WEB SERVICE REPLICA SELECTION ANALYSIS USING A MULTIAGENT–BASED SIMULATOR

by

Khashayar Habibi

Bachelor of Science, Sharif University of Technology, Tehran, Iran, 1989

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2010

©Khashayar Habibi 2010

Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

WEB SERVICE REPLICA SELECTION ANALYSIS USING A MULTIAGENT–BASED SIMULATOR

Master of Science 2010

Khashayar Habibi

Computer Science

Ryerson University

Abstract

A Distributed system always runs on top of a computer network and cannot be separated from it. In many cases this network consists of hundreds or thousands of computers and processing nodes. An effective distributed system simulator needs to simulate the underlying network. Unfortunately a great majority of existing simulation tools are pure network simulators. Even though they are very effective for designing, and evaluating computer networks, they could not be used to simulate a distributed application like a web service based application. Many components in a distributed system are complex servers and software applications running on top of all network layers. Network simulators cannot simulate them. A higher-level simulator is required to simulate their behavior.

This work introduces an agent-based simulation model that integrates the simulation of a computer network and higher-level components of a distributed application. The distributed nature of agents makes them suitable to model and simulate distributed architectures including computer networks and distributed systems. To evaluate this approach the behavior of a replicated web service application will be simulated to show how effectively multi- agent-systems could be used to simulate the behavior of a distributed system.

Acknowledgements

This thesis could not have be done without help and support of my supervisor, Dr. Abdolreza Abhari, whose expertise, encouragement, and guidance helped me to achieve this task.

I am greatly thankful to Dr. Alexander Ferworn whose lectures, and advice, during my masters studies, helped me to overcome many challenges and obstacles that I encountered in my research.

I am very thankful to Dr. Cherie Ding whose profound knowledge of software engineering and Service Oriented Architectures gave me enough insight to the concepts that contributed substantially to my work.

I would like to thank Dr. Alireza Sadeghian for his continuos support and for introducing me to the field of machine intelligence, which gave me great motivation and energy to pursue my thesis in this very attractive field.

I am also very thankful to Dr. Marcus Santos whose excellent method of teaching and practical case studies gave me a deep and broad knowledge of genetic programming.

My parents thought me how to walk and gave me reasons why not to stop but I was not in this position without unlimited support of my beloved wife, Pardis and my two little angels Mojan and Kiana whose family times were sacrificed on many occasions in the past two years. This work is dedicated to all of my family.

Contents

Acronyms **xiii**

1 Objectives, Motivation, Contributions, and Scope **1**

1.1 Objectives 2

1.2 Motivation 2

1.3 Contributions 5

1.4 Scope 6

1.5 Thesis Outline 7

2 Background Information and Related Works **8**

2.1 Background Information 8

2.1.1 Web Service Availability 8

2.1.2 Web Service Replication 9

2.1.3 Web Service Communities 9

2.1.4 Agents and Multi agent Systems 10

2.1.4.1 Agent Definition 11

2.1.4.2 Agent Environment 13

2.1.4.3 Agent Classification 14

2.1.4.4 Agent Design and Architecture 15

2.1.4.5 Multi–Agent Systems 18

2.1.4.6 Applications of Agents and MAS 18

2.2 Related Works 19

3 Modelling **23**

3.1 Conventional Web Service Model 24

3.2 Revised Web Service Architecture 25

3.3 Development of the Model Using MAS 28

3.4 Static and Dynamic Attributes 32

3.5 Modelling of Processing Time 33

3.6 Initializer Agent 34

3.7 Tester Agent 35

3.8 Client Agent 36

3.9 Web Server Agent 37

3.10 Web Service 37

3.11 Proxy Agent 39

3.12 Router Agent 40

3.13 Link Agent 40

3.14 Message Protocol Format 41

4 Results and Evaluation **46**

4.1 Configuration and Initialization Phase 46

4.2 Testing Phase 49

4.3 Random Selection 50

4.4 Best Last Selection 51

4.5 Best Median Selection 53

4.6 Parallel Selection 56

4.7 Simulator Evaluation 58

4.8 Effect of More than One Client 66

5 Conclusion and Future Work **69**

5.1 Conclusion 69

5.2 Future Work 72

Appendices **75**

Bibliography **102**

List of Tables

2.1 Properties of Agents 14

3.1 Components Categorization in the Simulation Model 29

3.2 Components in Simulator Application 30

3.3 Client Agent Attributes 36

3.4 Web Server Agent Attributes 38

3.5 Proxy Agent Attributes 39

3.6 Router Agent Attributes 40

3.7 Link Agent Attributes 41

3.8 Details of segments that are created and consumed by agents 44

4.1 Routing Table of Router Agent Number 11 48

4.2 Route Paths and Costs from Proxy Agent to all Web Server Agents . . . 48

4.3 Choosing Algorithm Tested in the Experiment 49

A.1 Test Scenarios Executed in the Experiment 76

A.2 Meaning of Collected Attributes in each Test Scenario 77

A.3 Random Replica Selection with Dynamic Processing Behavior OFF . . . 78

A.4 Random Selection with Dynamic Processing Behavior ON 79

A.5 Best Last Replica Selection with Dynamic Processing Behavior OFF . . . 80
A.6 Best Replica Selection with Dynamic Processing Behavior ON 81
A.7 Best Last Replica Selection with Dynamic Processing Behavior ON and Different Web Server Processing Speeds 82
A.8 Best Last Replica Selection with Dynamic Processing Behavior ON and Router Response Time Changes - Response Time of R4, R7, and R8 increased from 2 to 10 83
A.9 Best Last Replica Selection with Dynamic Processing Behavior ON and a Spike of 20ms in WS1 - Spike happens at Request# 20 in Test 1, Request# 40 in Test 2, and Request# 100 in Test 3 84
A.10 Best Median Replica Selection with Dynamic Processing Behavior ON value of k equal to 1, 3, 5, 7, and 9 85
A.11 Parallel Selection with Dynamic Processing Behavior OFF 86
A.12 Parallel Selection with Dynamic Processing Behavior ON 87
A.13 Random Selection with Two Clients and Dynamic Processing Behavior ON 88
A.14 Best Last Selection with Two Clients and Dynamic Processing Behavior ON 89
A.15 Best Median Selection with Two Clients, Dynamic Processing Behavior ON, and k=3 90
A.16 Parallel Selection with Two Clients and Dynamic Processing Behavior ON 91

B.1 Simulator Configuration Setup - Table 1 of 5 93
B.2 Simulator Configuration Setup - Table 2 of 5 94
B.3 Simulator Configuration Setup - Table 3 of 5 95
B.4 Simulator Configuration Setup - Table 4 of 5 96
B.5 Simulator Configuration Setup - Table 5 of 5 97

List of Figures

2.1 Simple Agent Functional Diagram 13

2.2 Web Service Replication Classification 19

3.1 Web Service Conventional Architecture 24

3.2 Revised Web Service Model 26

3.3 General Distributed System 27

3.4 Sample Simulator Network Topology: P =Proxy Agent, C=Client Agent, R=Router Agent, and WS=Web Server Agent 31

3.5 Simulator Main Class Diagram 32

3.6 Initialization phase started by operator and executed by Initializer Agent 35

3.7 Class relationship between Web Server Agent and Web Service 38

3.8 Simulator Message Protocol 43

3.9 A sample web service request message travelling from client agent to web server agent 45

4.1 Test Sequence Diagram 50

4.2 Random Replica Selection with Dynamic Processing Behavior Turned Off 51

4.3 Random Replica Selection with Dynamic Processing Behavior Turned On 51

4.4 Best Last Replica Selection with Dynamic Processing Behavior Turned Off 52

4.5 Best Last Replica Selection With Dynamic Processing Behavior Turned On 53
4.6 Best Median Replica Selection With Dynamic Processing Behavior Turned off for values of k equal to 1, 5, and 9 54
4.7 Best Median Replica Selection With Dynamic Processing Behavior Turned ON for values of k equal to 1, 3, 5, 7, and 9 54
4.8 Best Median Replica Selection With Dynamic Processing Behavior Turned ON for values of k equal to 1, 3, 5, 7, and 9 and different static processing time 55
4.9 Parallel Replica Selection With Dynamic Processing Behavior Turned Off 56
4.10 Parallel Replica Selection With Dynamic Processing Behavior Turned ON 57
4.11 Parallel Replica Selection With Dynamic Processing Behavior Turned ON for different web server speeds 57
4.12 Best Last Replica Selection With Different Server Response Times and Dynamic Processing Behavior On 60
4.13 Changing the Response time of Routers in Best Last Replica Selection . 61
4.14 Effect of Spike in Best Last Selection Algorithm - Spikes happen in WS1 with a jump of 20 ms in predefined request messages 63
4.15 Comparison of Spike in Best Last and Best Median Selections 64
4.16 Comparison of Single Client Response Times when Dynamic Processing Behavior is Off 65
4.17 Comparison of Single Client Response Times when Dynamic Processing Behavior is On 65
4.18 Random Selection: Two Clients, Dynamic Processing Behavior = Off . . 66
4.19 Random Selection: Two Clients, Dynamic Processing Behavior = On . . 67
4.20 Best Last Selection: Two Clients, Dynamic Processing Behavior = Off . . 67

4.21 Best Last Selection: Two Clients, Dynamic Processing Behavior = On . . 68
4.22 Comparison of Two Client Response Times when Dynamic Processing Behavior is On 68

Acronyms

ACL	Agent Communication Language
ART	Average Response Time
BDI	Belief-Desire-Intention
BRT	Best Response Time
CA	Client Agent
DS	Distributed Systems
DPB	Dynamic Processing Behavior
IA	Initializer Agent
JADE	Java Agent Development Environment
LMRT	Last Median Response Time
LA	Link Agent
LRT	Last Response Time
MAS	Multi-Agent System
NOH	Number of Hits
OSI	Open Systems Interconnections
PA	Proxy Agent
QoS	Quality Of Service
RA	Router Agent

RWS	Replica Web Service
SOA	Service Oriented Architecture
TA	Tester Agent
UDDI	Universal Description Discovery and Integration
URL	Universal Resource Locator
URT	User Response Time
WRT	Worst Response Time
WSA	Web Server Agent
WSDL	Web Service Description Language
WS	Web Service

Chapter 1

Objectives, Motivation, Contributions, and Scope

This work presents a framework for agent-based simulation of Web Services Replica Selection. The proposed architecture provides researchers and programmers with a platform that could be used to improve availability in web services. Availability plays a very important role in improving the Quality Of Service (QoS) in a distributed environment.

