 000 propagates the query to its root node 033 by first contacting parent node 003, then 003's parent 033. A short path is achieved since there is no rearward link on the querying trail. A DQT node can fit in to different levels in the hierarchy depending on its place. If a node is a member at level k, it is also a member at all levels fewer than k. Indicate a node p's parent as p.parent & children as p.child. The neighboring nodes are called siblings, which are denoted as p.sibling. The author Mapping from localization to DQT addressing: Each node in DQT can calculate the DQT address of the level one partition it resides in from its X,Y coordinates easily. Let (Xs,Ys) at NW and (Xe,Ye) at SE be the two endpoints of the area where DQT should be overlaid. (Demirbas, and Xuming, 2007). Assuming DQT has i levels. The region of each level 1 box of division is (w*1), where width

(3)

Then DQT address of a node(X,Y) can be calculated as claimed (Demirbas, & Xuming, 2007):

$$DQT_{addr} = \left[\left\lfloor \frac{X - X_s}{w} \right\rfloor (mod 2) \right] + \left[\left\lfloor \frac{Y - Y_s}{l} \right\rfloor (mod 2) \right] * 2 \quad (4)$$

The mappings compute the X and Y address individually, and next add them together. This can be verified this from figure 3.4, for instance, node ID 033 is obtained by adding 011 and 022, and node ID 332 is obtained by adding 110 and 223. The motivation that the second term in the DQT address computation is multiplied by 2 is because the Y addresses pace by two for every increment in DQT addressing scheme. Given this mapping, any node can locally compute its DQT address based on its coordinates (X,Y). Besides the DQT address, each node also maintains its (X,Y) coordinate address. By using the above encoding trick and assigning DQT addresses for DQT nodes, can start constructing the DQT structure.

3.3.2. A Quadtree-Based Data Dissemination Protocol for WSN with Mobile Sinks:

Mir, & Ko, (2006) proposed a wireless sensor network (WSN) made of a number of small sensors that are closely positioned to watch and work together with the physical world where each sensor can partially monitor the large topography. Environment monitoring application differs very much with one common aim of detecting and reporting on the event of interest to the sink. The author proposed an efficient and simple, Quadtree-based data broadcasting protocol for large scale wireless sensor networks that chains both stimulus and sink mobility. By construction the data propagation process self-governing of each other's current location. Quadtree-based Data Dissemination (QDD), a familiar hierarchy of data forwarding nodes is created by Quadtree based partitioning of physical space into following quadrants. A source node computes a set of rendezvous points by one

after another partitioning the sensor network space into four uniformly sized logical quadrants, and fires data packets to the nodes closer to the centric of each following partition.

The mobile sink follows the same approach for the data query packet propagation. It starts from querying the direct rendezvous node and continues until it finds the required data report which results in lower overhead. For example, a sensor node S with locality (X_s, Y_s) , gets the complete sensor network space N as the root of a Quadtree, after that reasonably partitions N into four equivalent sized quadrants. Each of these four quadrants North West (NW), South West (SW), North East (NE) and South East (SE) match up to a child of N , in that order. The root N stands for the entire network space, particulate by as claimed by (Mir, & Ko, 2006)

$$\mathbf{N.X_{(LB)}} = \mathbf{1} \quad , \quad \mathbf{N.Y_{(LB)}} = \mathbf{1} \quad , \quad \mathbf{N.X_{(UB)}} = \mathbf{2^k} \quad , \quad \mathbf{N.Y_{(UB)}} = \mathbf{2^k} \quad (5)$$

Where $(\mathbf{N.X_{LB}}, \mathbf{N.Y_{LB}})$ are coordinates for lower left corner (lower bound) and $(\mathbf{N.X_{UB}}, \mathbf{N.Y_{UB}})$ are coordinates for upper right corner (upper bound) of a square, respectively. If P is the parent of child quadrant C , then values for $C.X_{LB}$, $C.Y_{LB}$, $C.X_{UB}$ and $C.Y_{UB}$, depends upon whether C is the NW, SW, NE, or SE child of P . Next, each quadrant is considered as a split parent and divided into further four sub-quadrants. By Knowing the present position of node S (X_s, Y_s) , this procedure is repeated for each quadrant, until node S leftovers are the only node in a sub-quadrant (the leaf cell). This routine requires a relationship at each partition level, to test out if the current sub-quadrant is the leaf cell. For example, as proposed by (Mir, & Ko, 2006) if node S is in the NW quadrant of parent P (i.e., $C = P.NW$), after that:

$$\{ \mathbf{S} \in \mathbf{P.NW} : ([\mathbf{C.X}]_{,LB} \leq X_{,s} \leq [\mathbf{C.X}]_{,UB}) \& ([\mathbf{C.Y}]_{,LB} \leq Y_{,s} \leq [\mathbf{C.Y}]_{,UB}) \} \quad (6)$$

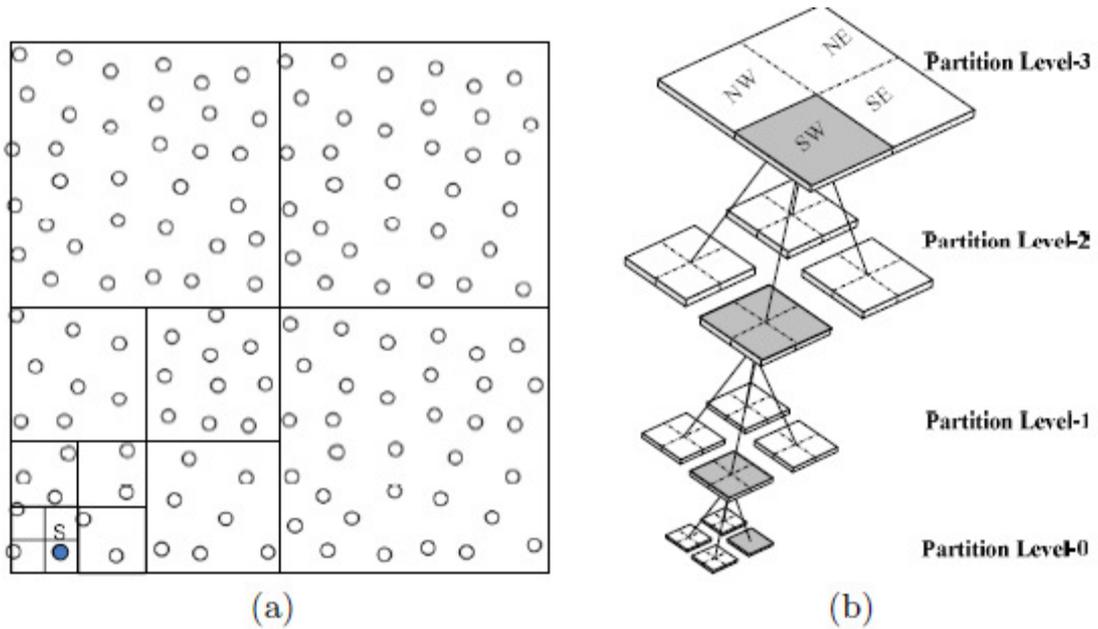


Figure 3.5 (a) Sensor network space N partitioning. (b) QT representation. (Mir, & Ko, 2006)

His method is in the mode a source node disseminates data. Upon sensing a stimulus, source node S executes a reasonable partitioning of sensor network field as above. For every partition level I represented by a square with lower corner and upper corner values set to $(\mathbf{i.X}_{(LB)}, \mathbf{i.Y}_{(LB)})$ & $(\mathbf{i.X}_{(UB)}, \mathbf{i.Y}_{(UB)})$ respectively, it calculates a list of central point's called rendezvous points $\mathbf{RP}(\mathbf{X}_1(i), \mathbf{Y}_1(i))$ given as proposed by (Mir, & Ko, 2006):

$$X_{(i)} = i \cdot X_{(LB)} + \frac{i \cdot X_{(UB)} - i \cdot X_{(LB)}}{2}, \quad Y_{(i)} = i \cdot Y_{(LB)} + \frac{i \cdot Y_{(UB)} - i \cdot Y_{(LB)}}{2} \quad (0 < i \leq k)$$

(7)

Figure 3.5 (a) illustrates the rendezvous points computed by node S and the data passing process. It begins from its current place as the first rendezvous point (0th level) and forwards data packet to the immediate rendezvous point (1st partition level) using geographical greedy forwarding. If S is not itself the neighboring node to the immediate rendezvous point, it searches in its neighbor's table for a neighbor that is closest to that point and forwards the packet to it.

Every node in turn recurs this process; until a node locates that, no other node in its neighborhood is nearer than itself. Now this node becomes the rendezvous node. Even though forwarding data packets, each rendezvous node maintains a local table so that the copy entries related to the same data packet can be known and then dropped. In addition, all table entry contains an expire field that decides how lengthy that entry would remain suitable before it is unnecessary from the table.

Chapter Four

4

Design of

Algorithms and

Methods Used

4.1 Overview

A comprehensive literature survey assists to build a case for this research and review of the literature relevant to our topic, which is a Self-Load Balancing in Autonomic Overlay Networks. Moreover conducting the literature survey in-depth helps, us to identify our problems in more accurate details based on a strong scholarly foundation. Moreover, it gives us a hand to design our own scheme.

We start by reviewing the materials that formulate the background to get adequate information to understand and to have a chance to find some interesting solution. For that, we started by Autonomic Computing (AC), Autonomic Service Specific Overlay Networks A-SSON, the recognition of Autonomic Overlay AO, a Quadtree (QT) and spatial indexes. Then moved one-step forward to realize the load balance in different aspects such as definition, types, strategies, and implementation in different areas for example in wired or wireless networks. Next, we go a further step by reviewing the Self Load Balancing proposed in the literature. Finally, we conducting an in-depth the literature survey to be able to identify gaps in our research.

4.2 The Strategy of Designing a Self-Load Balancing Scheme

1. Employ the DQT, for partitioning the space, in distributed manner, to cope with our era.
2. Identify each region of the network by a unique ID for that employing the Morton order to index each single box by a unique ID.
3. Adopt a new formula to calculate the power and take into perspective the different services that MPs provide.
4. Develop a method, to encode the computation resources, in the topology of the network.
5. Demonstrate a greedy mechanism to select the more powerful MPs to execute the incoming jobs.

4.2.1. Scheme Formulation

As described in the previous chapter, on section problem definition here again we will re-mention the main problems that face us through proposing such scheme in steps:

1. The essence of the load balancing problem focuses on distributing the load from a resource perspective and completely neglects the user's (or request) perspective.
2. Transparency of load balancing algorithms is an assumption that has not been treated as a requirement for traditional distributed systems applications.
3. Finally, the concept of AC implies that each autonomic entity (node, computer, or MP) is self-managed. That can be interpreted, as "there is no authority higher than the autonomic entity". This is vitally important because it means that traditional load balancing algorithms may not be applicable in the era of AC.

While reviewing, the literature we didn't find a suitable formula for calculating the power because our environment is completely different and all of the previous literature talks only about the so-called normal node which refers to a normal personal computer. In the end, we were convinced that in order to overcome this problem we must formulate our formula to calculate the power for each MP separately, according to the service provided by each one of it. For that, we should review the conditions that must be met and not neglect the calculation of power for each MP.

4.2.2 Computing the Processing speed

Below we identify the main factors affecting processing speed (Kitchen table computers website, 2010), (e-learning website, 2011). The circuitry design of a CPU determines its basic speed, but several additional factors can make chips already designed for speed work even faster.

1. CPU's registers:

The size of the registers is sometimes called the word size which indicates the amount of data with which the computer can work at any given time.

2. the memory:

The amount of RAM in a computer can have a profound effect on the computer's power.

3. data bus:

The bus refers to the paths between the components of a computer.

A. The data bus: An electrical path connects the CPU, memory, and the other hardware devices on the motherboard.

B. The address bus

It is a set of wires similar to the data bus that connects the CPU and RAM and carries the memory addresses. The reason why the address bus is important is that the number of wires in it determines the maximum number of memory addresses.

4. Cache Memory: It is similar to RAM, except that it is extremely fast compared to the normal memory, and it is used in a different way. It helps to reduce the time – consuming operation of CPU which is moving data back and forth to RAM.

5. Math coprocessor:

Passing math operations to a math coprocessor; the math coprocessor is a chip that is specially designed to handle complicated mathematical operations.

6. Bandwidth

Measured in bits, the bandwidth determines how much information the processor can process in one instruction. If you were to compare data flow to the flow of traffic on a highway, then clock speed would be the speed limit, and the bandwidth would be the number of lanes on the highway.

4.2.3 MPs Power Calculation

Each MPs has a resource list as declared in table 4.1 below. Any MPs can provide different services such as caching, adaptation, synchronization and routing. For that each service has resource limitations as shown in table 4.2 below and the percentage values for different services.

Table 4.1 show the MP resources and the percentage value for each service provided

Mediaport ID	RAM	C.P.U	Access speed of hard disk system (bus speed)	Bandwidth
01	2 G	3 G	400	100 Mbps

Table 4.2 show the MP resource and the percentage value for each service provided

Media Port	(RAP)	(CPP)	(BSP)	(BAP)
Services	RAM percentage	C.P.U percentage	Bus Speed Percentage	Bandwidth percentage
(CAC) Caching	15%	15%	40%	30%
(ADA) Adaptation	30%	30%	20%	20%
(SYN) Synchronization	25%	30%	15%	30%
(ROU) Routing	25%	25%	15%	35%

Before starting our calculation, the above information in the resources list table. 4.1 was converted to gigabyte (RAM, C.P.U, BUS SPEED, and BW).

4.2.4 Standardize the parameters Measurements

There are many types of parameters used deferent measurements, such as:

- A. Central Processing Unit in GHZ
- B. Cache Memory in GB
- C. Data Bus in MB
- D. Bandwidth in Mbit

Therefore the standard measure used in this research is Gigabytes, so all other measures have been converted into Gigabytes otherwise it is stated.

4.2.5 Local knowledge calculation:

Each MediaPort can calculate its power depending on a local knowledge and according to the following formula:

$$G_i(ni) = (P1 * CPU_i(ni)) + P2 * MEMORY_i(ni) + P3 * BS_i(ni) + P4 * BANDWIDTH_i(ni)) * Z \quad (1)$$

Where $P1+P2+P3+P4 = 1$ and corresponds to a weighted value shown in table 4.2.

$G_i(ni)$ = refers to the power or the load ability of MediaPort.

$CPU_{(ni)}$ = refers to the CPU power.

$MEMORY_{(ni)}$ = refers to the memory speed.

$BS_{(ni)}$ = refers to the access speed of hard disk system (bus speed).

$BW_{(ni)}$ = refers to the bandwidth or throughputs.

Z = refers to the load ability of MediaPort = 100%

$P1$ = refers to the percentage of c.p.u , $P2$ = refers to the percentage of memory

P3= refers to the percentage of access speed of hard disk sys. ,P4 = refers to the percentage of band width

4.2.6 The main formula:

$$\mathbf{[" interest]}_i((ni)) = ((P1 * \mathbf{[CPU]}_i((ni)) + P2 * \mathbf{[MEMORY]}_i((ni)) + P3 * \mathbf{BS}_i((ni)) + P4 * \mathbf{[...]}_i((ni))) \quad (1)$$

4.2.7 Formula for each type:

(2)

$$\mathbf{[" Adaptation]}_i((ni)) = ((0.30 * \mathbf{[CPU]}_i((ni)) + 0.30 * \mathbf{[MEMORY]}_i((ni)) + 0.20 * \mathbf{BS}_i((ni)) + 0.20 * \mathbf{[...]}_i((ni))) \quad (3)$$

$$\mathbf{[" Synchro.]}_i((ni)) = ((0.25 * \mathbf{[CPU]}_i((ni)) + 0.30 * \mathbf{[MEMORY]}_i((ni)) + 0.15 * \mathbf{BS}_i((ni)) + 0.30 * \mathbf{[...]}_i((ni))) \quad (4)$$

$$\mathbf{[" Routing]}_i((ni)) = ((0.25 * \mathbf{[CPU]}_i((ni)) + 0.25 * \mathbf{[MEMORY]}_i((ni)) + 0.15 * \mathbf{BS}_i((ni)) + 0.35 * \mathbf{[...]}_i((ni))) \quad (5)$$

In this way, can conclude the following formula:

$$\mathbf{load} = \frac{\mathbf{Job}}{\mathbf{power}} \implies \mathbf{power} = \frac{\mathbf{jobs}}{\mathbf{load}} \implies \mathbf{j_bs} = \mathbf{load} * \mathbf{power} \quad (6)$$

4.2.8 Proof

Jiang & Zhang (2007) suggested a formula to calculate power for each server. The goal is to achieve load balancing between multiple servers. The load status of servers can be reflected by the utilization ratio of resources including CPU, memory, hard disk system, and network. They defined the formula as follows

$$C(t) = ((1-a(t))*P + (1-b(t))*R + (1-c(t))*D + (1-d(t))*N) * Z \quad (7)$$

where

$a(t)$ = the utilization ratios of CPU.

$b(t)$ = the utilization ratios of memory.

$c(t)$ = the utilization ratios of hard disk system.

$d(t)$ = the utilization ratios of network at the time of t .

P, R, D and N is corresponding weighted value.

$$P + R + D + N = 1$$

Z is defined as the total load ability of the server.

MPs are similar in that they are computers and share the same properties as servers.

Even though the functionality of the server is completely different than the functionality of the MP, we can adopt the same procedure proposed by Jiang & Zhang, 2007.

However, we assume the values $CPU_{(t)}$ is the value $(1 - a(t))$, $MEMORY_{(t)}$ is the

value $(1 - b(t))$, $BS_{(ni)}$ is the value $(1 - c(t))$, and $BW_{(ni)}$ is the value $(1 - a(t))$. Moreover, the weighted values $P1$, $P2$, $P3$, and $P4$ are P , R , D and N respectively. Finally, we assumed the $u((ni))$ which is referred to the power or the load ability of MP. As a result, we get the following formula:

$$[("interest)]_{i((ni))} = ((P1 * [CPU]_{i((ni))} + P2 * [MEMORY]_{i((ni))} + P3 * BS_{i((ni))} + P4 * BW_{i((ni))}) * u((ni)) \quad (1)$$

$[("interest)]_{i((ni))}$ expresses the approximate processing capability of the MP in $[t, t + \Delta t]$,

and the load distributor assigns tasks to node according to $[("interest)]_{i((ni))}$.

To be able to compare one MP power to others, $[("interest)]_{i((ni))}$ has to be represented as a ratio value. This can be obtained by dividing the current MP power by the highest MP power in the network. This is called the MOST POWERFULL INTEREST MEDIAPORT. The most powerful MP is available in the network and treated as a benchmark.

Since we will map MP power into incoming edges, we have to divide the power by a value. This should result in a maximum number of incoming edges, as we should not allow an infinite or non-deterministic one. Therefore, the parameter Z in formula (7) is assumed to be 100 in formula (1) and we divide the result by 12.5. This will allow us to have a number of incoming edges that does not exceed 8.

Finally, it is worth noting that the hard disk value in formula (7) has been replaced by the bus speed in formula (1). This is because for the different services that the MPs provide, the bus speed is more important. Moreover, the hard disk size will dominate the result of the formula that will make other parameters invisible.

4.2.9 Encoding the resources in the topology of the overlay:

After adopting a new formula to calculate the power we need a method or way to encode the computation resources in the topology of the networks where this way should be directly proportional to the power of each MPs and inversely to the workload for each. The fundamental thought Self Load Balance Distributed Quad Tree (SLBDQT) is that the workload properties of a distributed computing system can be encoded in the topology of the network that connects the computational MPs. In representation language, an edge in a SLBDQT network represents a certain unit of unused capacity on the MPs to which the edge refers. On the one hand, when the MPs resources are being worn out, their in-degree will turn down. On the other hand, when the MPs obtainable resources are increasing, its in-degree will get higher, and this is only one possible way to encode information about a network in its topology.

In our proposed model, we make the maximum in coming edges equal eight edges because we want a very cheap way to build the overlay networks and the minimum in coming edge equal four edges to maintain and always produce a strongly connected component where we can sample and visit any MPs with a minimum effort and less time.

4.2.10 Selection of powerful MPs:

When an incoming job enters the network, the highest power of MPs to execute this job is needed. For that, we need to employ a greedy strategy to assign any incoming jobs to be run at powerful MPs. Greedy algorithms produce good solutions on some mathematical problems greedy choice properties can make whatever choice seem the best at the moment and then solve the sub problems that arise later. The reason behind that is to maintain the quality of service (QoS) to the client by reducing the execution time for this job.

Chapter Five

5

Proposed

Self Load

Balance Scheme

5.1 Introduction

To achieve the goal, which is A Self-Load Balancing overlay for Locality of autonomic entities; we use the dived and conquer principle with an indexing method, to build a hierarchical structure known as a distributed quad tree. Besides, we adopt a new formula to calculate the power for each MP, we take into consideration the MPs type and the service it provided, and finally employ a mapping technique between the power for each MP and the incoming edges to these MPs.

As described in the previous section, we are interested in using the DQT structure where the cluster head node has four children (in this proposed algorithm it is called cluster head, parent, or ancestor interchangeably where the children called producer, or leafs). The second interest is the indexing technique, which is used to enable to identify each geographical location by a unique ID. The indexing is at the heart of DQT. The leaves of this sub-tree correspond to the computing elements of a geographical location, and the root is a virtual node associated to the geographical location as each geographical location contains at minimum or at least four MPs.

5.2 Partitioning the geographical location of the network

A Quadtree is a hierarchical data structure that has advantages for geographic data storage. As, a two-dimensional geometric region is recursively decomposed into four quadrants. Every one of the four quadrants turns out to be a node in the quadtree. A superior quadrant is a node at an upper hierarchical level of the quadtree, and lesser quadrants come into sight at lower levels. The benefit of this organization is that the standard breakdown provides for straightforward and capable data storage, retrieval, and processing. The straightforwardness branches the facade of the geometric regularity of the breakdown into squares, and the effectiveness obtained by storing only those nodes containing data of significance.

5.3 Building the DQT

This network is divided into grid cells while inserting a DQT over the network. The suppose that the MPs which consist of about 30% of the whole network (the rest is a normal node which consist 70% of the network) sit on a two dimensional, and their (X,Y) coordinates are prepared accessible to themselves . A single box in DQT makes up the minimum cell vicinity in the DQT structure. It is assume that all nodes inside level one box are within one hop distance. The cost of querying an event is measured as the number of hops traveled from one node to another. The dissimilarity between DQT and the centralized quad-tree is that the first does not need a root of the tree. The four nodes in the first level service as the root. In order to build DQT an indexing deception is utilized. In this indexing, each cell in the structure allocates an ID, which uniquely identifies a region or location. The length of the ID is equal to the number of levels. We apply this addressing system to safeguard the position information of a node. As the centralized quad-tree, DQT is a hierarchical structure. In each level of partition, a node is assigned as a parent node of the region. The parent is always its own child in lower levels. A DQT node may belong to different levels in the hierarchy depending on its location. If a node is a member at level A, it is also a member at all levels less than A.

5.3.1 Building the DQT dependent only on local knowledge

As illustrated above each node in DQT, we can calculate the DQT address of the level one partition it residing in from its (X,Y) coordinates easily. DQT uses a local building instead of a bottom-up construction to reduce communication cost during initial constructions. A static and local scheme that uses the address of the box suffices for calculating every level parent and neighbors. Each node may have neighbors at North, South, East, and West. In the below sections, it is demonstrated how this

proposed algorithm is composed from an accumulation to achieve A Self-Load Balancing. As said in the beginning, this DQT will be a product of $2^n * 2^n$ where n is integer number > 1 . In the example in table 5.1, this networks product of $2^3 * 2^3 = 64$ node, and because $n= 3$ then should partition this network into 3 level equal to n , besides the number of digits for each index again equal $n = 3$. For that, the first bit refers to level one the first and second digits refer to level 2, the three digits together refer to level 3 which is the lowest level in the DQT as illustrated in figures from 5.21-5.32.

5.3.2 Node Type

Given a square two dimensional array with size $N * N$ where n is a power of 2

1. Normal node Type = 0.
2. MP caching type = 1.
3. MP synchronization type = 2.
4. MP routing type = 3.
5. MP adaptation type = 4.

5.3.3 Indexing the geographical location (Spatial Indexing)

G. M. Morton introduced it in 1966 (Frens, and Wisey, 1999). It is called Morton order, Morton code, or Z-order, which is a space-filling curve that maps multidimensional information to one dimension while maintaining the locality of the information points. The z-value of a point in multidimensions is designed by interleaving the binary representations of its (X, Y) coordinate values. Once the data are sorted into this ordering, any one-dimensional data structure can be used such as binary search trees. The Z-ordering can be used to efficiently construct quadtrees and related

higher dimensional data structures (Alexander, Frens, Gu, and Wise, 2001) As Figure 5.1 shows.

Indexing indexing any cell c in the grid to a unique index i

Begin procedure Indexing (c) {
 Convert the X coordinate of the node into binary using Log₂N bit representation
 Convert the Y coordinate of the node into binary using Log₂N bit representation
 Grouping by Interleaving (putting Y coordinator in an odd position and X coordinator in an even position) started from most left bit of y and grouped them in a single chunk.
 Get the index from the chunk by replacing each pair in the chunk as follows
 00 by 0 , 01 by 1 , 10 by 2 , 11 by 3 }
 Call PrentDetection (c)
 End Procedure

Figure 5.1 Procedure Indexing.

The figure below shows the two dimensional array which is a product of $2^3 * 2^3$ with integer coordinates $0 \leq x \leq 7$, $0 \leq y \leq 7$.

Table 5.1 The original node distribution in a grid

	0	1	2	3	4	5	6	7
0	0	1	4	5	16	17	20	21
1	2	3	6	7	18	19	22	23
2	8	9	12	13	24	25	28	29
3	10	11	14	15	26	27	30	31
4	32	33	36	37	48	49	52	53
5	34	35	38	39	50	51	54	55
6	40	41	44	45	56	57	60	61
7	42	43	46	47	58	59	62	63

To index any geographical location for example node 37 in bold and underlined

1. First, Get the (X,Y) coordinates as such as (X coordinate= 3,Y coordinate= 4)
2. Change the (X) coordinate from decimal to binary X coordinate= 3 in decimal change it to binary = 011
3. After that, Change the (Y) coordinate from decimal to binary Y coordinate= 4 in decimal change it to binary = 100,
4. Next Get the result for (X,Y) coordinates in binary which is (011,100)
5. Interleave the result starting from (Y) and from left to right

1	0	0	1	0	1
---	---	---	---	---	---

As shown, the odd number represents the Y coordinates (in red color and shading) and the even number represents the X coordinates (in black color),

6. Next collect them in a single chunk

1	0	0	1	0	1
---	---	---	---	---	---

7. Then, Change the pairs as follows (00 by 0, 01 by 1, 10 by 2 and 11 by 3)

2	1	1
---	---	---

8. Now, get the result that is the index value

2	1	1
---	---	---

Interleaving the binary coordinate values yields binary z-values as shown in the illustrative example below. Connecting the z-values in their numerical order produces the recursively Z-shaped curve.

Table 5.2 represents the Morton Order after applying it over table 5.1 above

	0	1	2	3	4	5	6	7
0	000	001	010	011	100	101	110	111
1	002	003	012	013	102	103	112	113
2	020	021	030	031	120	121	130	131
3	022	023	032	033	122	123	132	133
4	200	201	210	211	300	301	310	311
5	202	203	212	213	302	303	312	313
6	220	221	230	231	320	321	330	331
7	222	223	232	233	322	323	332	333

5.3.4 Parent Detection

Each node in the network can be determined, if it is a parent or a leaf node by checking its index. If it ends with zero values, then it is a parent, otherwise, it is a leaf node. For example in table 2 above, the index 333 is not ending with zeros value so, it is a leaf but in the same table index 220 ending with a zero value for so, it's a parent. As Figure 5.2 shows.

```

ParentDetection check index  $i$  is parent  $p$  or leaf  $l$ 

Begin procedure ParentDetection ( )
If (last digit in  $i$  ==0) then {
     $i$  is parent  $p$ 
} Else {

If (last digit in  $i$  != 0) then {
     $i$  is leaf  $l$ 
}

Call MyParentDetection ( )

End Procedure

```

Figure 5.2 Procedure ParentDetection.

5.3.5 My Parent Detection

Each node in the network has only one parent and wants to know its parent. To do this, first parent detection procedures is called, if it is a leaf node, then the last digit in the index is changed to zero; otherwise (if it is a parent then) the digit that proceeds the last zero value in the index is changed to zero. For example in table 5.2, the index 111 is a leaf node and to get its parent change the last digit to zero value then gets its parent

which is 110. However, in the same table, index 330 ends with a zero value so, it is a parent and a parent of a parent are 300. As Figure 5.3 shows.

```

MyParentDetection change leaf  $i$  to index  $i$  to be parent  $p$ 

Begin procedure MyParentDetection ( )
If (index  $i$  == leaf  $i$  ) then {
Change the last digit in the index  $i$  to 0 to be parent  $i$ 
} Else {
If (index  $i$  == parent  $i$  )then {
Change the digit proceeding the last 0 value in the index  $i$  to 0
}

Call ParentLevelDetection ( )

End Procedure

```

Figure 5.3 Procedure MyParentDetection.

5.3.6 Parent level Detection

As said above, each node in the network has just one parent but this parent has many indexes. Each index is for each level for that, each parent wants to know its different index according to the level it appears on. A number of successive zeros determined the level for each parent node. For example, in table 5.2 the index 111 is a leaf node and to get its level, we check the number of zeros the index contains. We have null zeros value so, it is a leaf node then it is at level one in DQT, besides it is at level 3, for that, this number refers to the number of digits for this index. Take another index which is 220. It is a parent and has just one zero value for that, add to it one which equals 2. Then it is at level two in the DQT. Besides, it is at level 2 for that, this number refers to the number of digits for this index which equals 22. As Figure 5.4 shows.

```

ParentLevelDetection determined where index  $i$  level  $lv$ 

Begin procedure ParentLevelDetection ( )
If (  $i == \square$  ) then {
     $\square$  at  $lv - 1$ 
    new  $\square = \text{original } \square$ 
} Else {
If (  $i = \square$  ) then {
    let Z be the numbers of successive 0 in the last digits of the  $\square$ 
    let L be the number of  $lv$ 
    L = Z + 1
}
}

Call ChildrenDetection ( )

End Procedure

```

Figure 5.4 Procedure ParentLevelDetection.

5.3.7 Children Detection

Each parent has just 4 children, besides it can calculate its children. For example, the parent index 330 can know its children by substituting the zero value in the index by number from 1-3 to get its children. So parent 330 children are (330,331,332 and 333) . As Figure 5.5 shows.

```

ChildrenDetection determined whose my children  $mchild$  for parent  $p$ 

Begin procedure ChildrenDetection ( )
If (  $i == \square$  ) then {
    For (S=0,S<=3,S++)
        Let C be the  $\square$ 
         $mchild = \text{concatenate C with ( S)}$ 
        Add  $mchild$  to ChildrenList
    Next
}
}

```

```

}
Call RootDetection ( )
End Procedure

```

Figure 5.5 Procedure ChildrenDetection.

5.3.8 Root Detection

Each tree has one root but this DQT has four roots why? Because it is interested in building a DQT which does not need any central point. Besides, working on a quad and so, this algorithm always produces four roots.

How can it be determined if the index is a root or not?

That can be known by numbering the successive zeros in the index. If it is more than or equal to $n-1$ then it is a root. For example, index 200 the number of successive zeros equal 2 and then $(n) = 3$ then $3-1=2$ then it is a root. Again the index (000) the number of successive zeros equal 3 and $(n) = 3$ then it is a root. As Figure 5.6 shows.

```

RootDetection. determined index i is root rot

Begin procedure RootDetection ( )
If (i >= N-1) then {
    i is rot
}
Call BrotherRootDetection ( )
End Procedure

```

Figure 5.6 Procedure RootDetection.

5.3.9 Brothers Root Detection

Because the root always lies on level one and the index in level one encodes by just one digit, each root node can compute its brother by substituting its value by numbers from 0-3. As Figure 5.7 shows.

```

BrotherRootDetection. determine brothers root                                for
root root

Begin procedure BrotherRootDetection ( )
If ( i == root ) then {
    Let B be the first digit of i
    Let C be the remaining of i
    For (S=1,S<=3,S++)
        broth $\square$ root = concatenate ((B+S) mod 4 ) with C
        Add broth $\square$ root to Brothersroot list
    Next
Return Brothersroot list
}

Call BrotherRootDetection ( )

End Procedure

```

Figure 5.7 Procedure BrotherRootDetection.

5.3.10 Calculating the power

Each node in the network knows its type (0,1,2,3,and 4) and can calculate its power according to its service. For more details, refer to chapter 4, calculating the power section 4.2.3. As Figure 5.8 shows.

```

CalculatePower Calculate a MP  $\bar{p}$  Power  $\bar{p}$  according to its type  $t_{\bar{p}}$ 

Begin procedure CalculatePower ( )
If (tp = 1) then
{  $\bar{p} = (0.15 * CPU_{(ni)} + 0.15 * MEMORY_{(ni)} + 0.30 * BS_{(ni)} + 0.40 * BANDWIDTH_{(ni)})$  }
Else If (tp = 2) then
{  $\bar{p} =$ 
 $(0.25 * CPU_{(ni)} + 0.30 * MEMORY_{(ni)} + 0.15 * BS_{(ni)} + 0.30 * BANDWIDTH_{(ni)})$  }
Else If (tp = 3) then
{  $\bar{p}$ 
=
 $((0.25 * CPU_1((ni)) + 0.25 * MEMORY_1((ni)) + 0.15 * BS_1((ni)) + 0.35 * BANDWIDTH_1((ni)))$  }
Else If (tp = 4) then
{  $\bar{p} =$ 
 $(0.30 * CPU_{(ni)} + 0.30 * MEMORY_{(ni)} + 0.20 * BS_{(ni)} + 0.20 * BANDWIDTH_{(ni)})$  }
Call RoutingAlgorithm ( )
End Procedure

```

Figure 5.8 Procedure CalculatePower.

5.3.11 Routing Algorithm

Each node in the DQT will be one of two (a leaf (child) node, or a parent node).

Each node can communicate with the other

1. The leaf (child) node does the following

- A. Broadcast information messages containing (my index, my ID, my IP, my Power, and my Type) by two hops. Because the network is constructed by such way, be sure that each node arrives to its parent by two hops only. As Figure 5.9 shows.

```

INFOMessage each child does information message info .

Begin procedure INFOMessage ( )

Broadcast my info containing ( $\bar{p}$  ,  $i_{\bar{p}}$  ,  $i_{\bar{p}}$  ,  $t_{\bar{p}}$  and  $\bar{p}$  ) by 2 TTL
Count=0, ++
If (Count ==2) then
{ Discarded message }

```

```

Call ListMessage ( )
End Procedure

```

Figure 5.9 Procedure INFOMessage.

2. When it reaches it's parent

A. Parents multicast a list message to all children.

A Parent check if this child belongs to it or not. If it belongs to it, storing it in a list that consist from information about its children, besides itself, then multicast a list message to all children. As Figure 5.10 shows.

```

ListMessage a parent P receiving information message info .
Begin procedure ListMessage ( )

If (t of received info == ) then

    { store          and my children info at mchild list
      containing (  , i , i , t and   and myp )
      multicast the list to all mchild }

    If (tp of node != 0) then
      { Broadcast }

Call WhereMyparentQuery ( )

```

```
End Procedure
```

Figure 5.10 Procedure ListMessage.

B. Parents do a query message

The parent node in the DQT has a responsibility to know where a parent is because it knows the logical address for it and searches for a physical address for that broadcasting a query message by 3 hops. If any query message knows the answer for this query, it sends it directly to the initiator of a query node to forward it to the nearest neighbors. As Figure 5.11 shows.

```
WhereMyparentQuery a parent  $P$  asking about its upper  $P$  physical
address

Begin procedure WhereMyparentQuery ( )

If (  $Q == P$  ) then
     $Q$  send query message by 3 TTL asking about its  $P$  physical address
    Each node receiving WhereMyparentQuery
    If (he knows the answer) then
        { send it directly to the initiator query  $Q$  }
    Else { forward the message to its neighbors }

    If (Count ==3) then
        { Discarded query }
    }
Call SummeryMessage ( )
End Procedure
```

Figure 5.11 Procedure WhereMyparentQuery.

C. Parents send a summary message to its upper parent containing (index, ID,IP, power, type). As Figure 5.12 shows.

```
SummeryMessage parent  $Q$  send an updated message updm for upper
 $P$ 

Begin procedure SummeryMessage ( )

If (  $Q == P$  ) then
```

```

    □ sends updm for its upper □
      containing (□ , i□ , i□ , t□ and □ )

Call SummeryMessage ( )

End Procedure

```

Figure 5.12 Procedure SummaryMessage.

D. Parents do Time Out Message (Die detection)

A Parent Periodically sends a hi message to all children, The children reply with "I am a live" message If no replies within time T=100, it is resend, if no replies, then the child is dying. As Figure 5.13 shows.

```

TimeOutMessage parent □ send hi message for mchild

Begin procedure TimeOutMessage ( )

If (□ == □ ) then
    Periodically □ sends h□ to mch□ld list
    mch□ld replies with li□e message
    If (no li□e message withint T= 100 ms time) then
    { □ resend h□ message
    If (no li□e message) then {
    mch□ld is dyeing }
    }

Call NumbersOfNodes ( )

End Procedure

```

Figure 5.13 Procedure TimeOutMessage.