There are different ways to improve the availability of a web service. One of the most common methods is *replication*. Replication simply means making copies of an interested entity (also known as *replicas*) and distributing those copies in different locations. The replicated copy could be a piece of data or a computer program like a web service. Another method to improve the availability of web services is through *Web Service Communities*. A web service community is a community of web services (hereafter referred as *generic web services*) that are not carbon copies of each other but provide similar functionalities.

A big challenge in either approach is the method of selection or the *Selection Algorithm* (also referred here as *Choosing Algorithm*). The main outcome of this thesis will be a simulator platform or a simulator application (hereafter referred as *the simulator*), developed based on Multi-Agent System (MAS) modelling and technology that will be used to analyze and compare the behavior of the web service selection methods and algorithms.

1.1 Objectives

The outcome of this thesis is to provide a software platform that will help to analyze, evaluate, compare, and enhance the web service selection methods and algorithms. The platform provides an experimental environment that helps researchers to host and test different web service selection methods to improve the availability of services. The software platform as mentioned previously is a simulation application. It simulates a web service running environment consisting of many software and hardware components and entities, like the web service hosting servers, proxy server(s), client machines, proxy software, interconnecting network, and web services. Conducting this research will open more opportunities to use MAS based simulation to simulate all sort of distributed systems and computer networks.

1.2 Motivation

This thesis uses a simulation platform to evaluate, compare, and examine replica selection algorithms. The platform is not using any of well tested, and widely used simulators. It has been built from the scratch by using agent based software development tools. The rationale of building a simulation platform based on MAS is as follows:

1: **Using real runtime environment is not feasible**: This experiment is about web service replica selection. In real scenario each web service replica is hosted by a separate server, capable of handling web service requests. The servers could be geographically located far from each other in the network. It is easy to find internet service providers that host web sites but it is not that easy to find the ones that provides public web service hosting. A Lot of time and resources are required to develop a web service and have it deployed by a service hosting company. Many steps should be passed including developing a service, finding the right hosting companies, finalizing service contract agreements, and more importantly paying for the required expenses. On the client side several hardware equipments are required to build and host clients and host proxy server(s). The bottom line is that the building of a real runtime environment is time consuming, and is not logistically and financially feasible. On the other hand using a local runtime environment like the campus network also does not provide an experimental platform that resembles the complexity of a real web service distributed system.

2: **Existing network simulators could not be utilized to simulate the nature of problems dealing with distributed systems**: A web service application is considered a complex distributed system [1, p545-552]. A replicated web service environment is an even more complex distributed system. Existing simulation tools in industry and academic environments like *ns–2/ns–3*, *OPNET*, *GloMoSim*, and *NetSim* do not address the complexity of a distributed application like a Web Service (WS) application. A different simulation platform is needed to simulate layers above application layer and software components that constitutes the distributed application. A distributed system could be perceived as a complex application that runs on multiple processing nodes and handles a specific business transaction from clients perspective. Existing simulators are suitable for simulating computer networks, not distributed systems like a WS application where client applications, web services, and web servers run on top of all layers of Open Systems Interconnections (OSI) reference model [2, p48-56].

3: **MAS architectural model makes it very suitable to simulate a Distributed Systems (DS)**: Agents and MAS could be used in many types of applications, especially in environments where the computation is decentralized and distributed. *Distributed Data Mining*, *Distributed Information Retrieval*, *Sensor Networks*, *Social Sciences*, *Artificial Life*, *Computer Games*, and *Soccer Robots* are very few examples of Agent and Malti-Agent System (MAS) applications [3, p3-4]. Multi Agent Systems, because of their distributed based architecture, are ideal platforms to simulate distributed systems. The architecture model of a Multi-Agent System is very similar to architecture model of a distributed system. Distributed systems and applications include many distributed computing nodes and programs that run in a large united group and execute a unified task. Each node in a distributed system is responsible for execution of a portion of this task. Each node does a specific job that is not necessarily the same job of the other computing nodes. Agents in MAS could be easily designed to have the same characteristic and behavior of processing nodes in a distributed system. Each agent has behaviours to help them to achieve their goals. The behaviour of an agent could be implemented in such a way to imitate the behaviour of a processing node in a distributed system or a network. Processing nodes in a distributed system are connected to each other in a network topology that is designed and imposed by its designers. Similarly each agent could be connected and communicate with other agents in MAS environment. Processing nodes in a distributed systems send messages to other nodes and respond to requests coming from other processing nodes. They do these actions based on the logics that have been designed and programmed for them. As an example a web sever knows what sort of messages it should expect from its client and how it should react to them. Similarly agents could be programmed to imitate the same behavior.

Each agent in MAS is executed in its own execution thread; therefore each component or processing node in a distributed environment could be represented and simulated by an agent. As an example an agent that simulates a web server instance is running as a separate thread. The hosting server is another agent running in its own execution thread. The relation of the web server agent to client agents could copy the relation of a web server application to its clients. With the exception of web service itself, every other entity in this simulator platform is an agent, even the components in the interconnecting network. More details about agents and why they are suitable candidates for simulation of a distributed systems will be given in next chapter.

1.3 Contributions

The simulation platform that was designed and implemented for this thesis has the following contributions:

I: As a stand alone application it simulates the behaviour of a web service application including the underlying network components like routers, and communication links. The simulation platform is easy to use and configure. Developers and researchers could design, create, and evaluate new selection algorithms or even combine existing ones to create more complex selection algorithms with less effort, something they could hardly achieve in the real environments.

II: Its architectural model could be reused as the foundation of a more general purpose simulation platform based on MAS to simulate more complex distributed systems.

III: The implemented router and link agents in this simulation platform could be used as a general purpose MAS–based network simulator package.

IV: The developed code in Proxy Agent (PA), the replica selection engine of platform, could be reused in real scenarios. Furthermore the internal behavior of PA, with a slight change, could be used to provide a generic resource selector.

1.4 Scope

Life cycle of web service consists of many stages. First it should be designed, implemented and deployed by its *Web Service Provider*. Then the web service API, its location, and its access information should become available and disclosed to its potential users, a stage called *Web Service Publishing*. The access information in many cases is put in a central location called *Web Service Registry*. A web service then is ready to be used as soon as the clients find the access information about the web service. This is the phase that is called *Web Service Selection*. This experiment is concentrated on web service selection phase. It is presumed that web service replicas are already discovered and their access functions are already known to their clients. The stages of web service publishing, and discovery will not be discussed in this document.

Simulation of network nodes has been done up to certain levels needed to establish a valid network behavior. Router agents simulates the high level functionality of real routers. To avoid too much details in the simulation, the dynamic routing, packet switching, and packet re-sending behaviors were not implemented by Router agents. These complex router behaviors will be left for future releases of the simulation framework. Router agents in this experiment use static routing methods based one preconfigured static routing tables. Router agents encapsulate the high level behavior of *Network* layers and Link agents simulate the behavior of *Link* and *Physical* layers in OSI reference model or Internet Protocol stack(see [2, 51])

1.5 Thesis Outline

Chapter 2 introduces background information and related works done on multi-agent systems and web service selection. Chapter 3 discusses the architectural model and implementation details of the simulation platform. Chapter 4 presents and analyzes the testing results, and finally Chapter 5 draws the conclusions, reiterate the contributions, and discusses about potential future works.

Chapter 2

Background Information and Related Works

This chapter consists of two major sections. The first section is background information that includes a brief explanation of concepts, terms, tools, and frameworks that will be used throughout this thesis. The second section exemplifies the related works that have been done by other fellow researchers.

2.1 Background Information

Before getting to details it is necessary to make ourselves familiar with some terminologies, definitions, and concepts that will be used frequently throughout this document.

2.1.1 Web Service Availability

A system is said to be *available* if it promptly delivers its service. What exactly this means should be clarified in the context of the requirements. For instance, a specification for a web service could clarify that it can reply to the client request within 2 seconds. Availability for a service means it is reachable regardless of hardware, software or user fault. Availability is usually shown as a sequence of nine digits. As an example a notation of 99.999% availability or *five-nine-availability* means the likelihood that not all the requests arriving within one minute get serviced is 0.00001. One method to improve system availability is through *Replication*. In this approach the system continuously provides the service because it has been replicated into similar copies (instances) spread around in different physical locations, so that each one would be able to respond to client requests even though other instances are busy or not working.

2.1.2 Web Service Replication

Replication simply means having multiple copies of a component so that if one fails or is busy the other one could handle the request. Access to a replica should stay similar to access to a single replicated entity and this process, including failover of replicas, should be transparent to the client. *Web Service Replication* means having multiple similar copies of the same web service code that will improve its availability. In this document a replicated web service is also referred as *Replica*.

2.1.3 Web Service Communities

Replication is not the only method that could improve the availability QoS. One other possible solution is through *Web Service Communities*. Web service communities are grouping of functionally similar web services that facilitate and speed up the web service discovery. If a client request fails then the community framework that is controlled through a master web service could delicate the request to another member of community and because that member is a web service with the same functionality then the client should get its desired response. All these should be transparent to a user. Web Service Community could be very beneficial and useful in SOA based architectures for dynamic service selections during a business process flow [4, 5]. In this document a web service communities is also referred as *Generic Web Services*.

2.1.4 Agents and Multi agent Systems

One of the key contributions of this work is introduction of a simulator tool based on agent and Multi Agent Systems (also known as *MAS*). Multi-Agent systems have been a very fast growing field of computer science in last several years. They have been used in different research and industrial applications and have seen by many scholars as a breakthrough in the field of machine intelligence and future of intelligent software development. Agents and MAS could be especially used in environments where the computation is decentralized and distributed. *Distributed Data Mining and Information Retrieval* [6, 7], *Sensor Networks* [8, 9], *Social Sciences*, *Artificial Life*, *Computer Games*, *Simulation*, and *Soccer Robots* are very few examples of Agents and Malti-Agent System (MAS) applications.

In this experiment MAS has been used to simulate a replicated web service application which is considered a distributed system [1, p545-554]. Sections 2.1.4.1 to 2.1.4.6 have been dedicated to definition, classification, and architecture of agents and MAS because of their important and fundamental role in this thesis.

2.1.4.1 Agent Definition

A simple search for the term Agent will result in a plenty of definitions. In computing field, this term is usually combined with other terms like *Agent Systems*, *Multi Agent Systems*, *Intelligent Agent*, and *Autonomous Agents*. In most of the definitions an agent is referred to as an *Autonomous Entity*. It emphasizes that agents actions and behaviour are autonomous. More search in literature shows that there is not a common agreement on the definition of (autonomous) agents.

Pattie Maes of MIT Media Lab, defines agents as computational systems that lives in some complex dynamic environment, and sense and act autonomously in this environment, and by doing so they realize a set of goals or tasks for which they have been designed. This definition recognizes the agents as goal-oriented entities [10].

Michael Coen defines (software) agents as programs that engage in dialogues, negotiations, and coordinate transfer of information . Michael Coen's definition is for so called *software agents*. It could be inferred from his definition that a category of agents is software programs [11]

Wooldridge and Jennings [12] describe an agent as a hardware or (more usually) software-based computer system that is *Autonomous*, *Social*, *Reactive*, and *Pro-Active*. They operate without the direct intervention of humans or others so they are *Autonomous*. They are *Social* in a sense that could interact with other agents (and possibly humans) via some kind of agent-communication-language. They perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in them so they are *Reactive* to their surrounding environment. They however do not simply act in response to their environment; they are goal-oriented and take initiatives so they are *Pro-Active*.

Brustoloni [13] define autonomous agents as systems capable of autonomous, purposeful action in the real world . By his definition an agent could only exist in the real world and so the software agents are not considered as agents.

Franklin and Graesser [14] state that many entities could fall in one of above definitions. Humans and animals, and some robots could be considered as autonomous agents. All of them exist in the real world and are real world agents. Software and *Artificial Life* agents live in computer operating systems, databases, networks, and so on. All of them are situated in some kind of environment, sense their environment, and act autonomously in that environment. They all pursue their own agenda and act continually over some period of time. A software agent runs until it decides not to and an artificial life agent runs until it is getting deceased or dies. Human and animals as well as other real world agents do have the same behaviour. Franklin and Graesser [14] come up with this definition which will be used further on in this document as agent definition:

> *"An autonomous agent is a system situated within, and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."*

Based on the above definition an agent could be depicted as shown in Figure 2.1. An agent perceives its environment through sensors and acts upon the collected information through actuators. As an example a human (agent) has the eyes, ears, nose, tongue, skin as sensors and hands, legs, mouth, and other body parts as actuators. A robot might have optical and infrared cameras as sensors and various types of motors as actuators. A software agent receives keystrokes, data streams like file contents and network packets as inputs and acts on its environment by sending information to screen, or writing to files,

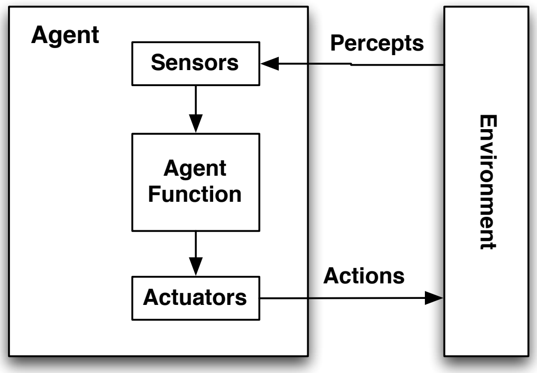

Figure 2.1: Simple Agent Functional Diagram

or sending network packets.

2.1.4.2 Agent Environment

Agents by definition are situated and live in an environment. They could not exist without an environment. In fact, their definition is closely related to the definition of their environment. A robot with visual sensors in an environment that does not have light is a useless creature. This means a system could be or could not be an agent with respect to the associated environment. An agent could not be defined and designed without specifying and defining the environment where it lives and possibly coexists with other agents. Russell and Norvig provide a comprehensive categorization of environment properties [6] as *Fully Observable* vs. *Partially Observable Environments*, *Deterministic* vs. *Non-Deterministic Environments*, *Episodic* vs. *Non-Episodic Environments*, *Static* vs. *Dynamic Environments*, *Discreet* vs. Continuous Environments, and *Single Agent* vs. *Multi-Agent Environments.*

The most complex environments obviously are those that are not observable, non-deterministic, sequential, dynamic, and continuous. Multi agency contributes to all these complexities.

2.1.4.3 Agent Classification

Agents are designed for different purposes and are used in different applications but all have four common properties. Table 2.1 lists these four properties plus several others. Agents could be classified based on these uncommon properties. As an example agents could be classified as *Mobile agents* vs. *Non-mobile agents*, or *Learning agents* vs. *Non-learning agents* and so on.

Table 2.1: Properties of Agents, Adapted from [14]

#	**Property**	**Meaning**
1	Reactive or Sensing/Acting	Responds in a timely fashion to changes in the environment
2	Autonomous	Exercises control over its own actions
3	Goal Oriented or Pro-Active	Does not simply react. It has its own agenda and planning to get to its goal.
4	Temporally Continuous	Is a continuously running process
5	Communicative or Social	Communicates with other agents, perhaps people
6	Learning or Adaptive	Changes its behaviour based on its previous experience
7	Mobile	Able to transport itself from one location (machine) to another location (machine)
8	Character	Believable Personality and emotional state

There are other possible classifications schemas. As an example classification based on the environment where agents exist and live. In this case they could be grouped into three categories of *biological*, *robotic* with physical bodies in real environment, and *software* agents (aka *computational* agents) that exist in computer running environments. Agents could be classified based on the task they perform, for instance, *information gathering* or *email filtering* agents. Another possible classification for (software) agents could be based on their control mechanism like *algorithmic* agents, *rule based* agents, *fuzzy logic* agents, *neural net* agents and etc. The most important classification is the classification of agents based on their internal architecture and design. The following full section has been dedicated to architecture and design of agents.

2.1.4.4 Agent Design and Architecture

A generally accepted grouping classifies agent architecture into three categories of *Reactive Architecture*, *Cognitive Architecture*, and *Hybrid Architecture* [15, p14–16]. A *Reactive* agent does not have any explicit representation of its environment and other agents. It simply maps its input to output based on some internal mapping tables, priority lists, rules, and policies. A *Cognitive* agent reasons from knowledge that represents its environment. It keeps track of state, properties, and dynamics of objects in the environment as well as the other agents who live in that environment. Based on this knowledge it sets up plans to reach its goals, evaluate these plans and goal(s), and if needed to change its plans, and even its goal(s). A cognitive agent does not simply react to its surrounding and that is how it is different from reactive agents. Reactive and cognitive architectures, even though propose two very different solutions are actually complementary. Reactive agents are more responsive and faster and respond to their input with less effort. On the other side cognitive agents think more before acting to make sure their actions directs them to their goal. This is a more intelligent behaviour but requires a lot of processing time so cognitive based agents are slower than their reactive counterpart. There are many situations that just a simple reactive response is sufficient or necessary to keep the agent on the right track. An agent should balance between its behaviours to provide a better, more precise, and more efficient response. Agents that are designed based on this approach are called *Hybrid* agents. They have two internal modules, one module implements reactive behaviour and the other one the cognitive behaviour. The challenge is to maintain a good coordination and balance between these two modules. A more granular architecture classification has been provided by Stuart Russell and Peter Norvig [16, p46-56]. They categorize the architecture of (software) agents into five groups of *Simple Reflex*, *Model Based Reflex*, *Goal Based*, *Utility*, and finally *Learning* Agent that aggregates the behaviour of first four into a more complex architecture. A Simple Reflex Agent is the simplest form of agents that selects its next actions based on current perception. Its decision is not based on the past perception information. Its power is its simplicity and its weakness is its limited intelligence. A Model Based Reflex Agent keeps track of the status of its environment, as well as its perception history to use them in its current decisions. Events change the environment and they could be initiated by non-agent factors or entities or by other agents. As an example in the real world the raining, snowing, or earthquake affects the state of environment. It could also be affected as a result of actions that are committed by humans, animals, plants and any live creatures. The actions committed by agent itself could also affect the environments status. An agent has to keep track of all these events to make a right decision. Agents who are equipped with the capability to follow a goal are known as *Goal Based* or *Goal Oriented Agents*. Goal based agents are more flexible than the previous models but are less efficient in terms of performance and response time. Goals by themselves do not guarantee high quality behaviour in many environments. For instance, an agent, based on a specific perception and internal state may come up with three best actions that all of them direct the agent to its goal, or may face a decision between two or more conflicting behaviours like speed and safety. A utility function that gives a score or numeric value to each selection could help agent to decide. This function could help the agent to make the best decision in many conflicting situations. Agents that use utility functions in their decision process are called Utility Based Agents. Simple Reflex, Model

Based Reflex, Goal Based, and Utility architectures are not independent designs. In fact, each one enhances the drawback of the previous one. All of these designs, even the most sophisticated one, i.e. utility based agents, in their life span, do not evaluate their own behaviour to improve their performance. Performance here means the efficiency of methods that agents use to achieve their goal. The goal oriented or utility agents are examples of intelligent agents who know how to use the inside and outside information (their knowledge) to pursue their goals. Their decision making engine will decide what to do next to get them closer to their goal. Utility based agent is smart enough to choose a better path from all possible happy ending paths. This looks very smart and in many scenarios it works. What these agents do not have in common is their ability to learn from their past mistakes or successes. Therefore they cannot evaluate their quality of actions to find out there could be better ways to do the same action with less amounts of resources, effort, and time. These agents could not be labelled as learning agents. A Learning Agent is an agent who is capable to learn based on its experience, performs better and potentially chooses better goals.

Michael Wooldridge proposes an almost similar categorization for agents architecture [17, p42–66]. He classifies the agent architecture into four groups of *Logic Based*, *Reactive*, *Belief-Desire-Intention*, and *Layered*. Logic Based Agents decide through logical deduction. Reactive Agents are similar to reactive and reflex agents mentioned before. Belief-Desire-Intention (BDI) agents are cognitive-based agents who decide based upon data manipulation of three internal data structures that represent their beliefs, desires, and intentions. BDI architecture has been built based on a simplified version of intelligence in the human being. In this architecture agents have a view on surrounding environment that constitutes their belief. Whatever goal that is set for an agent by itself or its programmer will shape the agent's desire, and agent's plan to achieve its goals forms its intentions. Layered Architecture is the same as hybrid architecture defined in [15] in which the decision is made via various software layers in a layered architecture. Each layer explicitly reasons about the environment at different levels of abstraction. In BDI architecture the main issue is choosing the right strategy of decision making between being a reactive agent or pro-active agent. These two types could actually be combined in one place. The result will be a layered architecture in which an agent is capable of reactive and pro-active behaviour under one shelter with having different behaving subsystems. The subsystems could be arranged in a hierarchical structure where each layer is responsible to take care of one of behaviours.

2.1.4.5 Multi–Agent Systems

In many scenarios an environment could have more than one agent. In fact there are very few applications in which only one agent exists in the environment. Agents are rarely used standalone and they usually coexist and live in the same environment. Software agents that exist in the Internet or soccer playing robots are good examples of environments of more than one agents. The system that consists of more than one agent is called a Multi Agent System or MAS. Unlike single agent environment, the agents in a MAS environment need to communicate, coordinate and resolve their conflicts so they could achieve their goals. All of these new features make the MAS agents much more complex.