E. Parents do Numbers Of Nodes Messages

From time to time, each parent sends Numbers of Nodes messages to their upper parent until reaching the root. The root receiving this message exchanges it with its root brothers. The least id root node calculate the numbers of node by addition; If the number of node $> 2^n * 2^n$ then, rebuild the DQT. As Figure 5.14 shows.

```

NumbersOfNodes  the least ID root check to reconstruct DQT

Begin procedure NumbersOfNodes ( )
    Periodically  $\square$  send #ofnodes message to upper  $\square$  until reaching the
     $r \square t$ 
     $r \square t$  receiving this message exchange it with its  $r \square t$  brothers
    The least id  $r \square t$  calculate the numbers of node by addition
        If (number of node  $> 2^n * 2^n$ ) then
            { Rebuild the DQT.}
        Else
            { Do nothing.}

Call ParentsMessage ( )

End Procedure

```

Figure 5.14 Procedure NumbersOfNodes.

3. Root do Parents Message

Any index equal root then send Parents Message containing (Index, ID, IP, Ttpe, Power)

To all its children. As Figure 5.15 shows.

```

ParentsMessage  root  $rot$  send ParentsMessage to all my child list
 $mchild$ 

    Begin procedure ParentsMessage ( )
    If ( $\square == r \square t$ ) then
        { send ParentsMessage containing ( $\square$ ,  $I \square$ ,  $I \square$ ,  $t \square p$ ,  $\square$ ) To
 $mchild$  list}
        Else If (index = parent) then {
            { Store it at list }
        }

    End Procedure

```

Figure 5.15 Procedure ParentsMessage.

5.3.12 Joining the Overlay

If any new node wants to join the overlay, it sends its information by a message to 3 hops to be insure to arrive to its parent. Then, parent check the index if it exited or not by using IP address and replay with change index by adding literal from A-Z . As Figure 5.16 shows.

```

JoiningOverlay any new node  $\bar{N}$  wants to join the overlay

    Begin procedure JoiningOverlay ( )
If ( $\bar{N}$  want to join) then
    {  $\bar{N}$  call INFOMessage ( ) + I hop
       $\bar{N}$  receiving this INFOMessage checks
        if ( $\bar{N}$  exists by IP address)
       $\bar{N}$  send change  $\bar{N}$  by adding a letter from A – Z to the end of  $\bar{N}$ 
    } Else {
       $\bar{N}$  replies with a join acknowledge message to  $\bar{N}$  with a list message
       $\bar{N}$  calls SummeryMessage ( )
       $\bar{P}$  stores it at list }

End Procedure
  
```

Figure 5.16 Procedure JoiningOverlay.

5.3.13 Leaving the Overlay

A. A Selective Leave

A Parent Leave

A Parent sends a leave message to its parent when it reaches parent act by

1. A Parent checks its children's list for any index ending with a literal (backup level)
2. If found then
3. substitute the leaved node and delete the letter from the index
4. else

5. The least ID node takes the place of the leaved node.
6. Send an update summary message to its parent.
7. Send a message to all children to delete the leaved node from the children.
8. Delete the leaved node from its list.
9. Connect all of children with its ancestor. As Figure 5.17 shows.

A SelectiveParentLeave parent \square wants the selective leave of the overlay

```

Begin procedure SelectiveParentLeave ( )
If ( $\square$  wants the selective leave overlay) then
    {  $\square$  sends a leave message to its upper  $\square$ 
       $\square$  receiving this leave message check
        if (mchChild list has  $\square$  with literal (backup level) then
substitute the leaved  $\square$  and delete letter from  $\square$ 
}else {
The least  $\square$  node takes the place of the leaved node.
Call SummeryMessage ( )
    Send a message to all mchChild list to delete the leaved node
     $\square$  delete the leaved node from mchChild list
    Connect all of child with their ancestor }
End Procedure

```

Figure 5.17 Procedure SelectiveParentLeave.

A Child Leave

CHILD sends a leave message to a parent when it reaches parent act by

1. A Parent checks the backup level
2. If found then
3. substitute the leaved node
4. else
5. Send an updated summary message to its parent.
6. Send a message to all children to delete the leaved node from children.
7. Delete the leaved node from its list. As Figure 5.18 shows.

SelectiveChildLeave child ch_{id} wants the selective leave of the overlay

```

Begin procedure SelectiveChildLeave ( )
If ( $ch_{id}$  wants the selective leave overlay) then
    {  $ch_{id}$  sends a leave message to its  $parent$ 
       $parent$  receiving this leave message check
        if ( $mch_{id}$  list has  $parent$  with literal (backup level) then
substitute the leaved  $ch_{id}$  and delete letter from  $parent$ 
        }else {
Call SummeryMessage ( )
    Send message to all  $mch_{id}$  list to delete the leaved node
     $parent$  deletes the leaved node from  $mch_{id}$  list
    }
End Procedure

```

Figure 5.18 Procedure SelectiveChildLeave.

B. A Force Leave

A Parent Leave

1. When the upper parent detects that the child is dying, it does the following
2. The Parent checks its children's list for any index ending with literal (a backup level)
3. If found then ,
4. substitutes the leaved node and deletes the letter from the index
5. else
6. The least ID node takes the place of the leaved node.
7. Send an updated summary message to its parent.
8. Send a message to all children to delete the leaved node from children.
9. Delete the leaved node from its list.
10. Connect all of children with its ancestor. As Figure 5.19 shows.

ForceParentLeave parent $parent$ forced to leave the overlay

```

Begin procedure ForceParentLeave ( )
If (upper  $parent$  detects its  $ch_{id}$  is die) then

```

```

    { if (mch[id] list has [id] with literal (backup level)
substitute the leaved [id] and delete letter from [id]
} else {
The least [id] node takes place of the leaved node.
Call SummeryMessage ()
    Send message to all mch[id] list to delete leaved node
    [id] delete the leaved node from mch[id] list
    Connect all of child with its ancestor }
End Procedure

```

Figure 5.19 Procedure ForceParentLeave.

Child Leave

When parent detects the child is die

1. Send update summary message to its parent.
2. The message is sent to all children to delete the left MPs from children.
3. Delete the leaved MPs from its list. As Figure 5.20 shows.

```

ForceChildLeave child ch[id] forced to leave the overlay

    Begin procedure ForceChildLeave ()
If ([id] detects its ch[id] is die) then
    { Call SummeryMessage ()
    Send message to all mch[id] list to delete leaved node
    [id] delete the leaved node from mch[id] list }
End Procedure

```

Figure 5.20 Procedure ForceChildLeave.

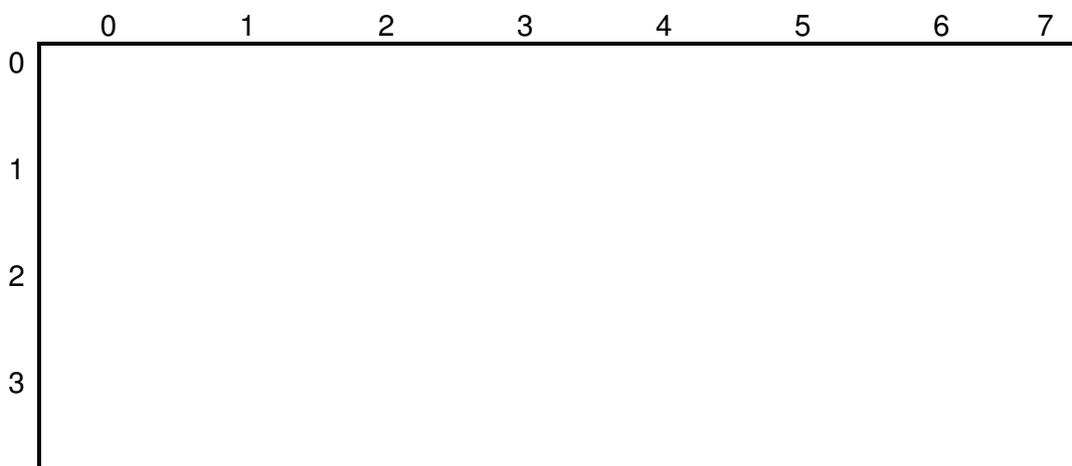




Figure 5.21 shows network as a two dimensional array.

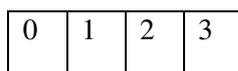
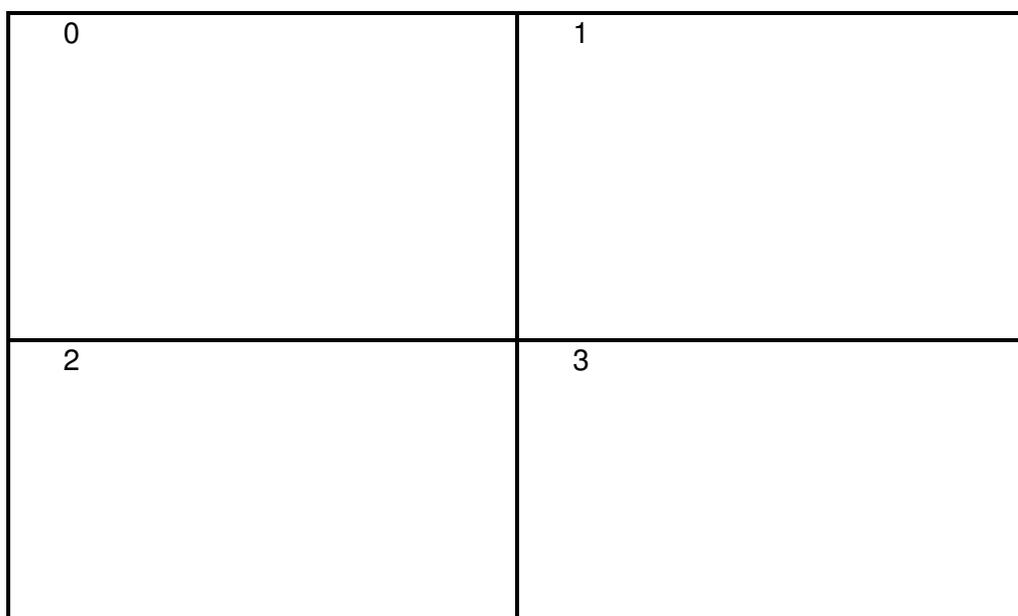


Figure 5.22 shows partitioning and indexing procedures in the first level and its root.

00	01	10	11
02	03	12	13
20	21	30	31

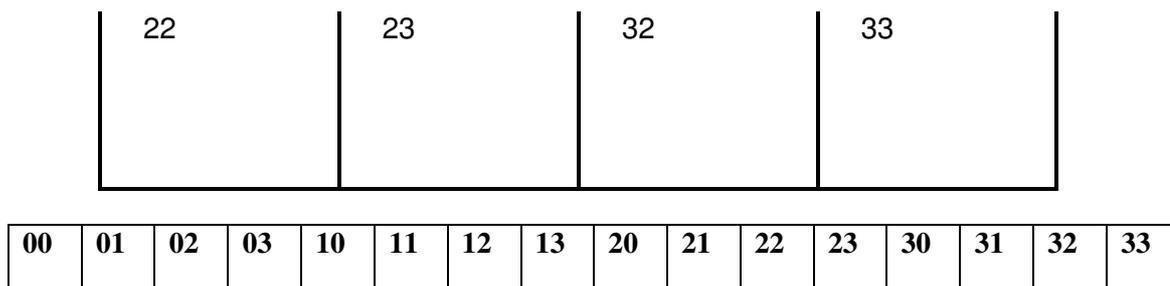


Figure 5.23 shows partitioning and indexing procedures in the second level.

000	001	010	011	100	101	110	111
002	003	012	013	102	103	112	113
020	021	030	031	120	121	130	131
022	023	032	033	122	123	132	133
200	201	210	211	300	301	310	311
202	203	212	213	302	303	312	313
220	221	230	231	320	321	330	331
222	223	232	233	322	323	332	333

Figure 5.24 shows partitioning and indexing procedures in the third level.

000	001	010	011				
002	003	012	013				
020	021	030	031				
022	023	032	033				



Figure 5.25 shows upper the left (NW) section.

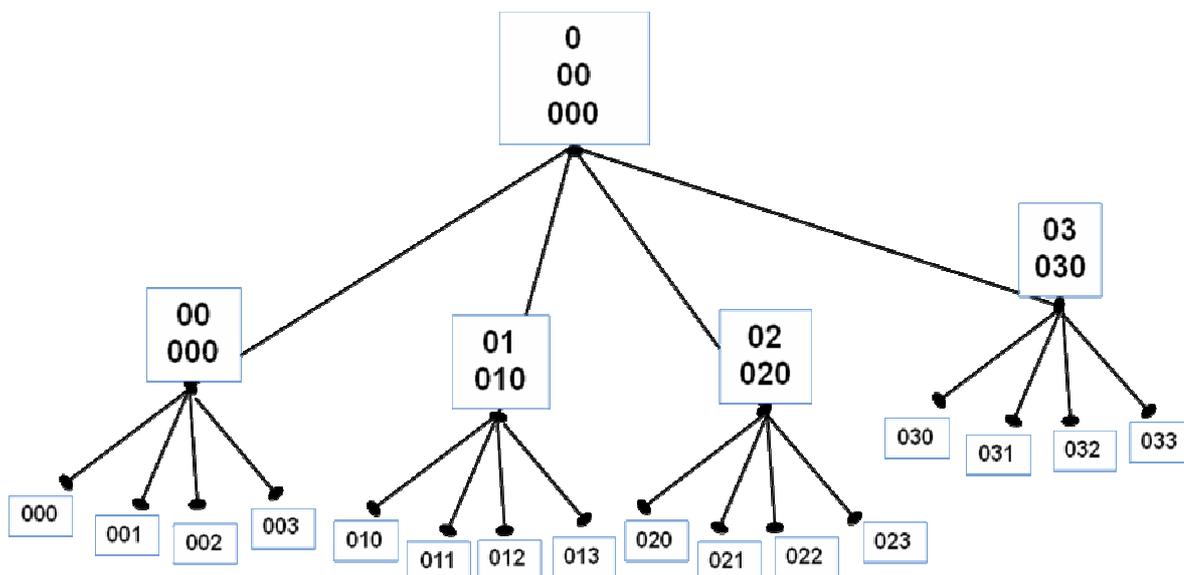


Figure 5.26 shows the upper left (NW) section, and the quadtree representation

				100	101	110	111
				102	103	112	113
				120	121	130	131
				122	123	132	133

Figure 5.27 shows the upper right (NE) section.

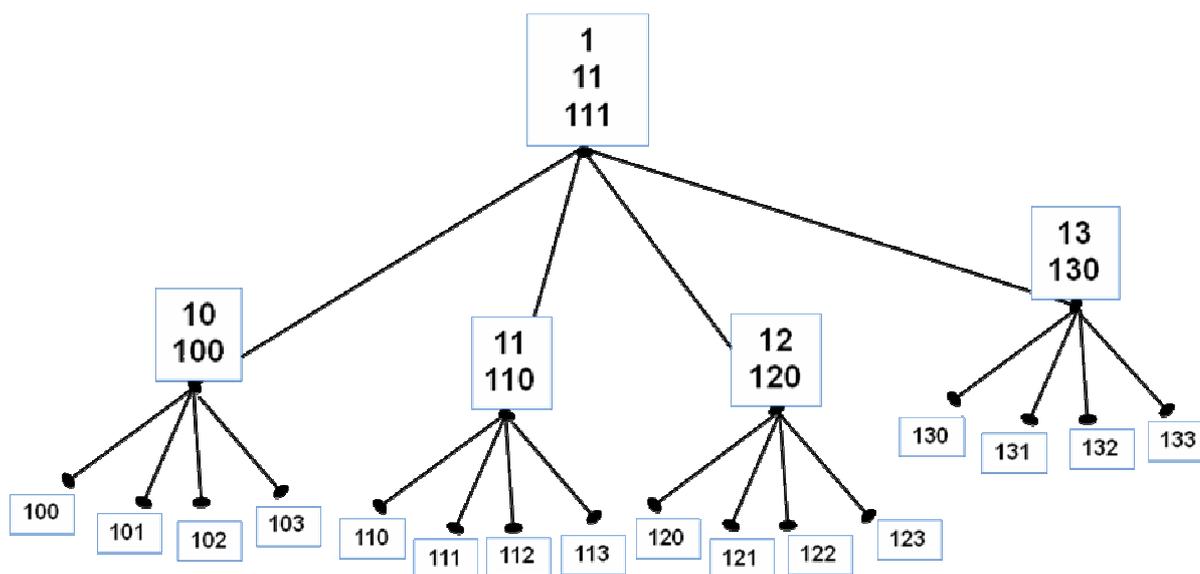


Figure 5.28 shows the upper right (NE) section, and the quadtree representation.

200	201	210	211				
202	203	212	213				

220	221	230	231				
222	223	232	233				

Figure 5. 29 shows the lower left (SW) section.

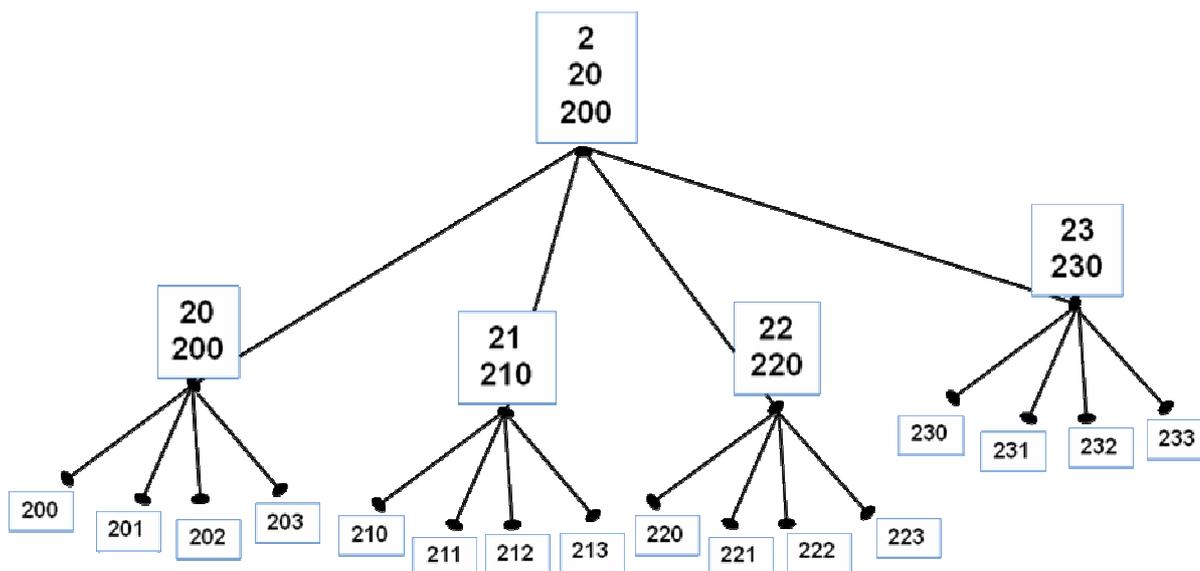


Figure 5.30 shows the lower left (SW) section, and the quadtree representation.

				300	301	310	311

				302	303	312	313
				320	321	330	331
				322	323	332	333

Figure 5.31 shows the lower left (SE) section.

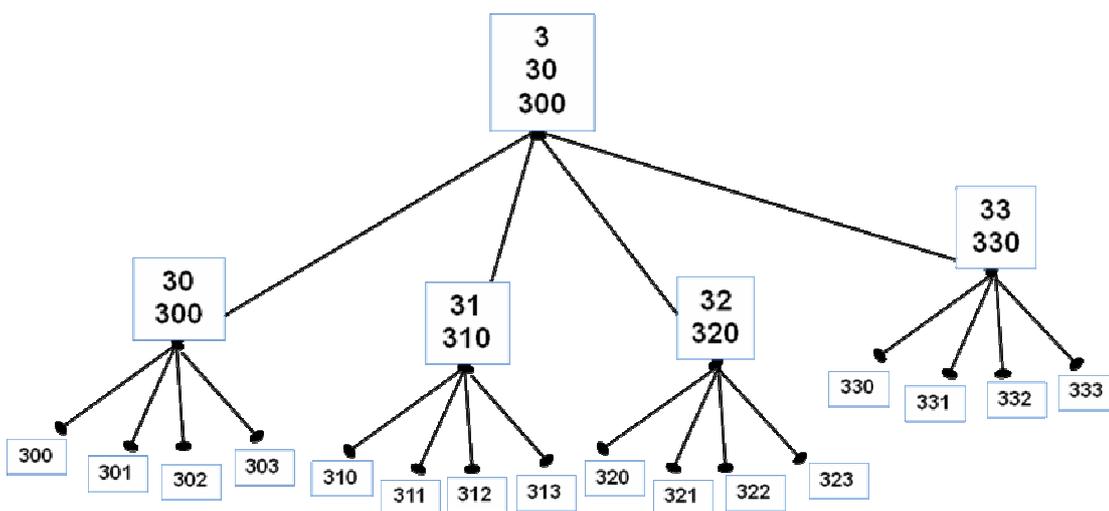


Figure 5.32 shows the lower right (SE) section, and the quadtree representation.

5.4 A Self-Load Balancing

In this section, we will try to cover and demonstrate each step to achieve the self-load balancing between all MPs available in the networks

5.4.1 The Procedure Power Percentage

Each MP knows its type and power then each of them must calculate its Power to be a percentage (between 0-100 percent). For that, each MPs according to its type

sends a message to its parent to determine the highest (power full) MPs. Finally, each MP can calculate its power percentage. As Figure 5.33 shows.

```

PowerPercentage Calculate MP  $\bar{P}$  Power  $\bar{P}$  percent according to its type
 $\bar{t}$ 

Begin procedure PowerPercentage ( )
If ( $\bar{t} = 1$ ) then
    {  $\bar{P} = \left( \frac{ITS_{MOST_{PCACCHINGMP}} P}{MOST_{PCACCHINGMP}} \right) * 100$  . }
Else If ( $\bar{t} = 2$ ) then
    {  $\bar{P} = \left( \frac{ITS_{MOST_{PgnchrizationMP}} P}{MOST_{PgnchrizationMP}} \right) * 100$  . }
Else If ( $\bar{t} = 3$ ) then
    {  $\bar{P} = \left( \frac{ITS_{MOST_{ProctingMP}} P}{MOST_{ProctingMP}} \right) * 100$  . }
Else If ( $\bar{t} = 4$ ) then
    {  $\bar{P} = \left( \frac{ITS_{MOST_{PadapizationMP}} P}{MOST_{PadapizationMP}} \right) * 100$  . }

Call CALInComEdges ( )
End Procedure

```

Figure 5.33 Procedure PowerPercentage.

5.4.2 The Procedure number of incoming edges

Each MP can calculate its incoming edges by dividing its power over 12.5 to get the incoming edges to these MPs. Why do they choose this value 12.5? To always get at maximum 8 incoming edges and minimum 4 incoming edges. Why are the 8 and 4 maximum and minimum respectively? Because they always need to build this overly network by the cheapest price and 4 edges to maintain the property which produces strongly connected components. As Figure 5.34 shows.

```

CALInComEdges Calculate in coming edges
Begin procedure CALInComEdges ( )
    NumbIn(edges) = round(power / 12.5)
    If (NumbIn (edges) < 4) then
        { NumbIn (edges) = 4. }

```

```

Call JoinInComEdges ( )
End Procedure

```

Figure 5.34 Procedure InComEdges

5.4.3 Procedure Joining incoming edges

Joining, the incoming edges to MPs, here setting a counter, which initial state equals zero and the maximum value equals the number of incoming edges. To receive the incoming edges, first each MP searches its own geographical area by looking into its own list for the lowest MPs power and should be the same type. If it still wants more incoming edges, MPs send a message to its parent asking for edges and its parent in turn forwards the request to the upper parents until it accumulates the required number of edges. After that, the accumulated list will be returned to the MPs that initiate the request. The list contains the following information (index, IP address, type) and finally connects with the accumulated list. As Figure 5.35 shows.

```

JoinInComEdges MP a receiving a incomedge from other lowest MPs
d
Begin procedure JoinInComEdges ( )
If ( z found MP z && same tz ) then {
    z selects the MP which belongs to its area
} Else {
    z send message to its z asking for incomedge
    its z sends a message to its upper z asking for incomedge
    If ( incomedge list is accumulated) then {
        forward the list to MP z that initiate the request.
        The accumulated list contains ( z , IPz , tz and z )
    }
    z connect with MP z
}
End Procedure

```

Figure 5.35 Procedure JoinInComEdges.

5.4.4 Procedure Assigning Job

When a new job enters the network, the job received by a normal node which is type = 0 then it forwards the request to the nearest MPs it knows, Otherwise, sending it to its parent. If the job received by a MPs node which is type = 1- 4. Then it can provide the service then MPs find out the highest incoming edges MPs and assign the job to it to process, and always chooses the MPs, which belongs to the same host area (the nearest MPs). If it cannot provide the service then MPs forwards the request to the nearest MPs it knows to serve this job; otherwise, sending it to its parent. After that, it calls the Delete Edge Procedure to decrease its connectivity. After finishing the job and exiting from the network, it calls Add Edge Procedure to increase its connectivity. As Figure 5.36 shows.

```

AssignJob a MP a receiving a job request jr

Begin procedure AssignJob ( )
If (a can provide the service) then {
    a find the highest incoming edge from MP b
    a assign the job to b .
} Else {
    a chooses the MP which belongs to the same host area.
    If (type = 0) then {

        forward the request to the nearest MP      in its knowledge.
    } Else { send the request to a parent. }
}

Call DeletEdge ( )
If (jr is finished) then { Call AddEdge ( ) }

End Procedure

```

Figure 5.36 Procedure AssignJob.

5.4.5 Procedure Delete Edge

Each MP can decrease its connectivity when the job is assigned to it. The removing edge will be from the farthest geographical area, and from the MP that has the highest power. After that, updating its power by subtracting about 12 from its original power. As Figure 5.37 shows.

```

DeletEdge the MP  $\alpha$  receiving a job request  $jr$ 

Begin procedure DeletEdge ( )
If ( $\alpha$  receiving  $jr$ ) then {
 $\alpha$  deletes one of its incoming edge from the farthest geog.area to the high MPs
 $\alpha$ 
 $\alpha$  = round ( $\alpha$  - 12.5)
 $\alpha$  sends an updated message to its parent until reaching root  $rot$ 
}
End Procedure

```

Figure 5.37 Procedure DeletEdge.

5.4.6 Procedure Add New Edge

Each MP can increase its connectivity when the job exits the network and sends a message to its parent asking him for new incoming edges from the lowest power value MP. Then, updates its power by adding about 12 to its original power. As Figure 5.38 shows.

```

AddEdge the MP  $\alpha$  finishing a job request  $jr$ 

Begin procedure AddEdge ( )
If ( $\alpha$  finishing  $jr$ ) then {
 $\alpha$  sends a message to its  $\alpha$  asking for inco-edge
its  $\alpha$  sends a message to its upper  $\alpha$  asking for inco-edge
If (inco-edge list is accumulate) then {
forward the list to MP  $\alpha$  that initiates the request.
Accumulate the list that contains ( $\alpha$  ,  $l\alpha$  ,  $t\alpha$  and  $\alpha$  )
}
}

```

```
}  
  E connects with it  
  E = round (E + 12.5)  
  E sends an updated message to its parent until reaching root rot  
}  
End Procedure
```

Figure 5.38 Procedure AddEdge.

Chapter Six

6

Experimental

Evaluation

6.1 Network simulation

Openxtra website (2007) argues, Network simulators try to model actual world networks. The inspiration being that if a proposed system can be modelled, then characteristics of the model can be altered and the output analyzed. Since the process of model modification is relatively cheap then a wide variety of scenarios can be analyzed at low cost comparative to making changes to a real network. Network simulators are not perfect. They will not perfectly model the network. They will, though, be close enough so as to give a meaningful insight into how the network is working, and how the modifications will affect its function. Network simulators are most useful when used to model large networks such as the environment that we are considering in this thesis.

6.2 The Simulation Tool (J-Sim Simulator)

This section introduces briefly the well-known JSim simulation tool. JSim (previously known as JavaSim) is a Java-based simulation system for building quantitative numeric models and analyzing them with respect to experimental reference data. J-Sim's primary focus is in physiology and biomedicine; however, its computational engine is quite general and applicable to a wide range of scientific domains (j-sim website, 2008).

Hou & Tyan (2005) argue, J-Sim is a component-based, compositional simulation environment. It has been built upon the notion of the autonomous component-programming model. The basic entity in J-Sim is components, but unlike the other component-based software packages/standards, components in J-Sim are autonomous and are realization of software ICs. Besides J-Sim is an object-oriented library for discrete-time process-oriented simulation. Its main application area is

queuing network simulation. However, the range of its use can be very wide – almost any system where object states change discretely can be modelled using J-Sim. The autonomous component architecture mimics the IC design architecture in the closest possible way (physiome website, 2011).

The behaviour of J-Sim components are defined in terms of contracts (in much the same way IC chips are defined in the specification in the cookbook) and can be individually designed, implemented, tested, and incrementally deployed in a software system. A system can be composed of individual components in much the same way a hardware module is composed of IC chips. Moreover, components can be plugged into a software system, even during execution. J-Sim has been developed entirely in Java™. This, coupled with the autonomous component architecture, makes J-Sim a truly platform-neutral, extensible, and reusable environment. J-Sim also provides a script interface to allow integration with different script languages such as Perl, Tcl, or Python. In the current release, it is fully integrated with a Java implementation of the Tcl interpreter (with the Tcl/Java extension), called Jacl. So, similar to ns-2, J-Sim is a dual-language simulation environment in which classes are written in Java (for ns-2, in C++) and "glued" together using Tcl/Java. However, unlike ns-2, classes/methods/fields in Java need not be explicitly exported in order to be accessed in the Tcl environment. Instead, all the public classes/methods/fields in Java can be accessed (naturally) in the Tcl environment (arcor website, 2009).

6.3 J-Sim for Network Simulation

Jsim official website (2010) argue J-Sim is executed on top of a component-based software architecture, called the Autonomous Component Architecture (ACA), that strictly mimics the integrated circuit (IC) design. The fundamental entities in the ACA are components, which talk with one another via sending/receiving data at their ports. When data arrives at a port of a component, the component processes the data without delay in a self-governing execution context. The software architecture of the ACA is aggravated by the conviction that software design cannot accomplish the same level of modularity as IC design due to the fact that the Object Oriented (OO) programming paradigm is fundamentally dissimilar from hardware design in component binding. Specially, in OO programming, a class makes straight references to other class instances and makes function calls to those exposed by other class instances (Tyan 2002).

6.4 Justification of the Method of Study

In this research, extensive simulation experiments have been conducted to explore performance-related issues of Self-Load Balancing in Autonomic Overlay Networks. This section discusses briefly the choice of simulation as a tool of study for the purpose of this research, justifies the adoption of J-Sim as the preferred simulation tool, and further provides information on the techniques used to reduce the opportunity of simulation errors.

After some consideration, simulation has been selected as the method of study in this research. In general, in addition to conducting measurements on a real practical system or test bed, there exist two techniques for system performance evaluation:

analytical modelling and simulation. One of the key considerations when adopting a given evaluation technique is the level of the desired accuracy. In general, analytical models have often-low requirements in terms of computation costs, but they often rely on many assumptions and simplifications that restrict their applicability to a limited number of scenarios. In contrast, simulation models can easily incorporate details to the desired level of accuracy in order to mimic more closely the behaviour of the real system. The consequence of this is that simulations often require a longer time to develop and run the code, compared to analytical modelling. However, as we have used the J-Sim simulator that has already been developed and extensively validated, we have easily incorporated our suggested algorithms into the simulator. This has helped to considerably cut down the development time and debugging of the code. Most often cost, along with the ease of being able to change configurations, is the prime motivation for developing simulations for expensive systems. The Self-Load Balancing algorithms designed and analysed in this study are for Autonomic Overlay Networks, which could consist of a large number of processors. Such a study could not be easily carried out on a practical system, as the experimental setup would require substantial and expensive resources. J-Sim has been widely used to evaluate the performance of network simulation. It is worth mentioning that we have evaluated the performance of our Self-Load Balancing algorithms based on a real workload trace and compared the results against those obtained from our simulation study based on 95% Confidence interval (95% CI) workloads. The results of the comparison have revealed that the conclusions reached on the performance merits of the Self-Load Balancing.

6.5 Experiment one: Distributed Quadtree (DQT)

This research employs the Limited-flooding to build the DQT. In a limited-flooding protocol, a service request is broadcasted to all direct neighbors of the requesting node. Close neighbors send it on to their neighbors; the propagation is controlled by a TTL value that indicates how far the query should be sent from the requesting node. The measurements will include the overhead of building different levels of distributed Quadtree, stretch, time, and success rate were tested against the query overhead in a large-scale network.

6.5.1 The DQT Building Message for Each Level

The DQT Message cost represents the total number of generated messages from the moment of initiating a broadcast until reaching the required DQT parent. Figure 6.1 shows that our DQT algorithm produces fewer messages as go up in the tree for example to construct level one need the biggest number of messages and messages cost decreases by quarter to build level two and so on until reaching the root . The curve in figure 6.1 shows that when moving up from level to level the messages cost decreases by a quarter, besides, all of this operation happened only one time. Moreover can conclude from the figure the confidence interval at 95% CI for all levels are distributed normally for each level mapped with numbers of messages.

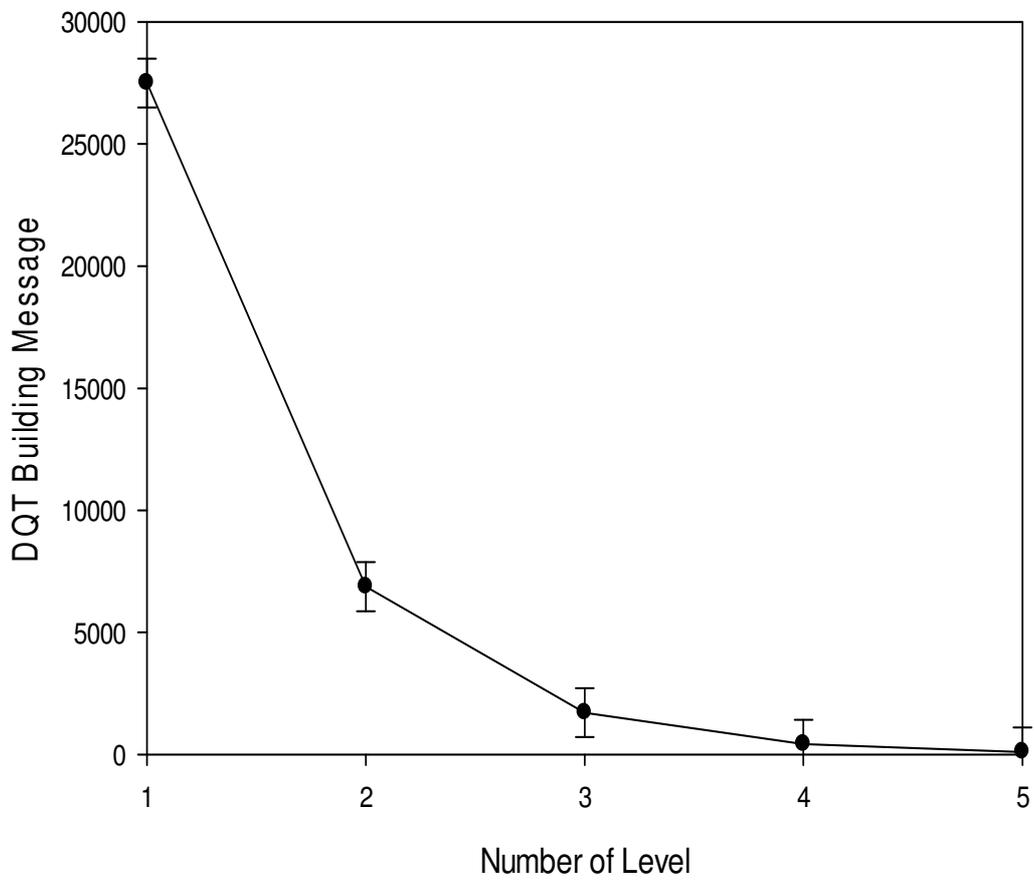


Figure 6.1 Distributed Quadtree building message

6.5.2 The stretch

A stretch is defined as the average number of hops taken by an overlay packet divided by the number of hops the packet takes when using an IP-layer path between the same source and destination. Figure 6.2 a & b. shows that the proposed DQT algorithm compared with the direct hop. The curve shows the maximum hops equal 20. The reason behind that the direct hop does better than the proposed algorithm. Direct hop is assumed each node knows the IP for the destination node. While the proposed algorithm used the DQT parent to go from level to level. In addition can conclude from the figure the confidence interval at 95% CI for all number of query are distributed normally for each number of query mapped with numbers of hops.