2.1.4.6 Applications of Agents and MAS

MAS could be potentially used in many types of applications especially in environments where the computation is decentralized and distributed. Multi Agent Systems, because of their distributed based architecture, are ideal platforms to simulate and develop any large scale distributed systems. Distributed systems and applications include many distributed computing nodes and programs that run in a large group together to execute a unified task. Each node in a distributed system is responsible for execution of a portion of this task. Each node does a specific job which is not necessarily the same job as the other computing nodes. Multi agent system model is a very close model to distributed architecture. Each node could be simulated or realized by an autonomous intelligent agent. Each one of these nodes has its own functionality and configuration information. *Network Simulation*, *Distributed Data Mining*, *Distributed Information Retrieval*, *Sensor Networks*, *Social Sciences*, *Artificial Life*, *Computer Games*, and *Soccer Robots* are very few examples of MAS applications.

2.2 Related Works

The first related work [18] provides a comprehensive classification of web-service- replication architectures. According to [18], there are two different approaches of replication when it comes to web services that are based on the physical locations of hosting machines. These two approaches are *Clustering Replication* and *Geographical Replication.* Geographical Replication could be divided into two subgroups by itself: *Client-Side Replication* and *Server-Side Replication.* Figure 2.2 shows a schematic of this classification.

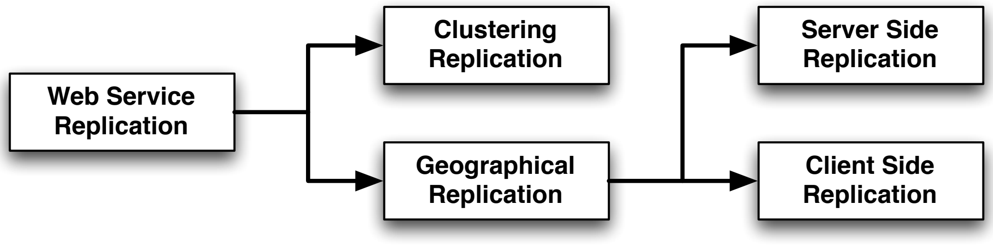

Figure 2.2: Web Service Replication Classification as Proposed by [18]

In cluster replication the replicas will be deployed in different servers or web clusters that are physically located in the same location. In this approach, there is a server that acts as a front end component, known as *Web Switch* or *Dispatcher* which is a proxy that intercepts all incoming requests from clients and dispatches the request to an available replica. In this systems clients only know the Universal Resource Locator (URL) of the dispatcher server. Clustering replication approach aims to maximize web servers throughput, rather than User Response Time (URT). It pays no attention to main network issues like client latency time over the Internet and network dynamics in runtime.

A more general approach, which tackles both issues, is Geographical Replication. In geographical replication, the copies of Replica Web Service (RWS) are distributed among different servers residing in different geographical locations. In reality, in most implementations, there are geographically distributed clusters of web services where each cluster is perceived by clients as one web service as explained in clustering replication. The challenge with this approach is to come up with the right choosing algorithm to connect the best possible replica(s) to the client.

Depending on the physical location where this choosing process is executed two different approaches could be emerged; Server Side Replication versus Client Side Replication. In the former approach the decision making entity is located close to hosting server. This entity could be a program or a hardware equipment or combination of both. A major drawback in this approach is that the client machine does not have any influence on selection of the replica, and the decision is made by servers, routers, DNS servers, or any software or hardware component that has been designed to handle this task. This replica selection mechanism is implemented as a commercial product that is usually very expensive, which is another drawback for server side replication.

In the client side replication, the decision making entity is either located in the client machine or running on a proxy server close to client machine(s). The decision making entity should know the URL of all RWS instances and the main challenge here is to find the best choosing algorithm to pick up the right RWS instance(s). Generally, this method is implemented via software packages that either will deploy in the clients browser (as an applet to download) or as a proxy server that will act as a broker between client and web service replicas. The simulator program will be designed based on the client side geographical replication to simulate and analyze the choosing algorithms.

The papers [19], [20], [21], [22], [23], and [24] include all works that either have implemented a standalone application to optimize the replica selection process or simulation applications that analyze and compare the different choosing algorithms. The work presented in [19] is based on the client side approach mentioned in [18] that analyzes and compares a number of choosing policies in the real environment. This paper presents an interesting assessment of five client-side choosing algorithms to access real web service replicas that are hosted and run by four distant servers around the globe. The assessment results show that how characteristics of local environment of client machines could impact the performance of some choosing algorithms. The simulator application presented in [20] defines a modelling framework by creating a broker proxy between clients and web service replicas. The broker proxies contact with each other and find the best available replica based on a utility function that could be defined based on different attributes like geographical locations of a web service provider. The work done in [21] addresses the issue from a different point of view. The authors show that how a peer to peer network of service registries could be used to locate the most appropriate replica in the network. The work done in [22] evaluates replication possibilities at the operating system level that uses features in the modern operating systems like windows 2003 and IIS servers to replicate web service codes. Its approach could be categorized as the server side, non-geographically based approach classified in [18].

A very similar work to our research is [23], a simulation platform that models web service selection. This is the closest work to the simulation framework proposed in this thesis. It is based on multi agent technology that provides an interesting approach in simulation of distributed environment that hosts web services. Works presented in [5] and [4] is about web service communities. These two papers explain the idea of web service communities and how they could be utilized to improve the availability of web services not through replication, but through creation of a society of functionally similar web services, or composition of web services, so that any of them could provide the desired service to the client(s).

The work presented here extends and enhances the simulation platform presented in [24] that is not based on MAS and uses utility functions to simulate the interconnecting networks between proxy and web servers. It combines several features from the mentioned works. It adopts the client-side service selection model presented in [18], uses a proxy server to choose the desired replica like [20], and is a MAS–based simulator similar to [23]; however it extends many of these works in different ways. It simulates the underlying network of the distributed systems including the connecting routers and links. This part could be extended and used as a standalone package to simulate any computer network and so its usage is not restricted only to web service related applications. It abstracts the high level components in a distributed systems as separate agents so it could also be used as a simulation tool to simulate more generic distributed systems. It could be configured to have more than one clients and could be deployed in a distributed running environment. Each agent or group of agents could be deployed in different machines so it could be easily scaled up to incorporate many more agents.

Chapter 3

Modelling

The simulation platform (or the *simulator* hereafter) consists of many agents, each simulates a host processing node or a communication link in web service distributed system and its underlying network. The simulated distributed system is a replicated web service application where each replica is hosted and executed by a separate web server. In real scenario a web service is deployed in a hosting server machine, which is capable of processing web service requests and delegating the requests to the appropriate web service. A web-service-call process constitutes of a request message sent from the client to host server and a response message coming back from the host server to the client. Web service client(s) could be geographically very far from the hosting server and so the messages could pass through many intermediate processing nodes in the network including intermediate servers and routers.

Simulating the behaviour of a web service requires understanding the conventional *Web Services Architecture* [1, p551-552]. This architecture will be reviewed in section 3.1 particularly from web service availability perspective. The simulation model, the scope of experiment, and how the model fits the conventional model will be explained later in the sections after that.

3.1 Conventional Web Service Model

In conventional web service architecture there are three key components as shown in Figure 3.1 below. These components are Web Service *Consumer*, Web Service *Provider*, and Web Service *Registry* or *Directory Service* also known as Universal Description Discovery and Integration (UDDI) server. A web service should be first *published* in directory server by Web Service Provider, the author of web service. To execute web service operations the consumer needs to find the service, a process known as *Web Service Discovery.* Web Service Discovery is realized by searching web service key information in UDDI server. One piece of information that UDDI server provides is Web Service Description Language (WSDL) file. This file contains web service binding information like its URL and signature of its operation(s). Consumer could then initiate a web service call. To do so the consumer compiles the request in a *SOAP* message. A SOAP message is usually sent on top of another protocol to host server. In majority of cases the hosting server is a *Web Server* and so the SOAP message is wrapped within a HTTP message.

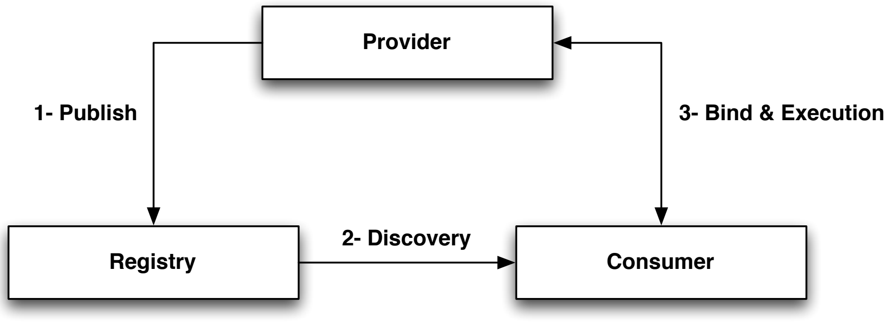

Figure 3.1: Web Service Conventional Architecture

In the conventional model the consumer could be a human user initiating the request from a web browser but in majority of scenarios it is a client program running from any locations in the network. One of key features that contributes to QoS is its *Availability*. Availability of a web service could be negatively affected by many factors like hosting server failure, network traffic, or numerous requests coming from different consumers. A low availability affects the service response time and hence it QoS. A general and very common method to address a service availability and its response time is through *Service Replication*. Web service replication and selection process could be realized by many approaches that were briefly explained in first two chapters of this document.

3.2 Revised Web Service Architecture

Figure 3.2 shows a revised web service architecture by introducing a new component that handles WS discovery, binding and execution on behalf of consumer. This new component, shown as *Proxy Server*, communicates with directory server to query web services and executes the web service calls on behalf of its consumers. Consumers will send the service requests to Proxy Server instead of sending them directly to the service. Proxy server is responsible to dispatch the requests to service provider. In case of replication the proxy server is responsible to choose and forward the request to the right replica based on some replica selection algorithms. Consumers only know the address of the proxy server and not the services. When the message comes into proxy server it routes the message to appropriate hosting server and in this process it substitutes the message source address (consumer address) with its own address and substitutes the destination address (proxy address) with the address of hosting server. The message is then sent to the host. The host only knows the sender address (proxy server address) and it has no knowledge of the consumer address. It is the responsibility of proxy server to make sure the consumer message gets to the appropriate destination and the response are getting back to the same consumer. The network between a consumer, a proxy server, and hosting servers consists of intermediate servers, routers, and communication links. In a complex DS like the Internet there could be many of these intermediate processing nodes and links, which are responsible to carry and froward the message from sender host to destination host.