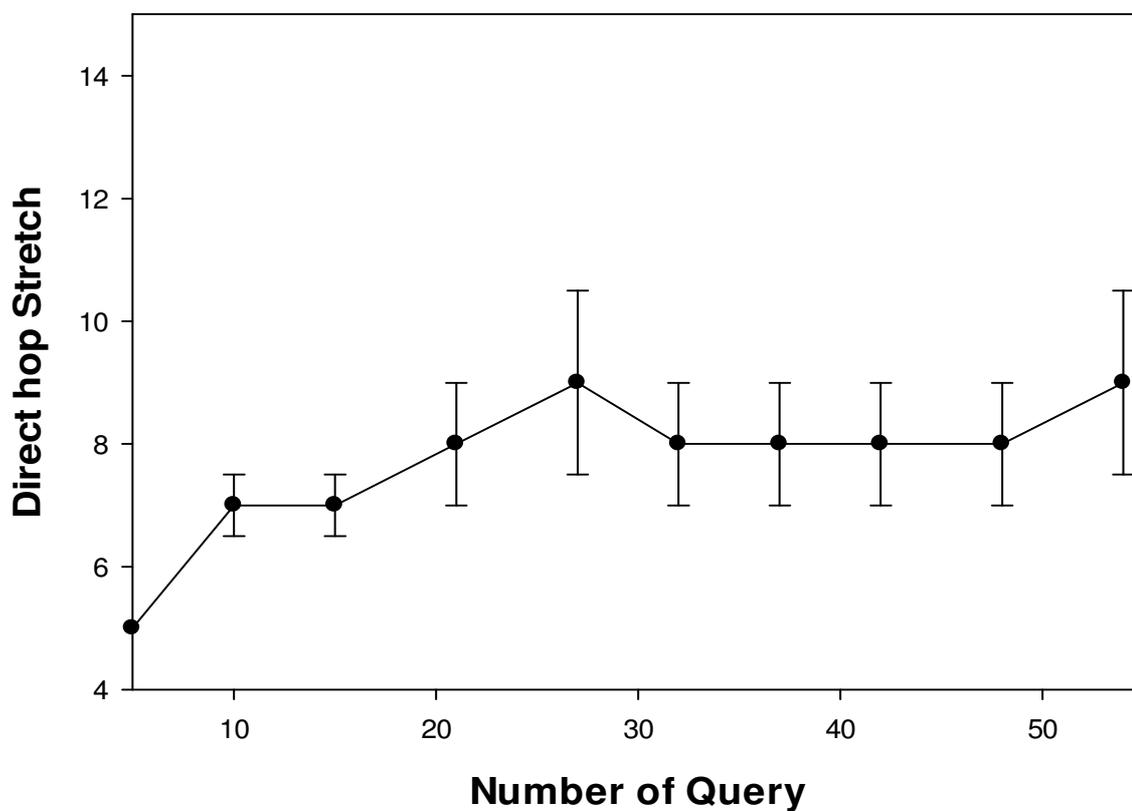


Figure 6.2a stretch

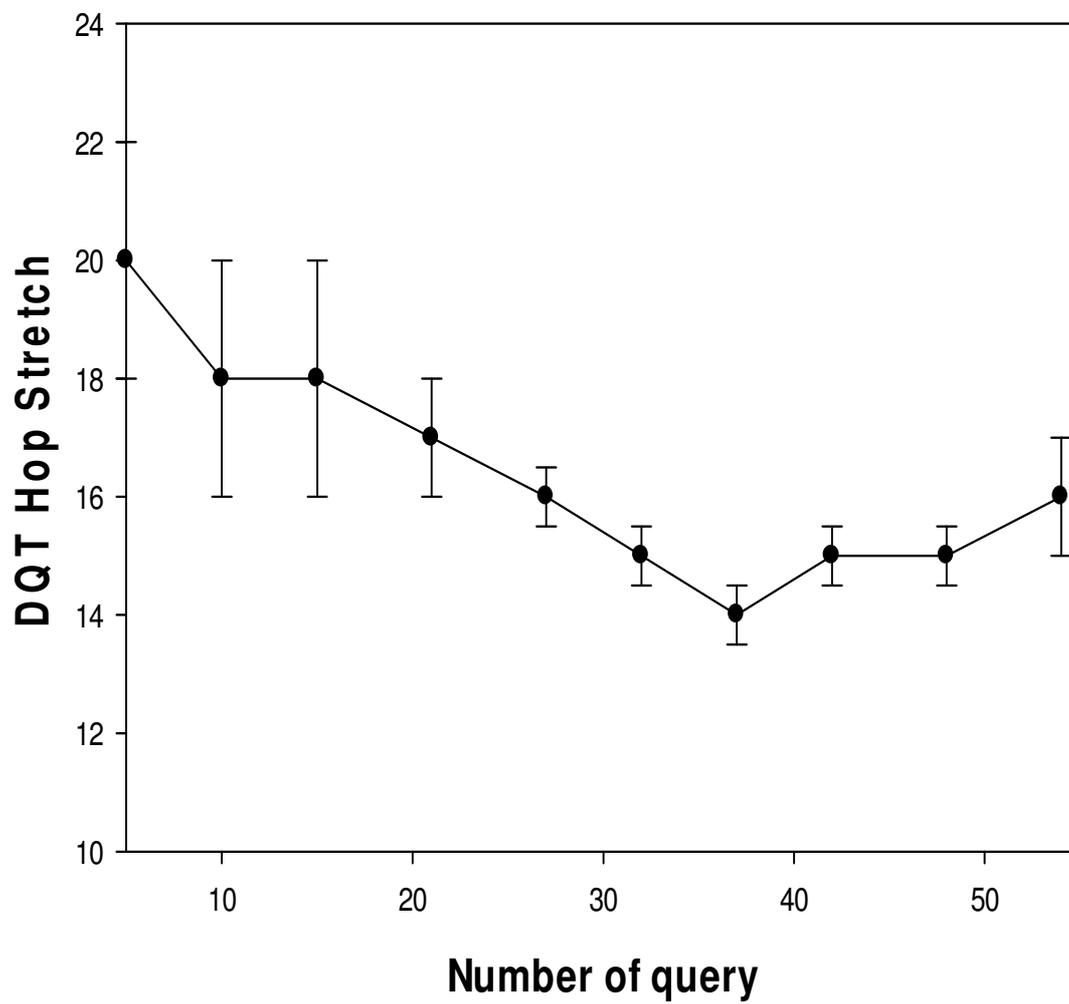


Figure 6.2b stretch

6.5.3 Response Time

The response time is the difference between the starting time of the query and the arrival time of the reply. Figure 6.3 shows the response time for queries. The proposed algorithm curves almost the same and does not exceed 120 ms. Here we can conclude from the figure the confidence interval at 95% CI for all numbers of query are distributed normally for each response time..

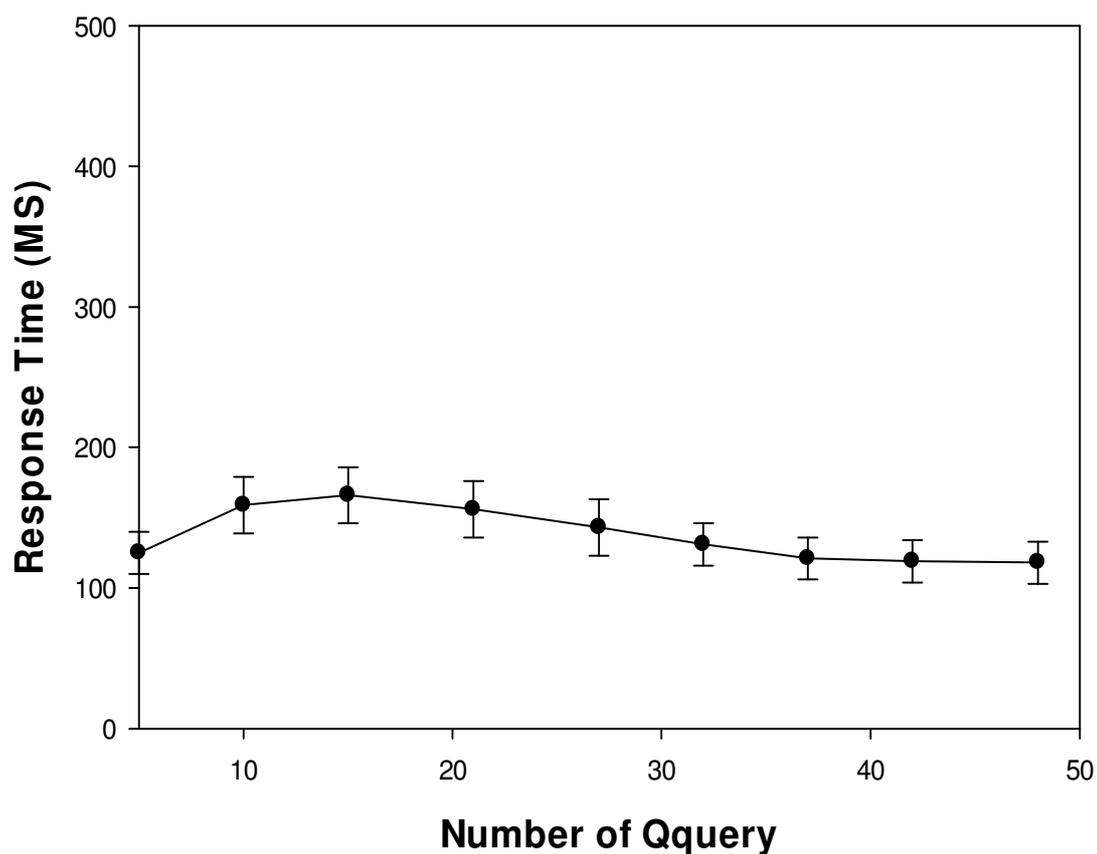


Figure 6.3 Response Time

6.5.4 Success Rate

The success rate is defined as the number of requests that receives positive responses, divided by the total number of queries. In a network, the stability success rate algorithm indicates a better performance. Figure 6.4 shows the success rate the proposed approach results in a constant success rate. We can say from the figure the confidence interval at 95% CI for all numbers of query are distributed normally mapped with success rate.

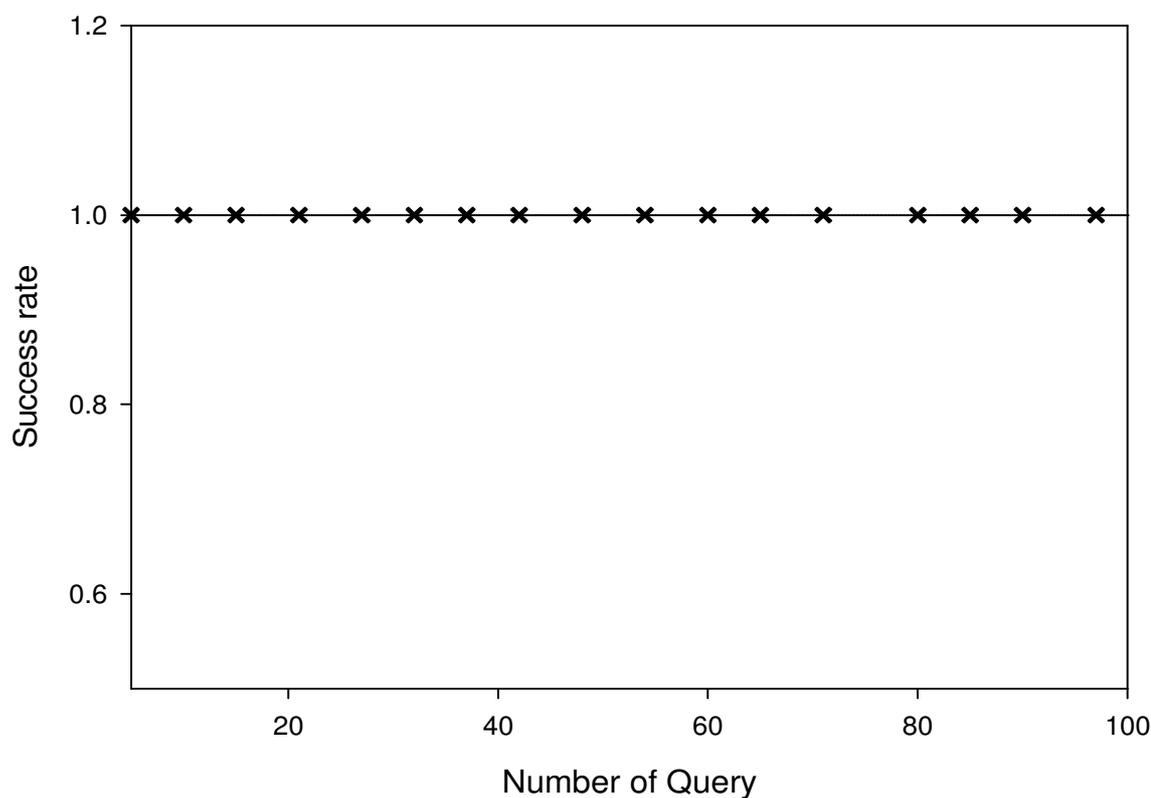


Figure 6.4 Success rate

6.6 Experiment Two: Joined Node

As described in chapter one, MPs have their own resource limitations. They can join the network as they are being owned by the network provider. Besides, users may join this dynamic network at any time.

6.6.1 The Network Load

The network load is the total number of messages used to join any new node to the DQT. Figure 6.5 shows the average network load. The proposed algorithm curve is almost the same and it does not exceed the 25 messages. In addition, from the figure the confidence interval at 95% CI for all joining nodes is distributed normally mapping with numbers of messages.

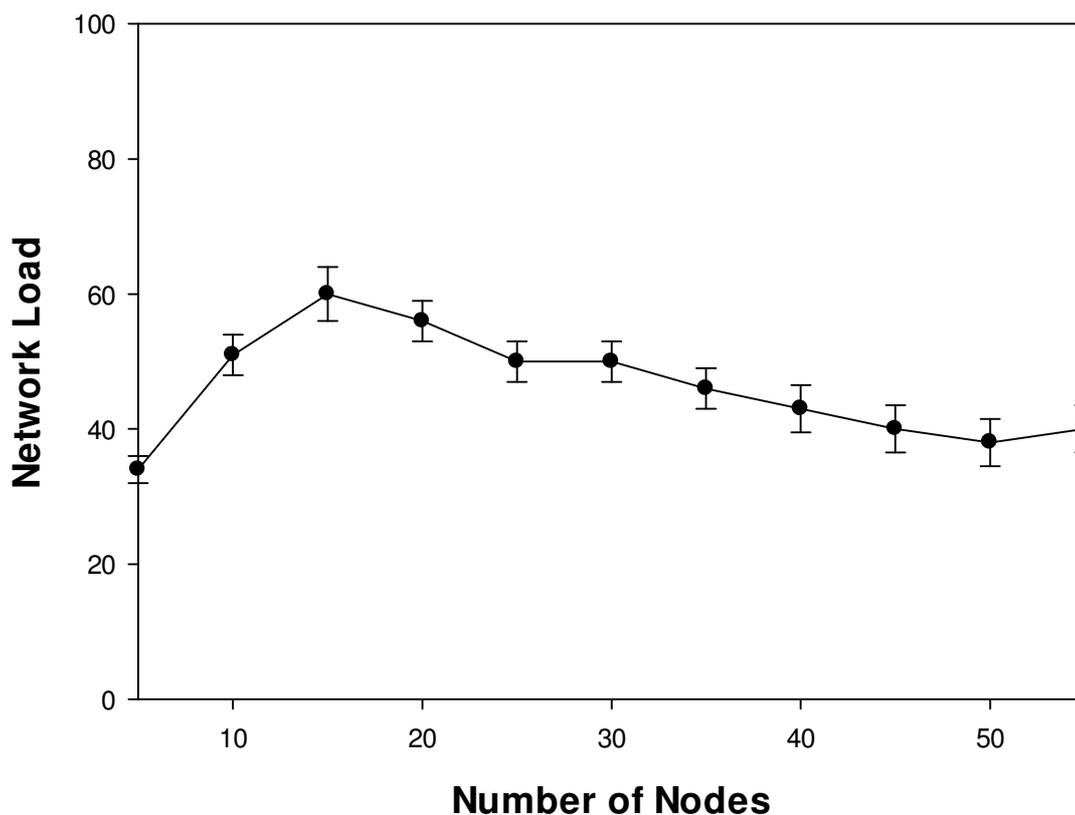


Figure 6.5 Network load

6.6.2 Response Time

The response time is the difference between the starting time of the query and the arrival time of the reply. Figure 6.6 shows the response time. The proposed algorithm curve shows the maximum response time equal 50 ms. In addition, from the figure the confidence interval at 95% CI for all joining nodes are distributed normally mapping with response time.

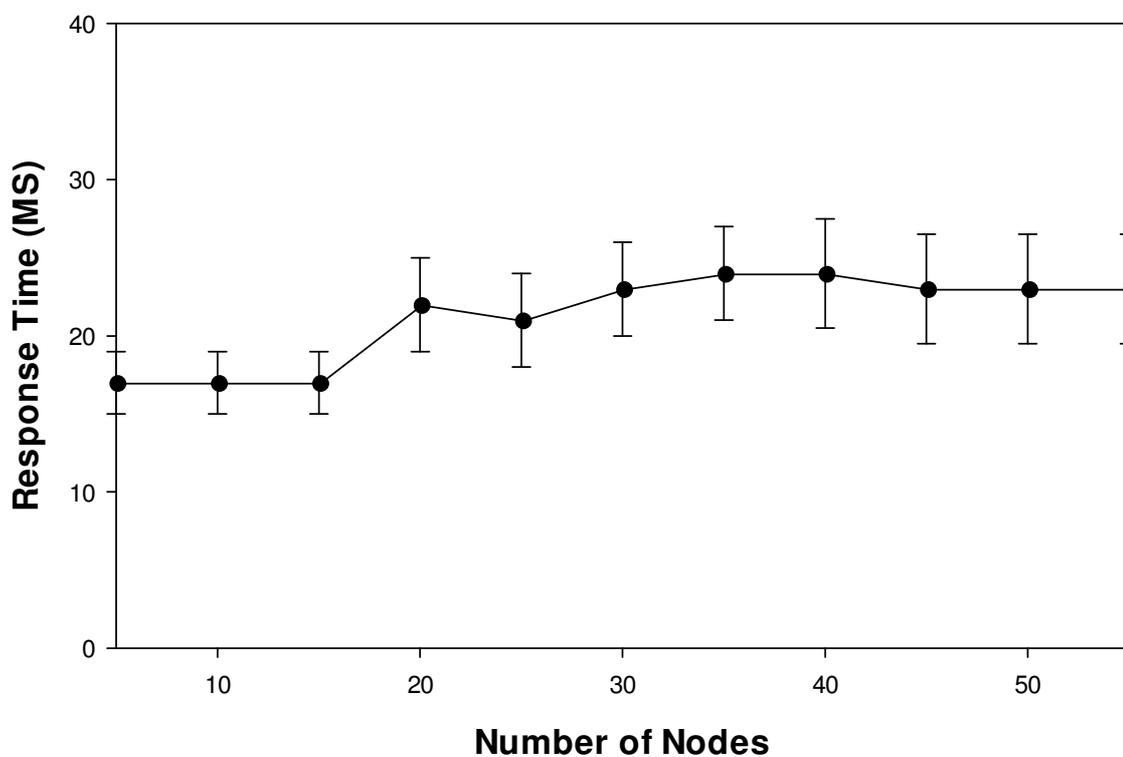


Figure 6.6 Response Time

6.7 Experiment Three: Left Node

Again, MPs have their own resource limitations they can leave the network, as the network provider owns them. Besides, users may leave this dynamic network at any time.

6.7.1 The Network Load:

Network load is the total number of messages used to leave any node from the DQT. Figure 6.7 shows the average network load. The proposed algorithm curve is constant because the number of messages generated by the leave nodes is constant. From the figure, the confidence interval is at 95% CI for all leaving nodes are distributed normally mapping with numbers of generated messages.

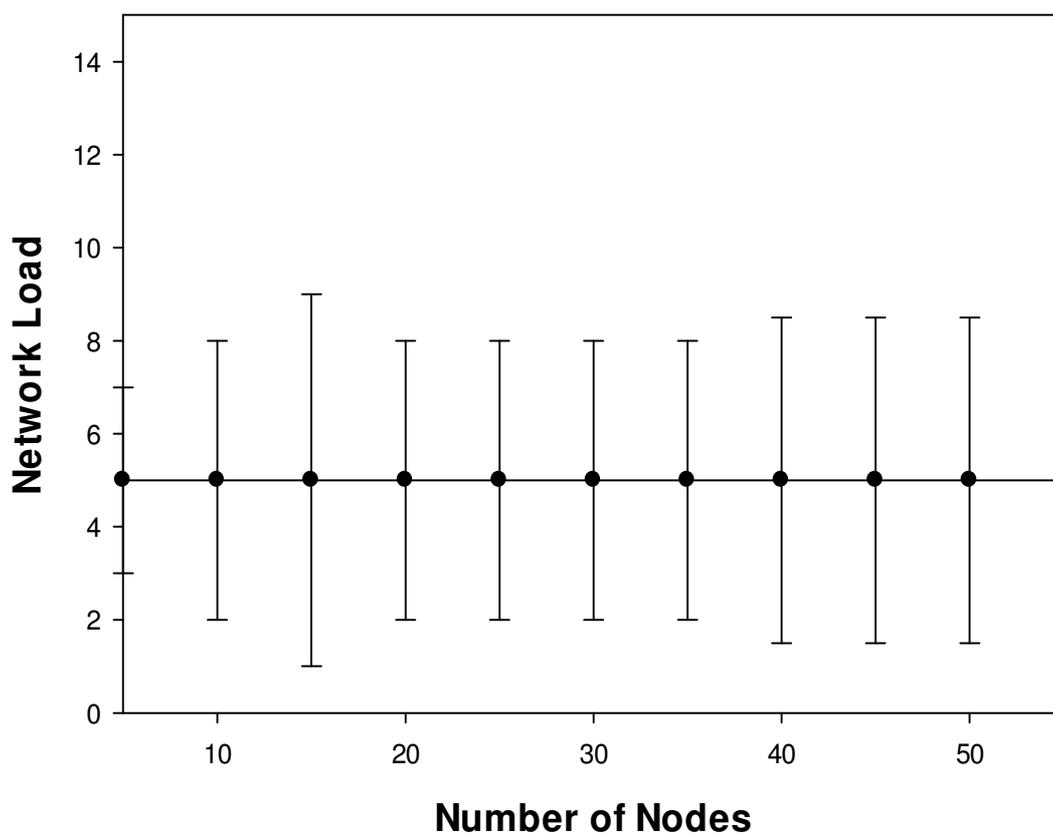


Figure 6.7 Network load

6.7.2 Response Time

The response time is the difference between the starting time of the query and the arrival time of the reply. Figure 6.8 shows the average response time. The proposed algorithm curves almost the same and does not exceed 50 ms. the figure show the confidence interval at 95% CI for all leaving nodes which are distributed normally mapping with response time.

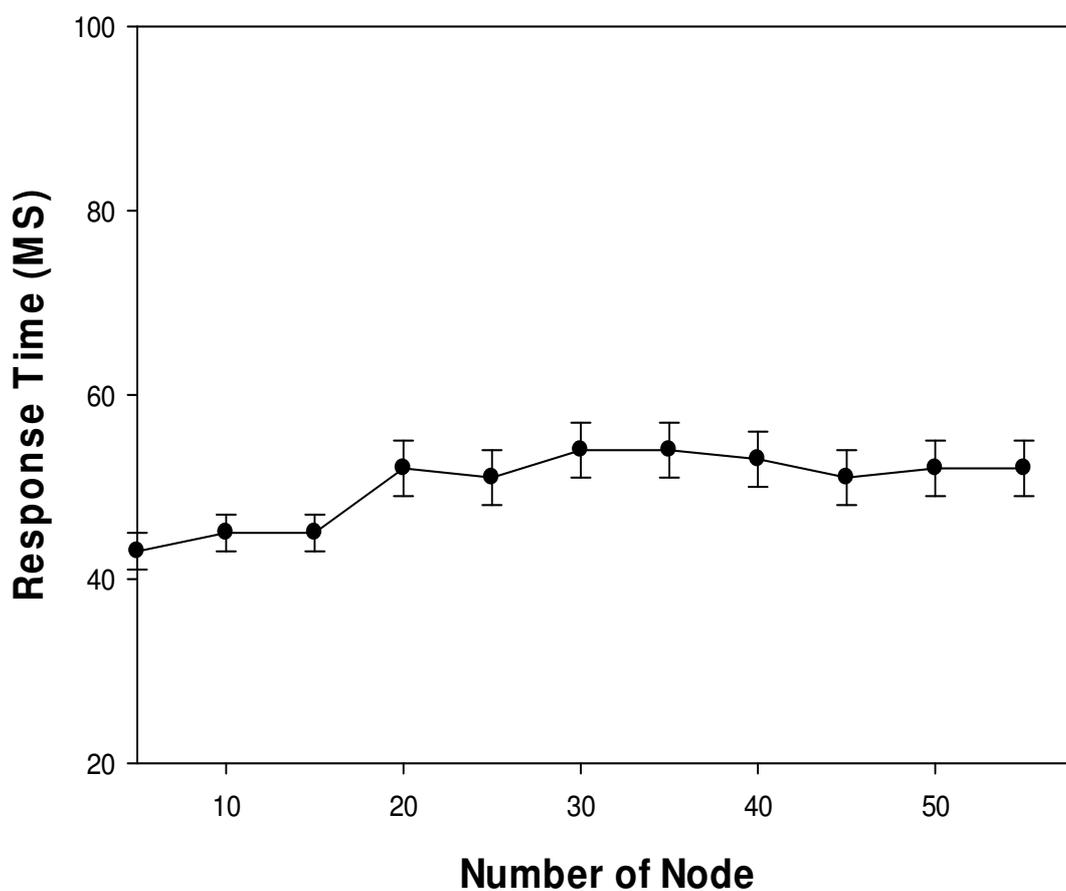


Figure 6.8 Response Time

6.8 Experiment Four: The Self-Load Balancing

In these experiments, we try to approve that load balancing is the process of roughly equalizing the workload among all MPs included in the autonomic system according to MPs power to produce a global improvement in system performance. As shown in figure 6.9 to figure 6.14.

6.8.1 Number of job equals 100 jobs

In figure, 6.9 below when assigned 100 jobs to be processed, the confidence interval at 95% CI for 100 jobs are distributed normally mapping with numbers of MPs.

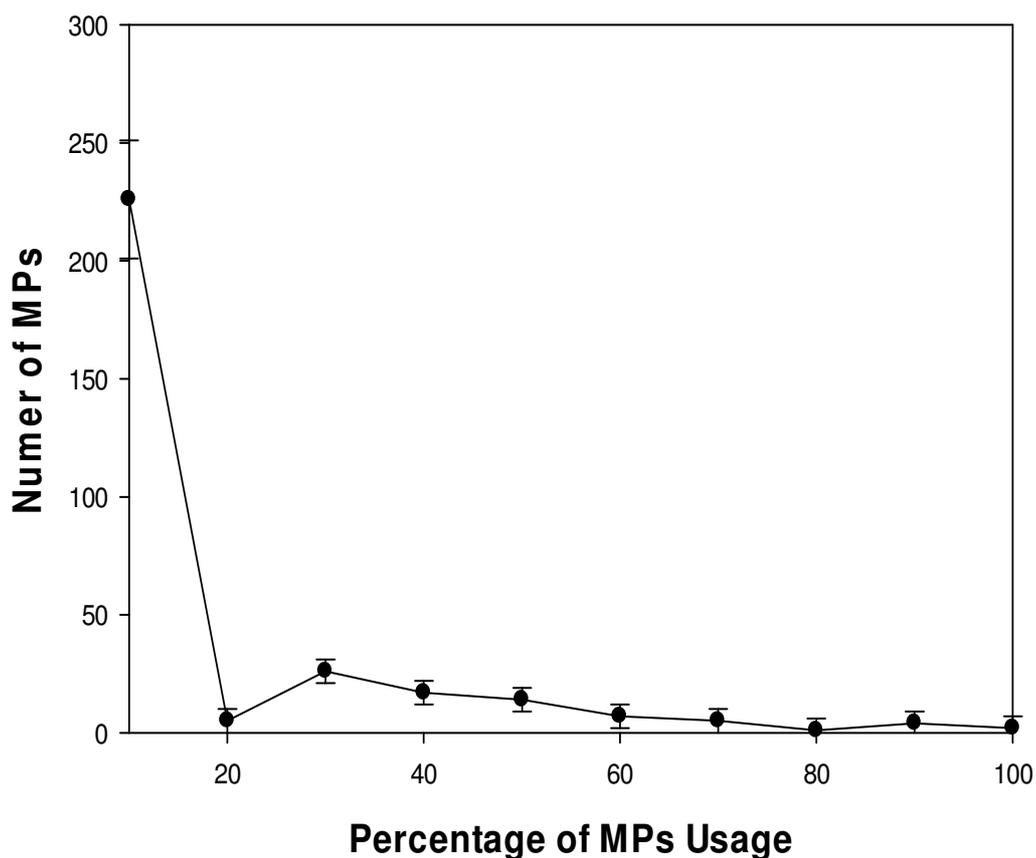


Figure 6.9 percentage usage of MPS when number of job equal 100 jobs

6.8.2 Number of job equal 200 jobs

In figure, 6.10 below when assigned 200 jobs to be processed, the confidence interval at 95% CI for 200 jobs are distributed normally mapping with numbers of MPs.

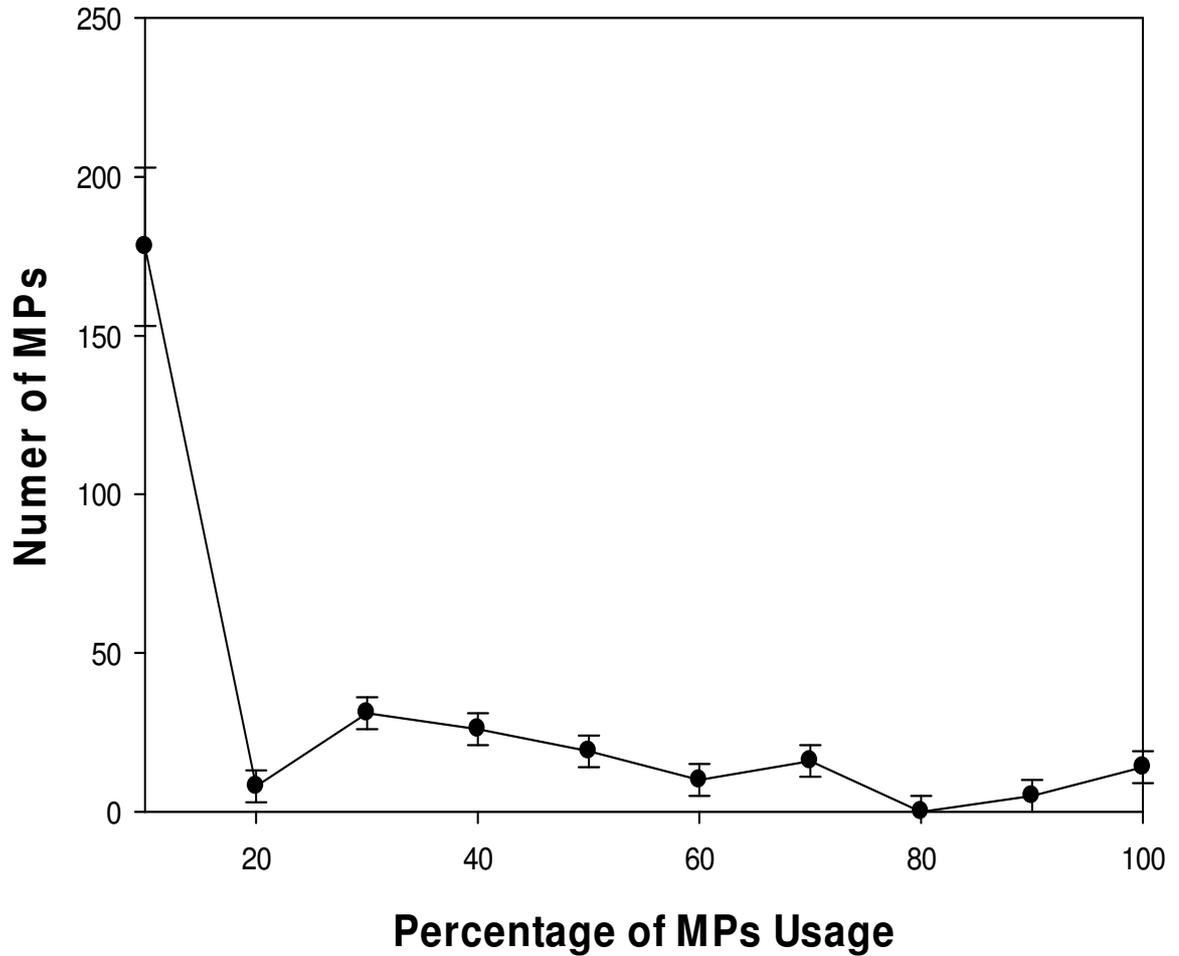


Figure 6.10 percentage usage of MPS when number of job equal 200 jobs

6.8.3 Number of job equal 300 jobs

In figure, 6.11 below when assigned 300 jobs to be processed, the confidence interval at 95% CI for 300 jobs are distributed normally mapping with numbers of MPs.

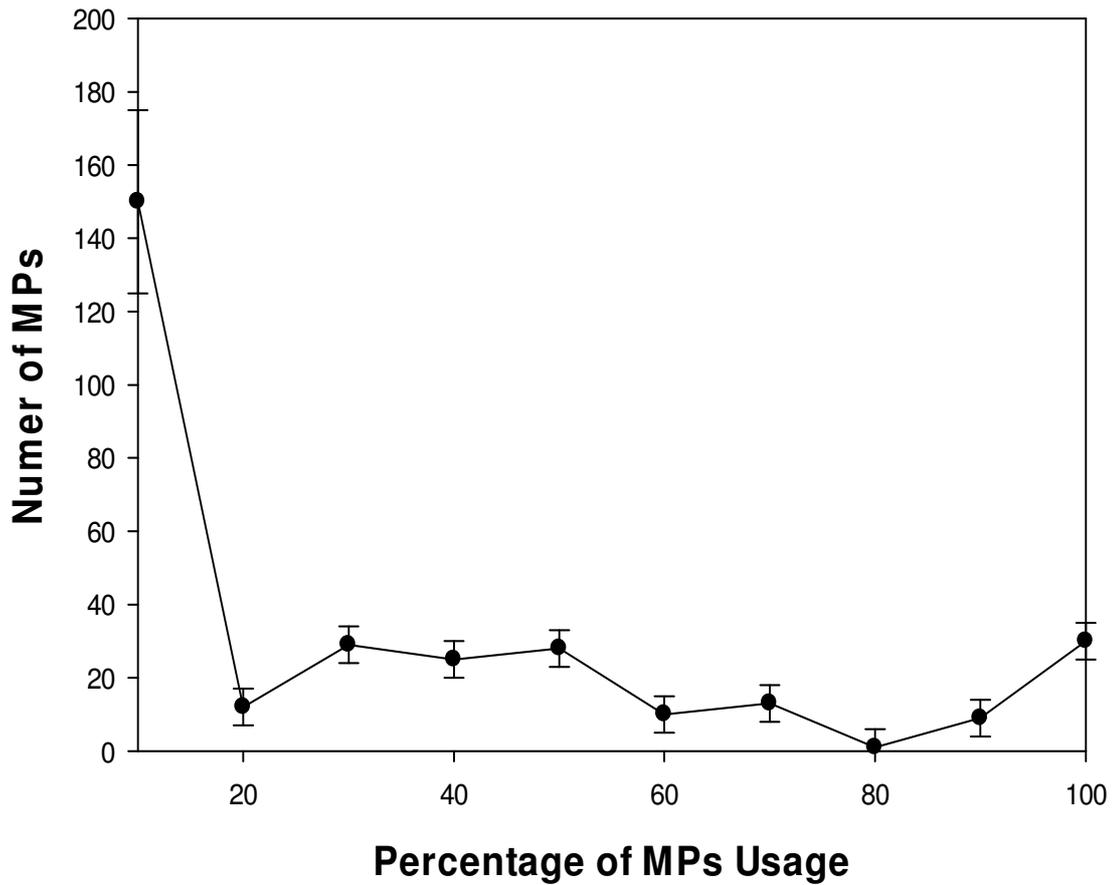


Figure 6.11 percentage usage of MPS when number of job equal 300 jobs

6.8.4 Number of job equal 400 jobs

In figure, 6.12 below when assigned 400 jobs to be processed, the confidence interval at 95% CI for 400 jobs are distributed normally mapping with numbers of MPs.