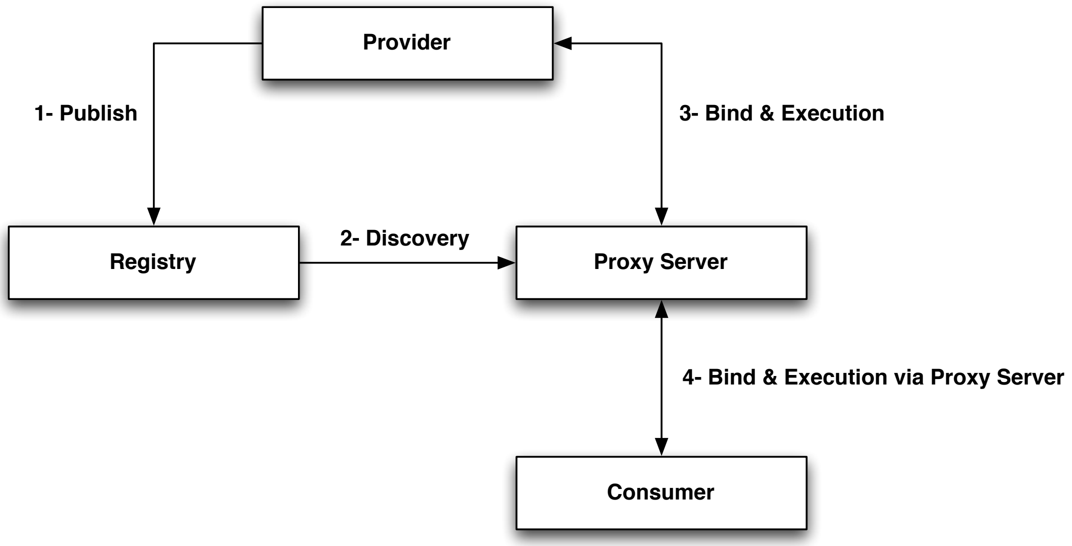

Figure 3.2: Revised Web Service Model

One clear benefit of proxy server is that it could intercept and control the service discovery and selection process. It can discover the *replicated services* (instances of the same web service) or *generic services* (services with similar functionality but different APIs). Proxy server also could cache in the information of replicas and keep track of their status during web service calls or between calls. A Proxy server could decide which replicas or generic web services are qualified enough to be considered as eligible services and which one are not. Proxy server could increase the QoS in many different ways.

The solution presented in this thesis only addresses one of the benefits of proxy server: *Replica Selection*. Web service *publishing*, and web service *discovery* and other potential benefits of having a proxy server will not be discussed here. Based on this presumption our model will be used to simulate the web service applications like Figure 3.3

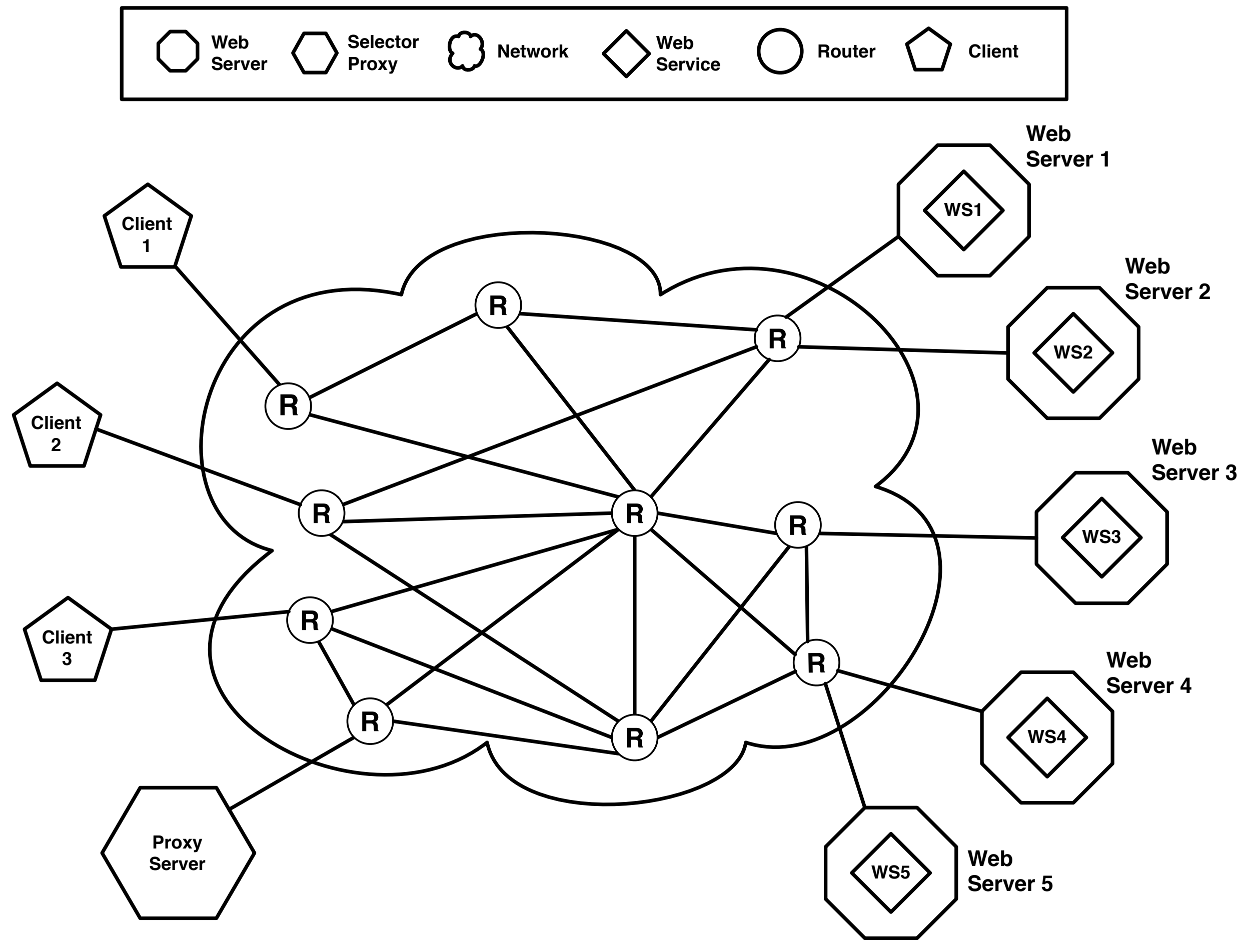

Figure 3.3: General Distributed System

As it is shown in Figure 3.3, the distributed network consists of a series of replicated web services, each hosted by a separate web server plus a series of client nodes and a server that hosts the proxy server. All these nodes are connected to the Internet shown as a cloudy network. The simulation platform has been designed based on *Client Side Geographical Selection* and so it is assumed that the physical location of proxy server is within the vicinity of client nodes.

3.3 Development of the Model Using MAS

The proposed model has been implemented using one of mostly used MAS development tools called Java Agent Development Environment (JADE) [25, 26]. JADE is an agent based development and runtime environment based on Java programming language.

Any distributed system executes on top of a computer network [1, p116-125]. To simulate a distributed system the underlying network should also be simulated. In this experiment agents simulate both key network components as well as more complex components running on top of the *OSI reference model* or *Internet protocol stack* [2, p48-56] like web servers, proxy servers, and web service applications. A web service application consists of major components like web servers, web services, client applications, processing nodes, and communication links that reside between client and server. In reality the network between *client hosts* and *server hosts* consists of routers, switches, bridges, and communication links [2, p11-12]. All processing nodes between source and destination hosts only process the messages (or packets) up to level three of network. The intermediate processing nodes are only responsible to process messages up to the level that guarantees the delivery the message or packets to their final destination host node [2, p315-316].

A computer network could be abstracted by a graph in which each *node* is a processing unit and each *edge* is a communication link. A processing unit could be a host machine as the original source or final destination of the message, or could be an intermediate machine that routes the message to its final destination. Communication links are data pipelines that pass along the message from one node to another. Examples for processing nodes could be desktop computers, network printers, routers, bridges, and server machines like web servers in the Internet. Examples for communication links are Ethernet network cards and cables [2, p53] . In the simulation program these components could be categorized in three main groups shown in Table 3.1.

Table 3.1: Components Categorization in the Simulation Model

Name	**Description**
Host	Original sender or final receiver of messages. A host node connects to the network through a link. In this experiment a message contains the web service request or web service response payloads. Clients, proxy server, and web servers are considered as host node
Router	Intermediate processing nodes that are responsible to route the messages to their final destination. Router nodes connect to the network by at least two links. In this experiment the router Nodes simulate a portion of functionalities defined in layers three of OSI reference model or Internet stack [2, p51]
Link	Simulates the communication lines and their behaviour which is part of the responsibilities of layer one and two of OSI reference model or Internet stack [2, p51] . A link connects two nodes to each other. The nodes could be host or router nodes.

Each main component in a real web service application is modelled and simulated by an agent and based on above categorization. There are agents that simulate client, proxy server, web server, router, and link agents. There are also a couple of administrative agents in this model that are responsible to administrate and configure the platform. In this model the agents could be categorized as *Administrative* Agents or *Runtime* Agents. Administrative agents are the agents that communicate with system administrator (a person) who operates the system and runs the test scenarios. Administrative agents do not participate in any simulation process and only accepts messages from the administrator. Runtime agents are the agents that simulate the behaviour and functionality of a real component in the network. Runtime agents do the real simulation task and only communicate with each other. Administrator also could directly communicate with runtime agent but only to read their internal status for debugging purposes. This categorization and type of each participating agent has been recapped in the Table 3.2.

Table 3.2: Components in Simulator Application

Agent Name	**Role**	**Description**
Initializer	Administrative	Initializes and creates the runtime environment including all agents
Tester	Administrative	Accepts testing commands from administrator and conduct the test scenarios
Client	Runtime	A host node that simulates the behavior of a Client
Proxy	Runtime	A host node that handles the replica selection process
Web Server	Runtime	A host node that simulates the behavior of a web server that hosts a web service
Router	Runtime	A router node that simulates the behavior of routers and intermediate processing nodes
Link	Runtime	Simulates the behavior of a communication line. A Link Agent is connected to either a router node or a host node

Figure 3.4 shows a sample of a modelled distributed system that can be simulated by the simulator. Any link and node in this figure is an agent. In this example there are 24 node based agents, out of which five nodes are Web Server agents, 17 are Router agents, one is Proxy agent, and two are Client agents. The remaining agents are Link agents. In total there are 64 runtime agents in this network. Initializer and Tester agents are not shown in this picture. The configuration and topology of network will be defined in a property file that is uploaded during system startup and is used by Initializer agent to initialize and create the whole runtime environment.