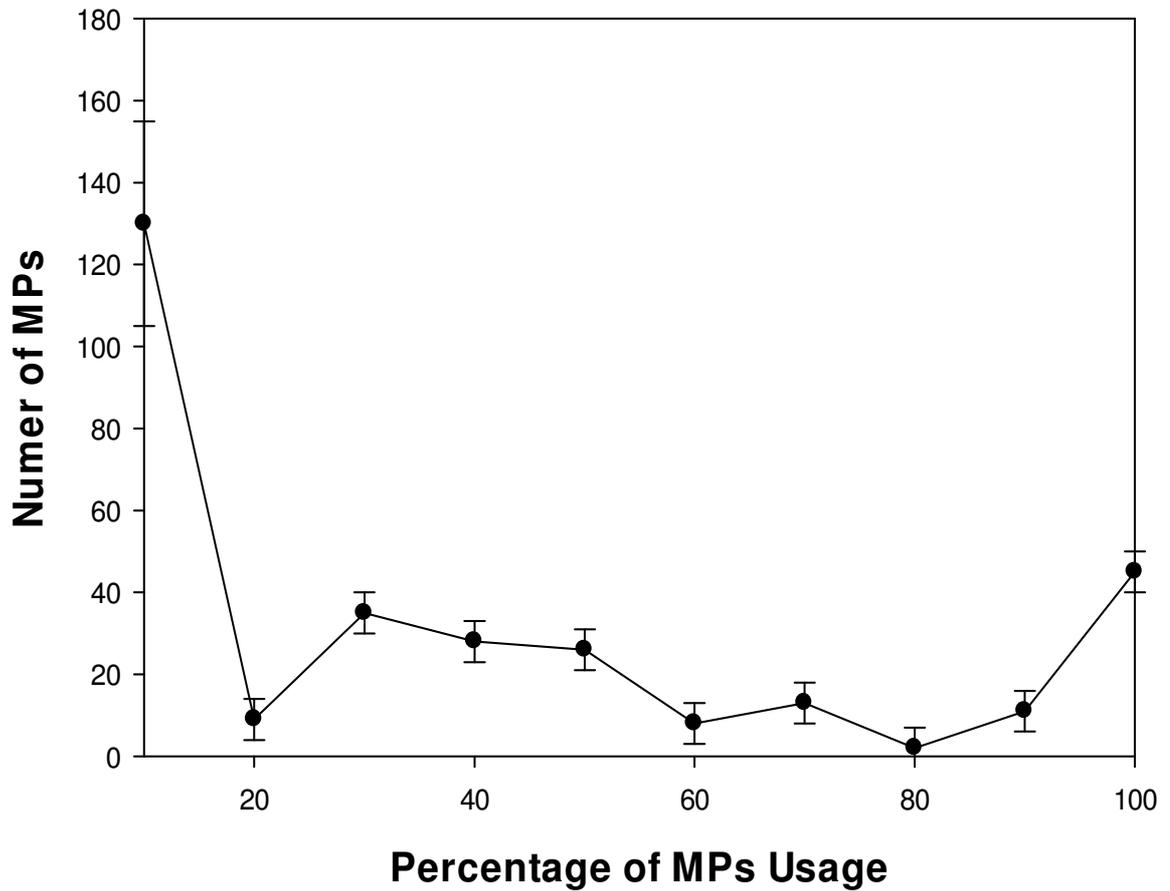


Figure 6.12 percentage usage of MPS when number of job equal 400 jobs

6.8.5 Number of job equal 500 jobs

In figure, 6.13 below when assigned 500 jobs to be processed, the confidence interval at 95% CI for 500 jobs are distributed normally mapping with numbers of MPs.

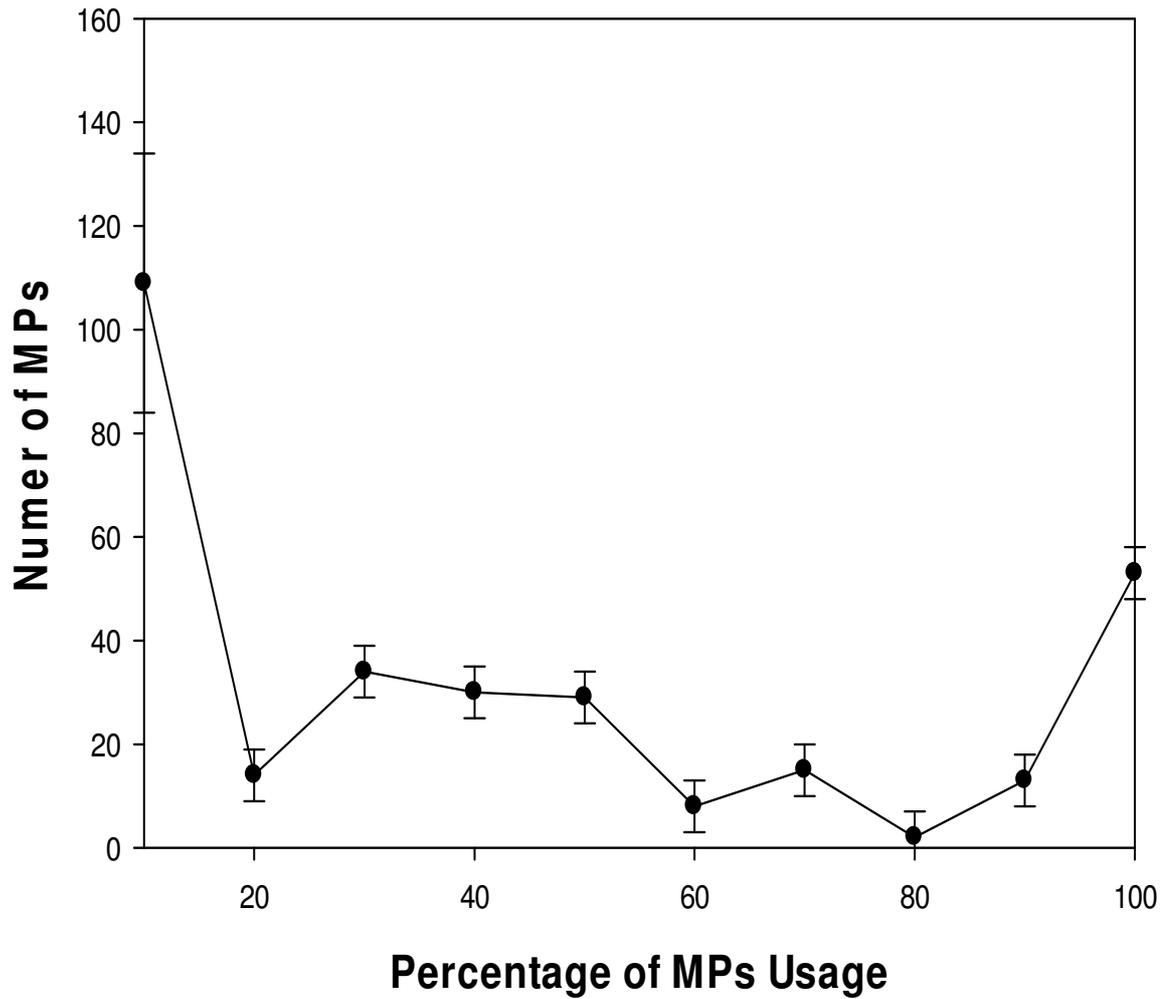


Figure 6.13 percentage usage of MPS when number of job equal 500 jobs

6.8.6 Number of job equal 600 jobs

In figure, 6.14 below when assigned 600 jobs to be processed, the confidence interval at 95% CI for 600 jobs are distributed normally mapping with numbers of MPs.

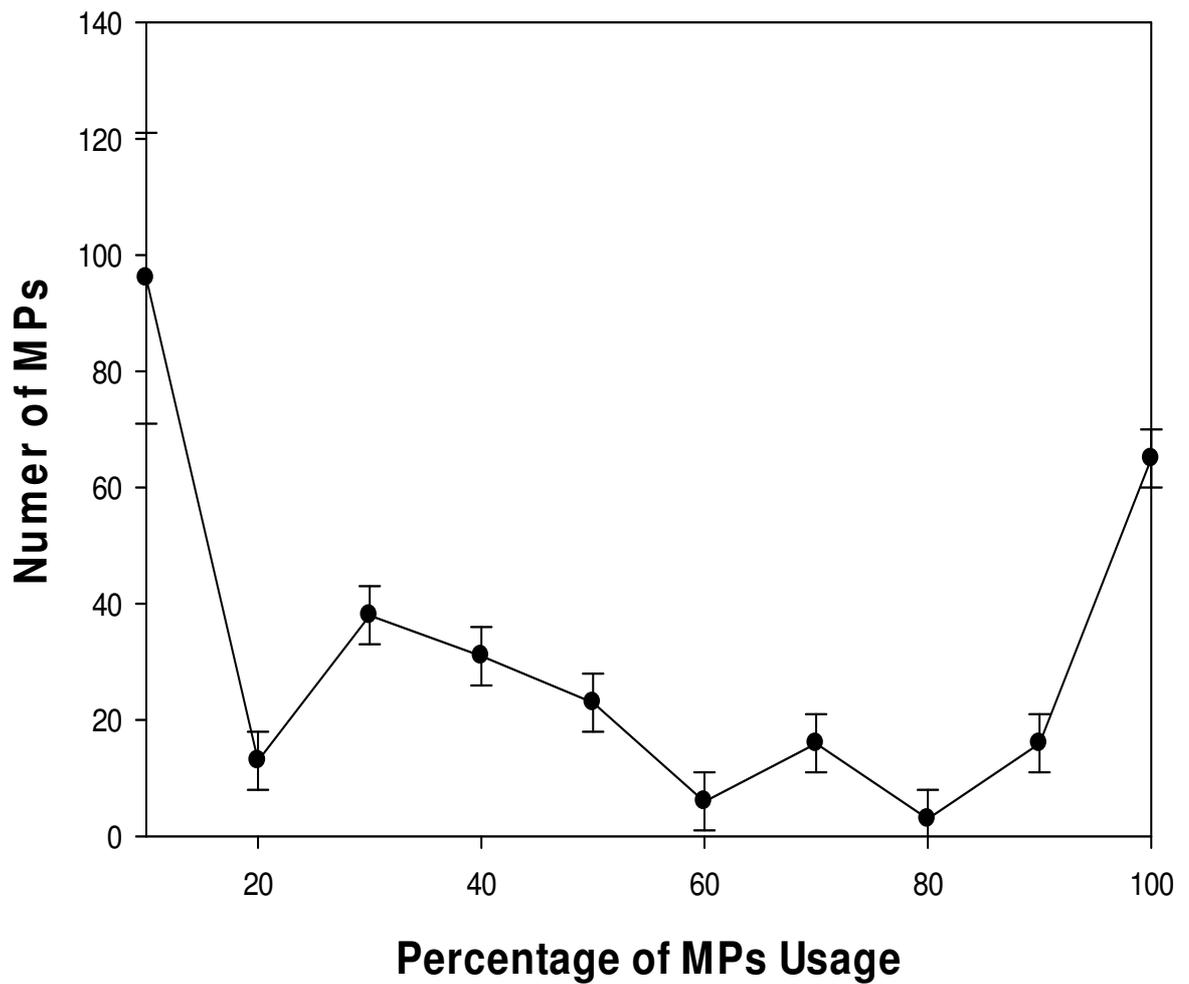


Figure 6.14 percentage usage of MPS when number of job equal 600 jobs

Chapter Seven

7

V & V of designing

a SLB Scheme

7.1 Overview

Currently, Simulation models are increasingly being used in problem solving and aid in decision-making. The developers and users of these models, the decision makers using information obtained from the results of these models, and the individuals affected by decisions based on such models are all rightly concerned with whether a model and its results are “correct”. This concern is addressed through model verification and validation. Model verification in such research case is often defined as “ensuring that the designed algorithm model and its implementation are correct. Model validation is usually defined to mean “substantiation that the designed algorithm within its domain of applicability possesses a satisfactory range of accuracy consistenting with the intended application of the model” (Schlesinger et al. 1979) and is the definition used here. A model sometimes becomes accredited through model accreditation. Model accreditation determines if a model satisfies specified model accreditation criteria according to a specified process.

In this chapter, we discuss verification and validation of simulation models based algorithms. Therefore, the verification and validation steps used in chapter seven for the designing of a self-load balancing that can be fairly implemented by a simulation at the system level for the designed algorithm procedure and their methods to ensure if the designed algorithm meets the initial design requirements and specification as well as the input and output to simulating process.

In software engineering, the verification defines the quality assurance process intended to check that a service meets a set of initial design requirements, specifications, and regulations. Which means (did you build the load self-balancing algorithm right in/with the system (business dictionary 2010).

However, the validation defines the quality control process intended to check that the development and verification procedures for a load balancing scheme results meets the initial defined requirements, specifications, and regulations. In other meaning, (are you built the right algorithm in/with the system) (Wikipedia website 2010).

Quality control and quality assurance are important concepts the first one refers to the quality related activities associated with the designed algorithm. It is the systematic measurements comparison with a standard, monitoring of processing as it may include dead fit for purpose and right first time. While the quality assurance the second one refers to the process used to create the deliverable algorithm (Wise geek website 2010).

This chapter is organized as follows. Section 2 presents a verification of the self-load balancing for the designed algorithm scheme. Section 3 presents a validation of the self-load balancing for the designed algorithm scheme. Finally, a summary is presented in section 3.

7.2 Verification of the Self-Load Balancing Based Algorithm

There are two basic approaches for deciding whether a simulation model of the self load balancing based algorithm is valid or not. In case of this research study, the designing approach and input methods used for the designed algorithm is very important to verified and the second approach requires the model development to conduct verification and validation as part of the model development process, which is discussed below. The first approach, and a frequently used one, is for the verification for the designed algorithm components to make the decision later on whether a simulation model for the designed algorithm is valid or not. A subjective decision is made based on the results of the various evaluations conducted as part of the model development process. The verification steps of the self-load balancing algorithm and its components include the following:

7.2.1 Verification of the Building DQT

The verification step for the two dimensional geometric region should include indexing and partitioning. The verification results for this step is as follows:

- The indexing and partitioning are defined on the biases of the X,Y coordinates of the defined two dimensional array and limited with 2^n with one overlay for all sizes, For example if the defined two dimensional array are including 3 nodes, the array size should be $2^1 * 2^1$ the maximum number of nodes should not more than 4 nodes, however, if we have 16 nodes, the size of the array should be $2^2 * 2^2$, in addition, if the number of nodes for instance are 17, the size should be exceeded to $2^3 * 2^3$, that means the maximum number of nodes should be included 64 nodes.

- If we already have defined the number of nodes in a specific geographical location (i.e. $2^2 * 2^2$) and the number of nodes exceed the defined partitioning, in this case, the reconstruct of the partitioning are not applicable if the number of nodes are not exceeded to 62 nodes. So, any new node that wants to join the defined overlay by sending its information by a message to 3 hops to insure the arrived its parent then the parent checks the index if it existed or not by using IP address and replay with change index by adding literal from A-Z, if the number of nodes are 63 or above in this case, the reconstruct of new distribute quadtree are needed for $2^3 * 2^3$.
- In the previous two paragraphs, the defined nodes specify their self types on the biases of the node specification, for instance if node 1 identifies its type with 0 this means this node type is normal. Otherwise, if the node type specified their types between 1 to 4, this mean the nodes types are considered as media ports. From the literature we considered in the designed algorithm. There are 30% of the defined nodes of the constructed DQT have Mediaports (IBM Corporation, 2006) & (Al-Oqily, & Karmouch, 2008).

7.2.2 Verification of the Self-Load Balancing

The verification step for the Self-Load Balancing should include power percentage verification and incoming edge verification. The verification results for this step as follows:

- The self-load balancing is built as upper layer based on the DQT layer see the previous section.

- The power for the media port (media node) can locally be calculated based on the (CPU, Memory, Bandwidth, and Bus speed). In the previous published work bus speed component are not taken into their consideration, but they are considered the hard disc, in case calculating the hard disc percentage the highest power obtained belong to the hard disc because its value is always big. Moreover, when we want to get actual power for any node within the network you should not neglect the bus speed because its important factor that determined the power
- The power percentages are calculated based on a defined power of each media port or media node divided by the highest media port multiplying with 100% on the defined DQT.
- To identify the incoming edges, we dived the defined power by 12.5 to produce as a maximum 8 edges to construct the self load-balancing overlay with a minimum effective load in the defined DQT as well as the algorithm defined the minimum edge with 4 for strong connected components.
- When a new job enters the network, the media node provides the service and finds out the highest incoming edges. If it cannot provide the service then MPs forward the request to the nearest MPs to serve this job; otherwise, calling the Delete Edge Procedure to decrease its connectivity.
- After finishing the job and exiting from the network, then call Add Edge Procedure to increase its connectivity.

7.3 Validation of the self-load balancing algorithm

A Conceptual model validation based simulation is defined as determining that the theories and assumptions underlying the conceptual model are correct and that the model representation of the problem entity is reasonable for the intended purpose of the model. The validation steps of the self-load balancing algorithms include the following:

7.3.1 Validating The DQT Building Message for Each Level

The following data are used as part of the validation for the designing algorithm from level 1 to level 5 with the maximum number of messages (27500) and with the minimum with 107 messages

- Level 1 which is the leaf level need 27500 message to construct them
- Level 2 parent1 level need 6875 message to construct them
- Level 3 parent2 level need 1718 message to construct them
- Level 4 parent3 level need 430 message to construct them
- Level 5 parent4 level need 107 message to construct them

Our validating result based on the collected data papers that the messages cost decrease by quarter when we move from level to level which means that DQT building message could be useful for reducing the cost using the DQT building message.

7.3.2 Validating The Stretch And Response Time:

The following data are used to validate the number of hops taken by an overlay packet divided by the number of hops when using an ip layer path between the same source and destination. C2 and C3. While the C4 uppers the average response time for enquiries

Table 7.1 show the number of queerer node, direct hop, DQT hop and response time of the simulation model

Number of Queerer	Direct Hop	DQT Hop	Response Time
5	5	20	125
10	7	18	159
15	7	18	166
21	8	17	156
27	9	16	143
32	8	15	131
37	8	14	121
42	8	15	119
48	8	15	118
54	9	16	119
60	9	16	117
65	9	16	114
71	9	15	107
80	9	15	100
85	9	15	98
90	9	15	98
97	9	15	101
104	10	15	102
109	9	15	105
115	9	15	106

7.3.3 Validating algorithm for Joined and leave Node

The following data are used to validate the joined and leaved node. C2 for joining nodes and its load where C3 illustrates the joining nodes with corresponding response time. C4 for leaving nodes and its load. While C5 refer to leaving nodes with response time.

Table 7.2 show the number of Join, Left node, network load and response time of the simulation model

Number of node	Join network load	Join response time	Leave network load	Leave response time
5	17	34	5	43
10	17	51	5	45
15	17	60	5	45
20	22	56	5	52
25	21	50	5	51
30	23	50	5	54
35	24	46	5	54
40	23	43	5	53
45	23	40	5	51
50	23	38	5	52
55	23	40	5	52
60	23	43	5	51

7.4 Discussion and Results

This section describes various validation techniques used in model verification and validation. Most of the techniques described here are found in the literature, although some may be described slightly differently. They can be used either subjectively or objectively. By “objectively,” we mean using some type of mathematical procedure, for more details see chapter 5., and a hypothesis tests or confidence intervals for a simulation or experimental test for more details see chapter 6.

A combination of techniques is generally used. These techniques are used for verifying and validating the sub algorithm models and the overall model.

1. Animation: The algorithm model and sub model operational behaviour is displayed graphically as the model moves through time. For example the movements of parts through a factory during a simulation run are shown graphically see chapter 6.
2. Comparison to Other Models: are not used while the research idea is slightly innovative, and there is no schemes or algorithms designed before for such research problem.
3. Degenerate Tests: The degeneracy of the model’s behaviour is tested by appropriate selection of values of the input and internal parameters. For example, section 2, a verification of the load-self balancing algorithm and their components.
4. Event Validity: The “events” of occurrences of the simulation model in chapter 6 are subjectively compared to those of the real system to determine if they are similar. For example, the number of queerer node, direct hop, DQT hop and response time of the simulation model as well as the Join, Left node, network load and response time in the simulation.