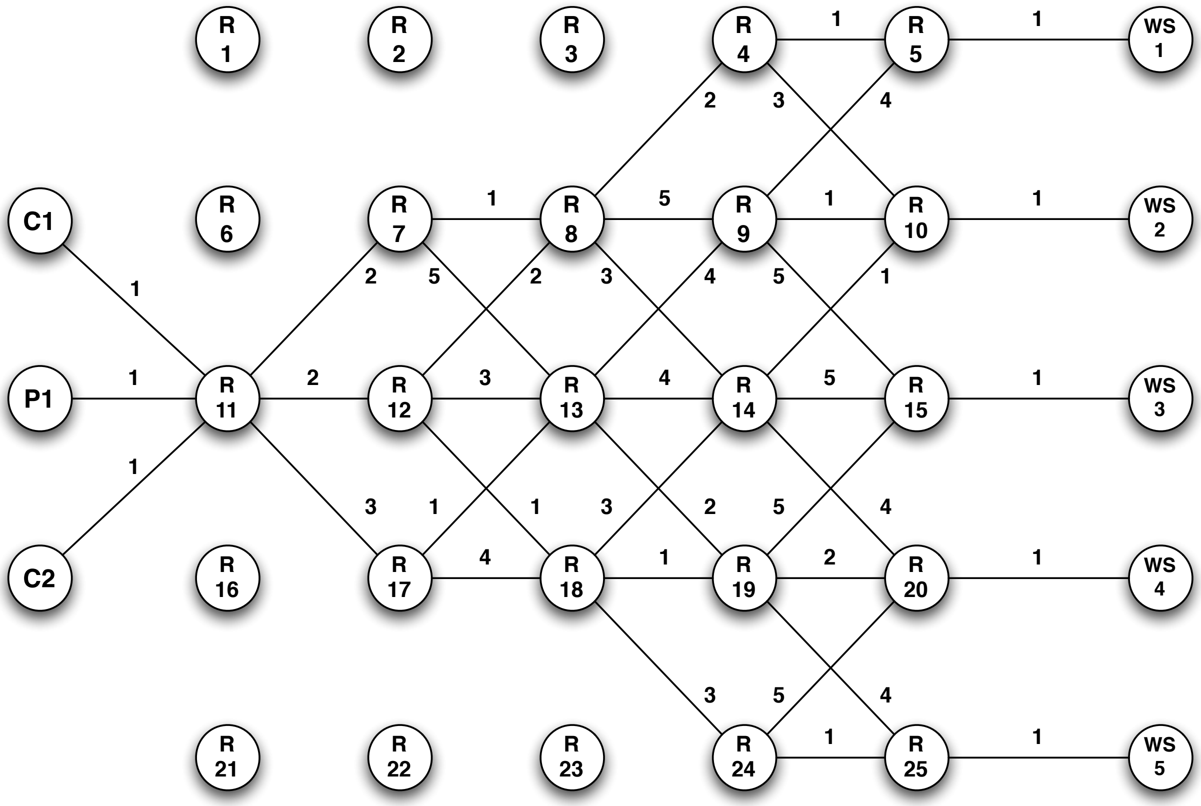

Figure 3.4: Sample Simulator Network Topology: P =Proxy Agent, C=Client Agent, R=Router Agent, and WS=Web Server Agent

Each agents has its own internal static and dynamic attributes that will be used by the agent during runtime.These attributes will be initialized by Initializer agent in startup and later on will be used and manipulated by that agent. Figure 3.5 shows the UML class diagram of all agents and their associated classes.

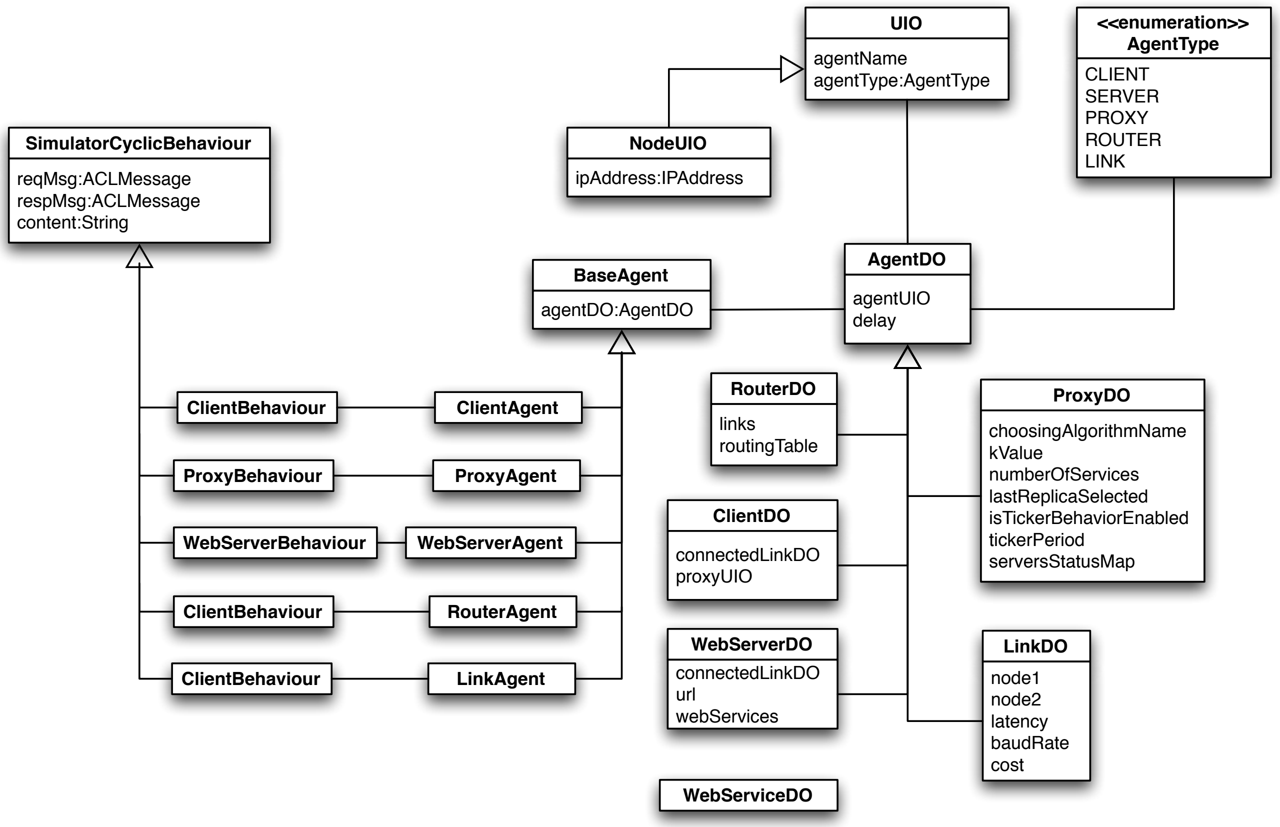

Figure 3.5: Simulator Main Class Diagram

3.4 Static and Dynamic Attributes

A real distributed system consists of many processing nodes and communication links. Each processing node could be made up of a number of software and hardware components. The status of each processing node or communication link could be represented by a series of attributes that could be categorized into two groups: *Static Attributes* and *Dynamic attributes*. The value of static attributes does not change by time but the value of dynamic attributes does.

The status of a web server, for instance, could be represented by many static and dynamic attributes. IP Address, URL, CPU clock speed, number of CPUs, and machine memory size are all examples of web server static attributes. Number of incoming messages, and CPU temperature are examples of web server dynamic attributes. A twisted pair cable, as another example, could be represented by latency and baud rate attributes that do not change by time. Each agent in this simulation platform has been modelled like its real counterpart and has a series of static and dynamic attributes. For instance each Web Server Agent has its own IPAddress, and URL, or each Link agent has its own latency, and baud rate values.

3.5 Modelling of Processing Time

In the simulation process, the real time could not be used for calculation of response times because the real processing time tightly depends on the running environment where the simulator program is being executed. A computer with higher CPU speed executes the simulation program faster than a computer with lower CPU speed, therefore time has to be simulated.

Processing time of each entity is a function of its relevant static and dynamic attributes. Relevant attributes are attributes that could affect the response time of an entity and could be static or dynamic attributes. For instance the CPU speed of web server machine (a relevant static attribute), or number of waiting messages in web server incoming queue (a relevant dynamic attribute) could affect its overall response time; however a web service IP Address (an irrelevant static attribute) does not affect its response time.

To simplify the time simulation, the timing behavior of an entity can be represented by two special purpose attributes. One as a static attribute and one as a dynamic one. The static attribute is configurable and initialized in system startup and its value is not changed by time. The dynamic attribute is calculated during runtime by each agent or class. Total value of delay or processing time of an entity will be the addition of these two values, therefore the processing time of any entity could be simplified and presented by the following formula:

$$TP = SP + DP \tag{3.1}$$

in which TP is total processing time, SP is static processing time and DP is dynamic processing time. The effect of dynamic processing time could be turned on or turned off before each test. The unit of time is conventional and for simplicity is shown as *ms* in the the reports.

3.6 Initializer Agent

Initializer Agent (IA) is responsible to create and initialize the whole running environment including Links, Routers, Web Server, Proxy, Client, and Tester agents. IA is an administrative agent and is not involved in the simulation process. It is the first and only agent that is started in the platform and accepts only one command from operator (administrator) to initialize the simulation platform. Initializer agent reads all the startup and configuration information from a property file including agents specific attributes, network topology, routing table information, and the type of choosing algorithm (See appendix B to see these configuration properties). Figure 3.6 shows the detailed UML sequence diagram of Initializer agent as soon as it receives the start command from the operator. During initialization phase the Initializer Agent delegates the creation of each agent type to a separate handler class. For instance, the web server agents and their associated web services will be created and initialized by *Server Agents Handler*.

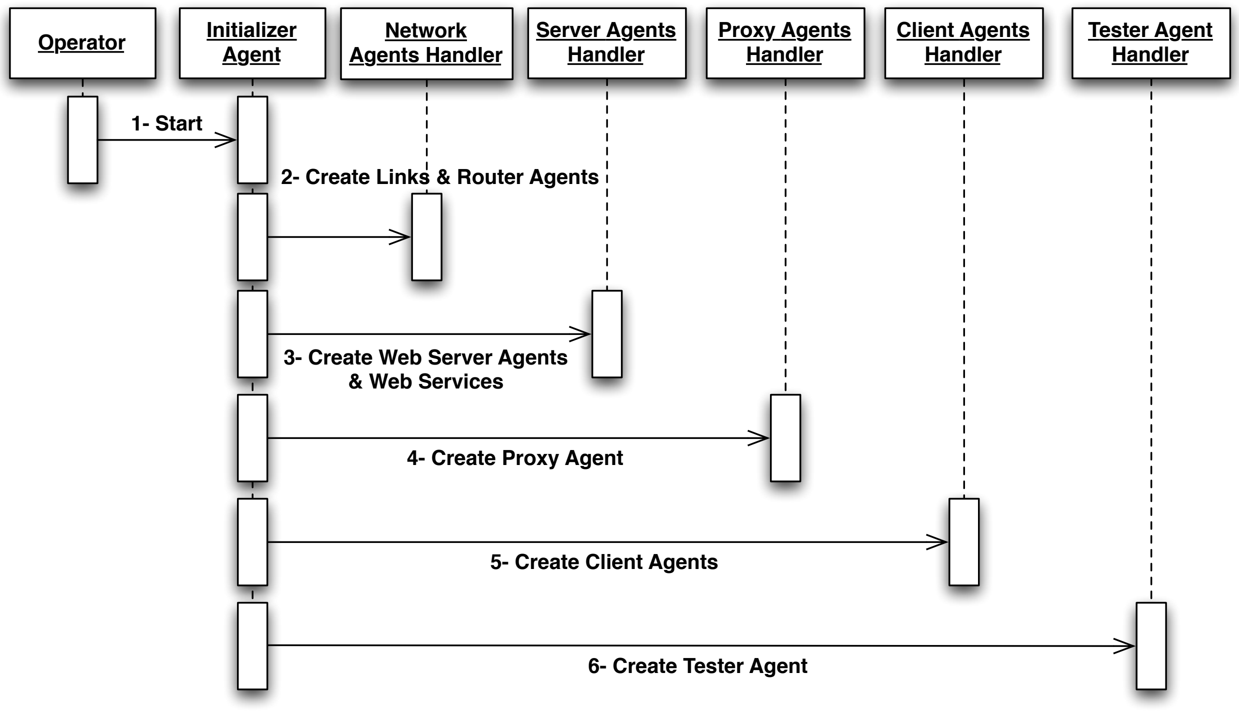

Figure 3.6: Initialization phase started by operator and executed by Initializer Agent

3.7 Tester Agent

Tester Agent (TA) is an administrative agent responsible to execute the test case scenarios. Administrator sends testing command to tester agent and tester agent sends appropriate commands to Client Agent(s) on behalf of administrator to initiate web service calls. Tester Agent accepts two parameters to initiate test case scenarios: *number of clients* and *number of requests*. The first one determines the number of client agents that should participate in the test and the latter one determined number of web service calls that must be sent out by each client agent.

3.8 Client Agent

Client Agent (CA) simulates the behaviour of a web service client. In reality a web service is identified by its associated WSDL which is referenced in a form of URL. A real web service request is wrapped in a SOAP message and is sent over to server. The SOAP message is usually wrapped within another transporting message like HTTP. These protocols have not been modelled in the experiment. CA uses a proprietary message protocol to send out web service calls that includs the requested operation name, and its parameter(s). This protocol will be explained in section 3.14.

Clients in this model do not know the web server address. Delegating the request to the right web server, or better to say the right Web Server Agent (WSA), is the responsibility of PA. CA only knows the address of PA and so sends the request messages to PA. CA is a host node and therefore is connected through a Link Agent (LA) to the next processing node that is a Router Agent (RA). A Client Agent keeps track of the following attributes shown in Table 3.3.

Table 3.3: Client Agent Attributes

Attribute Name	**Description**
Agent Name	Agent Unique Name Starts with 'C' and a number like 'C1'
Agent Type	An enumeration that specifies the agent type. CLIENT for this agent
IP Address	Client agent is a node agent so it should have an IP Address
Connected Link	Binding information to connected Link (Link Agent)
Proxy Agent	Binding information of receiver host (Proxy Agent)

3.9 Web Server Agent

Web Server Agent (WSA) simulates the behavior of a web server. It hosts and deploys a web service. In real scenario a web service is deployed in a server (usually web server) with a *Web Service Engine* that can parse and handle a SOAP message. A web service implementation is nothing but a simple program with one or more public operations. The signature of the operation is defined in WSDL file. A simple and straightforward model of a web service could be a class that implements the web service public operations. When a hosting server (like a web server) receives a web service request it extracts the SOAP payload and passes it to the SOAP engine. The SOAP engine parses the payload to extract the operation name as well as the argument values and call the appropriate operation of the object or program that implements the web service. WSA does the same process. WSA could deploy one or more web services in the model but in this experiment only one web service is deployed per each web server. WSA is a host node that is connected to a Link Agent (LA). It keeps track the attributes shown in Table 3.5.

3.10 Web Service

A Web Service (WS) is not an agent in this model and is a simple Java class. The detailed class model of web service and its relation to its hosting web server agent is shown in the UML class diagram depicted in Figure 3.6. LengthConverterWS is the name of a sample web service that has been designed and implemented to conduct the experiments. This web service has two operations that handle simple length conversions.

Table 3.4: Web Server Agent Attributes

Attribute Name	**Description**
Agent Name	Agent Unique Name Starts with 'W' and a number like 'W3'
Agent Type	An enumeration that specifies the agent type. SERVER for this agent
IP Address	WSA is a node agent so it should have an IP Address
URL	Web Server URL, a value given at startup
Connected Link	Binding information to connected Link (LA)
Session Status List	A simulation specific internal object that keeps track of start and end time of each processed request to calculate server response time
Deployed Web Services	A internal map that has the information of all deployed web services. This map is used during request execution to find the information needed to call a web service
Request Message Counter	A simulation specific object that calculates the number of processed requests for reporting purposes

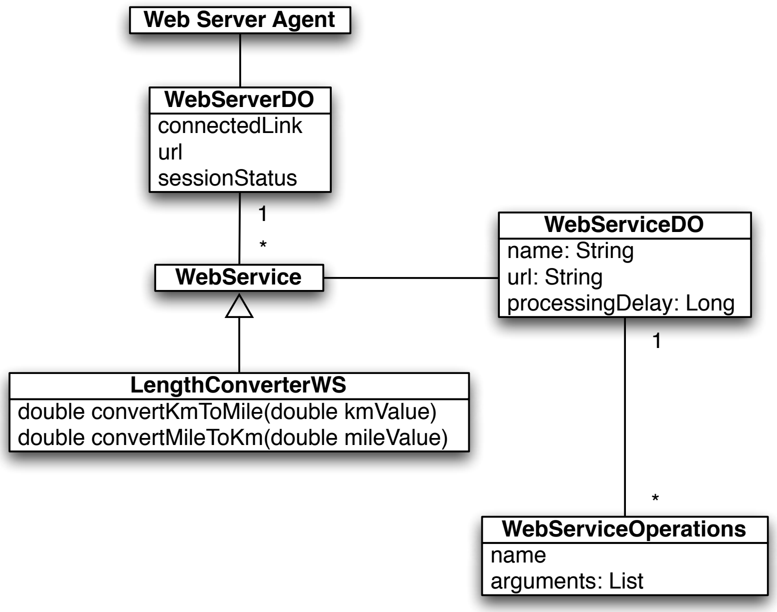

Figure 3.7: Class relationship between Web Server Agent and Web Service

3.11 Proxy Agent

Proxy Agent (PA) is the most complex agent of the simulation platform. It core business logic could be reused in real scenarios without any modification. PA is responsible to receive web service requests from client agents and forward them to the appropriate replica based on the choosing algorithm method selected in startup. A proxy agent has to collect and keep track of many dynamic and static information during system runtime by which it will be able to find the best replica. PA functionality could be extended to transform the operation requests to a new form so both replicas and generic services could be used in the distributed environment. This capability is not currently implemented in proxy agent and would be an extension to this work in future. Proxy Agent is a host node and therefore is connected only to a link agent. It keeps track of the attributes shown in Table 3.5.

Table 3.5: Proxy Agent Attributes

Attribute Name	**Description**
Agent Name	Agent Unique Name Starts with 'P' and a number like 'P1'
Agent Type	An enumeration that specifies the agent type. PROXY for this agent
IP Address	Proxy Agent is a node agent so it should have an IP Address
Connected Link	Binding information to connected Link (Link Agent)
Choosing Algorithm	Choosing Algorithms that should be used to select the replica
K Value	Size of history window of past requests
Number of Servers	Number of servers (replicas) that this proxy knows about
Last Selected Replica	The status information of the replica selected in previous request
Server Status List	An object that keeps track the status of previous requests for each server

3.12 Router Agent

Router Agent (RA) simulates the behaviour of a router. It is connected to at least *two* link agents: incoming link agent and outgoing link agent. RA receives a message from incoming link agent and after calculating the destination node it delivers the message to outgoing link agent. RA does the routing logic by consulting a routing table that is initialized and configured for each router in system startup. Each RA keeps track of the attributes shown in Table 3.6.

Table 3.6: Router Agent Attributes

Attribute Name	**Description**
Agent Name	Agent Unique Name Starts with 'R' and a number like 'R4'
Agent Type	An enumeration that specifies the agent type. ROUTER for this agent
IP Address	Proxy Agent is a node agent so it should have an IP Address
Connected Links	List of Link Agents Connected to This Router
Routing Table	An object containing the routing information for all possible destination hosts

3.13 Link Agent

A Link Agent (LA) represents and behaves like a communicatio link. It actually simulates layer one and two of the network in OSI reference model [2, p53]. Each LA connects two nodes in a network. Each node could be a host or a router node. A node based agent could be a Client, Proxy, a Web Server, or a Router Agent. A Link Agent must know the address of its source and destination agents. A source agent is the agent that passes the message to the link agent and the destination agent is the agent that the message is delivered to. LA is not a node based agent and so does not have an IP Address. It encapsulates the attributes shown in Table 3.7.

Table 3.7: Link Agent Attributes

Attribute Name	**Description**
Agent Name	Agent Unique Name Starts with 'L', and names of agents that this link agent connects them to each other separated by '_' like 'L_R5_R9'
Agent Type	An enumeration that specifies the agent type. PROXY for this agent
Connected Node 1	Connection information of one of ending nodes
Connected Node 2	Connection information of the other ending node
Latency	Latency value of the connection link that is initialized during startup
Baud Rate	Baud rate throughput of connection link that is initialized during startup
Cost	Cost value of this connection link that is calculated based on Latency and Baud Rate and is used by routers to calculate the next route

3.14 Message Protocol Format

In the simulation model, the client, proxy, and web server agents simulate the behavior of components that run on top of all seven layers in OSI reference model. Router Agent simulates the behavior of layer 3 of OSI reference model [2, p51-52], and Link Agent simulates the behavior of layer one (physical layer) and layer two (link layer) of OSI reference model [2, p53]. An agent, in order to talk to other agents, should send a message to their input queue. Transferring of message from a source agent to destination agent(s) is handled by JADE framework. Each message must comply with a well defined communication protocol otherwise the receiver agent could not process and handle the message. JADE platform supports Agent Communication Language (ACL) standard which means the agents running in this environment communicate based on ACL protocol [26, p13-14, p65-66]. The data content of an ACL message is different per application. It is the responsibility of the application designers to define their own communication protocol and make sure that all agents comply with that protocol. The message should be wrapped inside an ACL message when agents communicate with each other. This MAS–based simulation platform is no exception and has its own communication protocol. The format of this protocol and the details of each data segment are illustrated and explained in Figure 3.8 and Table 3.8.