5. Face Validity: Asking individuals knowledgeable about the designed algorithm of the load balancing scheme and its behaviour are reasonable. For example, is the logic in the conceptual model correct and are the model's input-output relationships reasonable.
6. Historical Methods: The two historical methods of validation are rationalism, and empiricism. Rationalism assumes that everyone knows whether the underlying assumptions of a model are true, Logic deductions are used from these assumptions to develop the correct (valid) model. In the design algorithm based simulation achieves this method through a formal representation of the self load balancing scheme, see chapter 5 . and Empiricism requires every assumption and outcome to be empirically validated are achieved (shown in chapter 6).
7. Internal Validity: of the design algorithm are represented in chapter 5 and 6 as well as (runs) of a stochastic model are made to determine the amount of (internal) stochastic variability in the model, see table 7.1 and table 7.2 in the previous section.
8. Predictive Validation: The designed algorithm model in this thesis is used to predict (forecast) the self load balancing efficiency of the distributed behaviour based geographical region by predictive logically indexing and partitioning the DQT then build their predictive of the network self load balancing view.
9. A Conceptual model validity for the designed algorithm built in this these is determines that (1) the theories and assumptions underlying the conceptual model are correct and (2) the model's representation of the problem entity and the model's structure, logic, and formal and causal relationships are "reasonable" in chapter 5 for the intended purpose of the model.

Chapter Eight

8

Conclusion

and

Future Work

8.1 Conclusion:

Load balancing is a challenge due to the increased complexity, cost, and heterogeneity of current technologies. This work, reviewed the state of the art in load balancing techniques. This study shows that traditional techniques are not adequate to face the ever-increasing challenge and complexity of technology. A Self Load Balancing in Autonomic Overlay Networks is proposed. The scheme employs the spatial index and partitions the network to build a distributed quad-tree. Another logical layer is built based on the available resources. This layer is connecting resources of the same type thus facilitating the process of load balancing. The local knowledge is exploited to achieve a better performance. A simulation tool has been used to test the proposed method and to quantify its cost and efficiency. Results show that MPs overlay can efficiently balance MPs load. The experiments take into account the importance of heterogeneity in available computing resources and extra burst in incoming jobs.

8.2 Future work:

In the future, we plan to reflect our proposed scheme and adopt it to wireless sensor networks (WSN), and rather than using DQT, we plan to investigate the use of Octree in a distributed manner.

Referencias

- Aboulnaga, A., & Aref, W.G. (2001) Window query processing in linear quadtrees. *Citeseerx Distributed and Parallel Databases*,10,2.
- Abrahamsson, H., & Gunnar, A. (2004) Traffic engineering in ambient networks: challenges and approaches. *In the Second Swedish National Computer Networking Workshop (SNCNW)*, Karlstad.
- Alakeel, A. (2010) A guide to dynamic load balancing in distributed computer systems. *International Journal of Computer Science and Network Security IJCSNS*, 10,6.
- Alavi, M.h., & Hariri, B., Mohammadi, S.S. (2009) Using geometrical routing for overlay networking in mmogs. *Journal Multimedia Tools and Applications*,45.
- Alexander, G.A., Frens,J.D., Gu,Y., & Wise,D.S. (2001) Language support for morton-order matrices. *ACM Proceedings Of The Eighth SIGPLAN Symposium On Principles And Practices Of Parallel Programming (Ppopp)*,24-33.
- Al-Oqily, I., & Karmouch, A. (2008) A self-organizing composition towards autonomic overlay networks. *IEEE Network Operations and Management Symposium (NOMS)*, 287-294.
- Al-Oqily, I., Karmouch, A., & Glitho, R. (2008) An Architecture for Multimedia Delivery Over Service Specific Overlay Networks. *Springer Boston IFIP Conference on Wireless Sensor and Actor Networks II*, (264) 97-112.
- Amis, A.D., & Prakash, R. (2000) Load-balancing clusters in wireless ad hoc networks. *Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 25-32.
- Ang, ch., & Samet, H. (1989) Node distribution in a pr quadtree. *Citeseer In Proceedings 1st International Symposium On Large Spatial Databases*, 233—252.
- Apache camel website (2010) (on-line) available:<http://camel.apache.org/load-balancer.html>
- Arcor website (2009) (on-line), available <http://home.arcor.de/jensaltmann/jsim-e.htm>
- Avin, C., Dvory, Y., & Giladi, R. (2011) Geographical quadtree routing. *IEEE Computers and Communications (ISCC)*, 302 -308.
- Backhaus, H., & Krause, S. (2010) Quon – a quad-tree based overlay protocol for distributed virtual worlds. *ACM Journal In IJAMC*, 4 (2), 126-139.
- Badidi, E. (2000) *Architecture and services for load balancing in object distributed systems*, (Published doctoral dissertation), Faculty of High Studies, University of Montreal, Montreal,Canada.
- Bauer, B. Gufler, B., Kemper, A., Kuntschke, R., Reiser, A., & Scholl, T. (2008) Scalable community-driven data sharing in e-science grids. *Journal In Future Generation Comp. Syst*, 25 (3), 290-300.

- Berg, M.D., Haverkort, H.J., Thite, S., and Toma L. (2010) Star-quadtrees and guard-quadtrees: i/o-efficient indexes for fat triangulations and low-density planar subdivisions. *ACM Computer Geometry*, 43 (5), 493-513.
- Berk, V., Cybenko, G., & Roblee, C. (2005) Implementing large-scale autonomic server monitoring using process query systems. *Proceedings Second International Conference On Autonomic Computing*, 123-133.
- Beygelzimer, A., Kakade, S., & Langford, J. (2006) Cover trees for nearest neighbor. *Italian Campaign Master List (ICML)*.
- Bhattacharjee, B., Han, B., Levin, D., Lumezanu, C., & Spring, N. (2010) Don't love thy nearest neighbor. *Citeseer Proceedings IPTPS'10 Proceedings Of The 9th International Conference On Peer-To-Peer Systems*.
- Bridgewater, J.S.A., Boykin, P.O., & Roychowdhury, V.P. (2007) Balanced overlay networks (bon): an overlay technology for decentralized load balancing. *IEEE Parallel and Distributed Systems*, 18 (8), 1122 -1133.
- Buragohain, C. and Agrawal, D. and Suri, S. (2006) Distributed navigation algorithms for sensor networks. *IEEE International Conference on Computer Communications. Proceedings 25th INFOCOM*, 1 -10.
- Business dictionary website (2010) (on-line), available: <http://www.businessdictionary.com/definition/verification.html>
- Casavant, T.L. & Kuhl, J.G. (1988) A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14 (2), 141 -154.
- Chang, S., & Smith, J.M. (1994) Quad-tree segmentation for texture-based image query. *Proceedings of the Second ACM International Conference on Multimedia*, 279-286.
- Chess, D.M., & Kephart, J.O. (2003) The vision of autonomic computing. *Computer*, 36 (1), 41-50.
- Chhabra, A., & Singh, G. (2006) Qualitative parametric comparison of load balancing algorithms in distributed computing environment. *IEEE International Conference on Advanced Computing and Communications (ADCOM)*, 58 -61.
- Clem, K., Luetzgen, M.R., W. and Willsky, A.S. (1994) Efficient multiscale regularization with applications to the computation of optical flow. *IEEE Transactions On Image Processing*, 3 (1), 41 -64.
- Corson, M.S., & Park, V.D. (1997) A highly adaptive distributed routing algorithm for mobile wireless networks. *IEEE Proceedings Sixteenth Annual Joint Conference Of The IEEE Computer And Communications Societies (INFOCOM)*, 3, 1405 -1413.
- Dalal'ah, A. (2006) A dynamic sliding load balancing strategy in distributed systems. *The International Arab Journal Of Information Technology (IAJIT)*, 3, (2).

- De Leenheer, M., Farahmand, F., Thysebaert, P., Volckaert, B., De Turck, F., Dhoedt, B., Demeester, P., & Jue, J. (2005) Anycast routing in optical burst switched grid networks. *IEEE 31st European Conference On Optical Communication(EOC)* , 3, 699 – 700.
- Dehne, F., Ferreira, A.G. and Rau-chaplin A.(1991) Parallel processing of pointer based quadtrees on hypercube multiprocessors. *Citessr International Conference On Parallel Processing*.
- Deldari, H., & Salehi, M.A., (2006) A novel load balancing method in an agent-based grid. *IEEE International Conference On Computing Informatics*, 1 -6.
- Demirbas, M., & Xuming Lu. (2007) Distributed quad-tree for spatial querying in wireless sensor networks. *IEEE International Conference On Communications (ICC)*, 3325 -3332.
- Deqiang, C. , Haenggi, M. and Nicholas, L., J. (2007) Distributed Spectrum-efficient routing algorithms in wireless networks. *IEEE 41st Annual Conference On Information Sciences And Systems(CISS)*, 649 -654.
- Dooley, R. (2004) A distributed quadtree dictionary approach to multi-resolution compression. *IEEE International Conference On Information Technology: Coding And Computing*, 2, 155-156.
- Eager D.L., Lazowski E.D., & Zahorjan J. (1986) Adaptive load sharing in homogeneous distributed systems. *IEEE Trans Software Eng.*, 12 (5), 662-675.
- Eisenstat, D. (2011) Random road networks: the quadtree model. *The Eighth Workshop On Analytic Algorithmics And Combinatorics (ANALCO)*, 76–84.
- e-learning website (2011) (on-line) available:http://e-learning.mfu.ac.th/mflu/1301103/data_process/web1_files/right_files/web6.html
- Ephremides, A., & Michail, A. (1998) A distributed routing algorithm for supporting connection-oriented service in wireless networks with time-varying connectivity. *IEEE Symposium On Computers And Communications*, 587 -591.
- Eppstein, D., Goodrich, M.T., & Sun,J.Z. (2005)The skip quadtree: a simple dynamic data structure for multidimensional data. *ACM In Proc. 21st ACM Symposium On Computational Geometry*, 296-305.
- Falchi, F., Gennaro, C., Rabitti, F., & Zezula, P. (2007) A distributed incremental nearest neighbor algorithm. *ACM Proceedings Of The 2nf International Conference On Scalable Information Systems*, 82.
- Flatebo, M., Datta, A.K. & Bourgon, B. (1994) Self-stabilizing load balancing algorithms. *IEEE 13th Annual International Phoenix Conference On Computers And Communications*, 303.
- Frens, J.D., & Wisey, D.S. (1999) *Morton-order matrices deserve compilers support*.(**Technical Report 533**).

Fuse source website (2010) (on-line), available:http://fusesource.com/docs/ide/camel/1.0/eip/MsgRoutLoadBalancer.html#_IDU_LoadBalancer_HSH_Failover

GIS website (2011) (on-line), available:<http://www.gis.com>.

Gorman R.M., Popinet, S., Rickard G. J., & Tolman H. L. (2010) A quadtree-adaptive spectral wave model. *Journal Ocean Modelling*, 34, 36-49.

Goscinski, A. (1991) *Distributed operating systems: the logical design*, California: Addison-Wesley Pub. Co.

Hariri, S., & Parashar, M. (2004) Autonomic computing: an overview. *In UPP*, 257–269.

Har-Peled, S. (2006) *Geometric Approximation Algorithms*.

(on-line), available:<http://graphics.stanford.edu/courses/cs468-06-fall/Papers/01%20har-peled%20notes.pdf>

Harwood, A., Samet, H., & Tanin, E. (2005) A distributed quadtree index for peer-to-peer settings. *CiteSeer In Proceedings of the International Conference on Data Engineering (ICDE)*, 254-255.

Harwood, A., Samet, H., & Tanin, E. (2005) A distributed quadtree index for peer-to-peer settings. *CiteSeer In Proceedings of the International Conference on Data Engineering (ICDE)*, 254-255.

Hernandez, C., Marin, M. & Rodriguez, M.A. (2008) A p2p meta-index for spatio-temporal moving object databases. *Springer 13th International Conference On Database Systems For Advanced Applications*, 653-660.

Hou, J.C., Sobeih, A., & Tyan, H.Y. (2005) Towards composable and extensible network simulation. *IEEE International Proceedings Parallel and Distributed Processing Symposium*, 8.

IBM Corporation, (2006). *An architectural blueprint for autonomic computing*. White Paper. Armonk, New York, United States.

Israr, N., & Awan, I. (2007) Multihop clustering algorithm for load balancing in wireless sensor networks. *International Journal of Simulation Systems Science and Technology*.

Jiang, C. & Zhang, J. (2007) A load balancing technology oriented to common service information system, *International Conference On Service Systems And Service Management*, 1-6.

Jsim official website (2010) (on-line), available <http://sites.google.com/site/jsimofficial/>

J-sim website (2008) (on-line), available <http://www.j-sim.zcu.cz/>

Kang, M., Laurini, R., Li, K., & Servigne, S. (1999) Indexing field values in field oriented systems: interval quadtree. *CIKM*, 335-342.

- Kannan, G., Merchant, S.N., & Desai, U.B. (2007) Cross layer routing for multihop cellular networks. *IEEE 21st International Conference On Advanced Information Networking And Applications Workshops (AINAW)*, 2, 165 -170.
- Karatza, H.D. (1994) Job scheduling in heterogeneous distributed systems. *Journal. Of Systems And Software*, 56, 203–212.
- Khalid, A., Haye, M.A., Khan, M.J. & Shamil, S. (2009) Survey of frameworks, architectures and techniques in autonomic computing. *Fifth International Conference on Autonomic and Autonomous Systems(ICAS)*, 220-225.
- Khan, M., Pandurangan, G., & Anil Kumar, V.S. (2009) Distributed algorithms for constructing approximate minimum spanning trees in wireless networks. *IEEE Transactions On Parallel And Distributed Systems*, 20 (1), 124 -139.
- Kitchen table computers website, (2010) (on-line), available:<http://www.kitchentablecomputers.com/processor2.php>
- Ku W., Nguyen, T., Wang, H., & Zimmermann, R. (2006) Annatto: adaptive nearest neighbor queries in travel time networks. *International Conference On Mobile Data Management*, 50.
- Lario, R., Antonijuan, M., & Pajarola, R.(2002) Quadtree based triangulated irregular networks. *IEEE Visualization*, 395 -402.
- Lohman, G. M. and Lightstone S. S.(2002) Smart:Making db2 (more) autonomic. *In Proceedings Of The 28th International Conference On Very Large Data Bases VLDB Endowment*, 877–879.
- Manolopoulos, Y., Tzouramanis, T., & Vassilakopoulos M. (2000) Overlapping linear quadtrees and spatio-temporal query processing. *Journal In Comput. J.*,43 (4), 325-343.
- Mazumder, P.,(1987) Planar decomposition for quadtree data structure. *ACM Computer Vision, Graphics, And Image Processing*,38, (3), 258-274.
- Meng, L., Qiu, X., Zhang, H., & Zhang, X. (2010) Design of distributed and autonomic load balancing for self-organization. *IEEE 72nd Vehicular Technology Conference Fall (VTC 2010-Fall)*, 1-5
- Mir, Z.H. & Ko,Y. (2006) A quadtree-based data dissemination protocol for wireless sensor networks with mobile sinks. *International Conference On Personal Wireless Communications*, 447-458.
- Mundia, L.C. (2007) "Strengthening the Importance of Geographical Information Systems (GIS) Components in Organisations for a Successful GIS Technology" (On-Line), available: http://www.gim-international.com/download/whitepaper_uploadfile_3.pdf.
- NICTA website (2010) (on-line), available:http://www.nicta.com.au/research/project_list/completed_projects/ambient_networks

- Oliver, M. A., & Wiseman N. E. (1983) Operations on quadtree leaves and related image areas. *The Computer Journal*, 26 (4), 375-380.
- Openxtra website (2007) (on-line), ailable <http://www.openxtra.co.uk/articles/network-simulation>
- Palanisamy, V., Baskaran, K., & Prabeela. S. (2011) Efficient distributed clustering-based anomaly detection algorithm for sensor stream in clustered wireless sensor networks. *European Journal of Scientific Research*, 54 (4), 484-498.
- Physiome website (2011) (on-line), available: <http://www.physiome.org/jsim/>
- Picone, M., Amoretti, M., & Zanichelli, F. (2010) Geokad:a p2p distributed localization protocol. *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 800 -803
- Ramakrishnan, K. (2005). *An improved model for the dynamic routing effect algorithm for mobility protocol*, (Published Master's Thesis) Ontario University, Ontario, Canada.
- Samet, H., & Shaffer, C.A. (1986) Optimal quadtree construction algorithms. *Computer Vision, Graphics, And Image Processing*, 37, 3, 402-419.
- Samet,H. (1984) The quadtree and related hierarchical data structures. *Journal in ACM Computing Surveys (CSUR)*, 16 (2).
- Saravanakumar, E., & Prathima, G. (2010) A novel load balancing algorithm for computational grid. *IEEE International Conference On Innovative Computing Technologies (ICICT)*, 1 -6.
- Schuster, G.M. and Katsaggelos, A.K.(1998) An optimal quadtree-based motion estimation and motion-compensated interpolation scheme for video compression, *IEEE Transactions On Image Processing*,7 (11), 1505 -1523.
- scribd website (2011) (on-line), available: <http://www.scribd.com/explore>.
- Sharma, M., Sharma, S., & Singh, S.(2008) Performance analysis of load balancing algorithms. *Proceedings Of World Academy Of Science Engineering And Technology*, 38.
- Singh, M., & Suri, P.K. (2010) An efficient decentralized load balancing algorithm for grid. *IEEE 2nd International Advance Computing Conference (IACC)*, 10 -13.
- Slimani, Y., & Yagoubi, B. (2006) Dynamic load balancing strategy for grid computing. *Citeseerx World Academy Of Science, Engineering And Technology*.
- Sumengen, S., & Balcisoy,S. (2005) Real-time simulation of autonomous vehicles on planet-sized continuous lod terrains. *The 13-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Pilsen, Czech Republic*.

Tanin, E. (2009) Hierarchical data summarization. **Springer Encyclopedia of Database Systems**, 1300-1304.

Tayeb, J., Ulusoy, Ö., & Wolfson O. (1998) A quadtree-based dynamic attribute indexing method. *Citeseer Computer Journal* 41 (3), 185-200.

Tyan, H.Y., (2002). *Design, realization and evaluation of a component-based compositional software architecture for network simulation*, (Published doctoral dissertation) Department of Electrical Engineering, The Ohio State University, USA.

Wang T., & Morris, R.J.T.(1985) Load sharing in distributed systems. *IEEE Transactions On Computers*,34 (3) 204 -217.

Weber, D. (2007) *Distributed query processing for locality-aware data in p2p networks* (published master dissertation) Technische Universität München Fakultät für Informatik: Germany.

Wikipedia website (2010) (on-line), available:
http://en.wikipedia.org/wiki/Verification_and_validation

Wikipedia website (2010) (on-line),
available:http://en.wikipedia.org/wiki/Geographic_information_system

Wikipedia website (2010) (on-line), available:http://en.wikipedia.org/wiki/Ambient_network

Wise geek website (2010) (on-line), available:<http://www.wisegeek.com/what-is-quality-control.htm>

World News Website (2011) (on-line) available:<http://wn.com/quadtree>.

Yiwei Wu., & Yingshu Li. (2009) Distributed indexing and data dissemination in large scale wireless sensor networks. *IEEE Proceedings Of 18th International Conference On Computer Communications And Networks(ICCCN)*, 1 -6.