In Each web service call the client agent fabricates a web service request message and wraps it in an ACL message and then sends it to the next destination agent, which is a link agent. Link agent's job is to deliver the message to next processing node. Link agent first calculates and updates the processing time segment (segment number 7 shown as PT in Figure 3.8) based on the value of its baud rate and latency attributes and then forward the message to its receiving processing node (a router agent). Router agent is responsible to first calculate the processing time field and then deliver the message to the next agent. Router agent will extract the the receiving host node address from the message to calculate the address of the next agent (a link agent). Router agent updates the destination processing node address segment in the payload and forward it to the right link agent. This process continues until the message gets to the proxy agent. Proxy agent is responsible the update receiving host address and selects the replica web service that should receive the message. It will then update the destination host node address fields and redirect the message to the right link agent. This sequence of actions continues until the message is delivered to it finals destination web server agent. Figure 3.9 shows a sample web service request message that is sent from Client Agent 1 to Web Server Agent 1. In this figure where LM and HM indicate link and host messages receptively, the request message has to travel through eight link agents (*L–C1–R11*, *L–R11–P1*, *L-P1–R11*, *L–R11-R7*, *L–R7–R8*, *L–R8–R4*, *L–R4–R5*, and *L–R5–WS1*), and seven node based agents (*R11*, *P1*, *R11*, *R7*, *R8*, *R4*, and *R5*) before it gets to its final destination (*WS1*). This sample scenario is based on the network that was configured and tested in this experiment (see Figure 3.4).

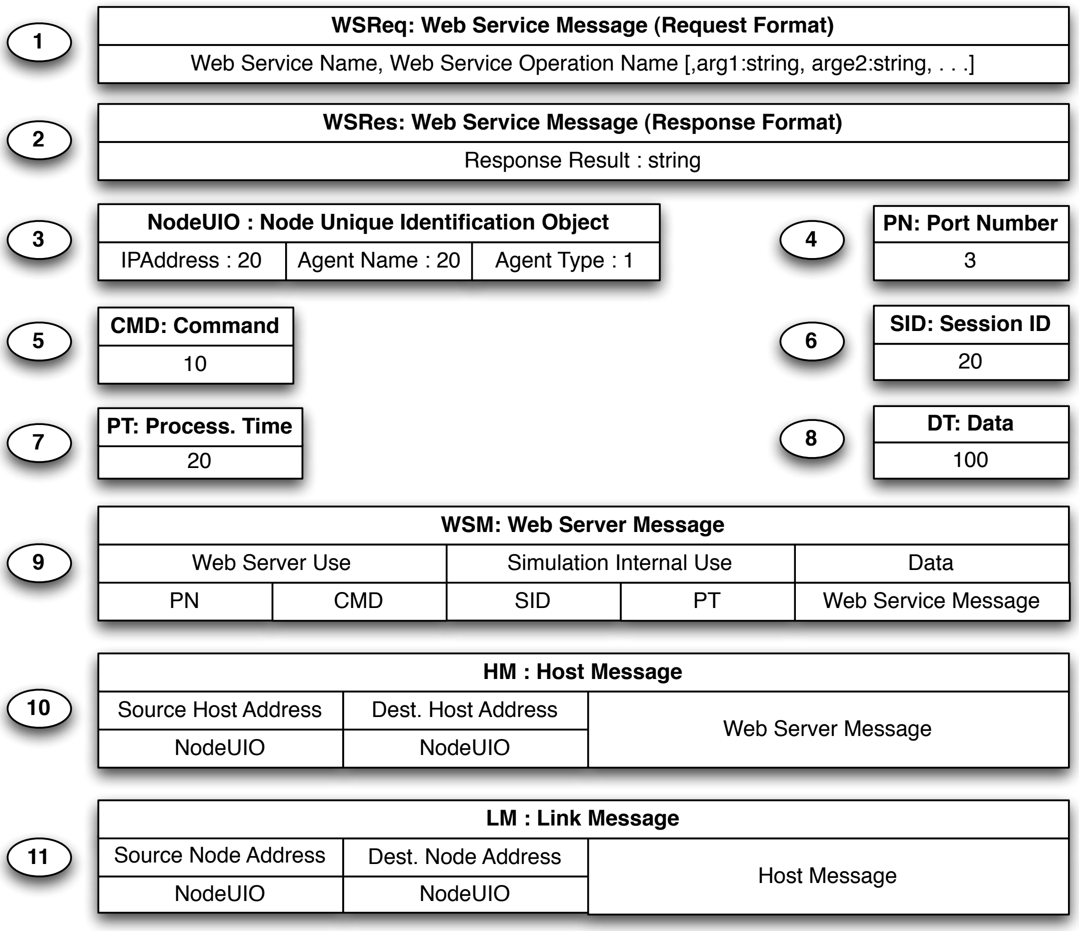

Figure 3.8: Simulator Message Protocol

Table 3.8: Details of segments that are created and consumed by agents

#	**Seg. Name**	**Creator Agent**	**Consumer Agent**	**Description**
1	WSReq	CA	WSA	Contains a Web Service Request
2	WSRes	WSA	CA	Contains response result of a web service call
3	NodeUIO	CA PA WSA RA	CA PA WSA RA	Contains IP Address, agent name, and agent type of the source or destination node agent
4	PN	CA	WSA	Port number of destination web server
5	CMD	CA	WSA	A web server used this filed to execute the right command
6	SID	CA	CA PA WSA	Session ID that uniquely identifies a transaction and used to correlate and keep track of each web service call
7	PT	CA	CA PA WSA RA LA	Simulation internal use to calculate and record the simulated processing time of each agent
8	DT	CA WSA	CA WSA	Data section that contains either a web service request or response
9	WSM	CA WSA	CA WSA	Generated and consumed by main source and destination host nodes (client and web servers)
10	HM	LA	CA PA RA WSA	Acceptable message by all Node–Based–Agents
11	LM	CA PA RA WSA	LA	Acceptable message Link Agent

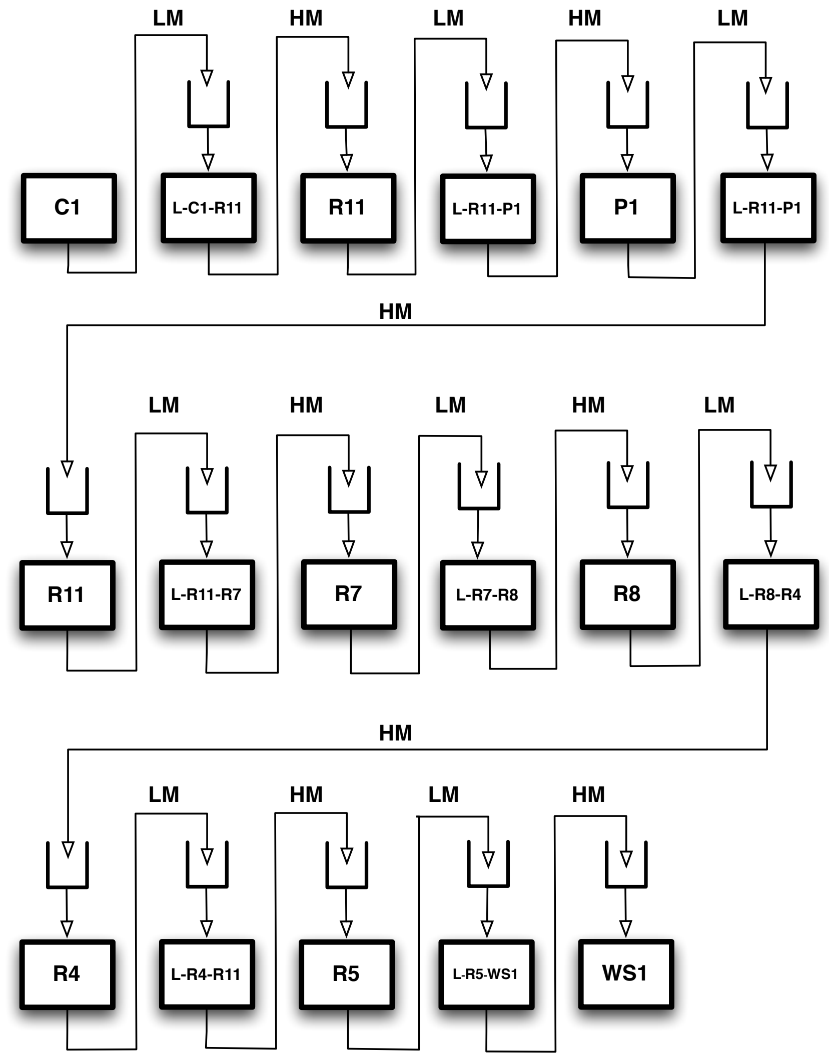

Figure 3.9: A sample web service request message travelling from client agent to web server agent

Chapter 4

Results and Evaluation

In this chapter the results of the experiment will be presented. A single computer system with dual core Intel^R processor and 4GB of RAM has been used to execute the test scenarios. The full specification of the equipment and runtime environment could be found in Appendix C. The first part of this chapter (sections 4.1 and 4.2) explains about the setup and configuration of runtime and testing environment. The second part (sections 4.3 to 4.8) evaluates the simulator behavior and analyzes the results.

4.1 Configuration and Initialization Phase

The simulator platform as explained in chapter 3, is a MAS–based application. The network topology, number of client, web server, proxy, router, and link agents is defined in a configuration file. Agent's specific attributes like IP Addresses, URLs, routing information, and link latency, baud rate, and cost values are also determined in configuration file. The full list of configuration properties could be seen in Appendix B. The configuration file is read by a special purpose agent called Initializer Agent (IA). IA is responsible to create the whole running environment before the experiment begins. This includes creation and initialization of all agents and related classes. IA is the first and the only agent that is created at startup. It will then wait to get the *start* command from the operator (the person who conducts the test) and after doing its job it will no longer be used during the experiment. The initialization phase is done step by step in a predetermined order as shown in UML sequence diagram shown in Figure 3.6.

Figure 3.4 shows the examined distributed web service system and the underlying network topology. This system consists of two client agents (C1 and C2), one proxy agent (P1), five web server agents (WS1 to WS5), five web service instances (not shown in the figure but each is deployed and run within a dedicated web server agent), 25 router agents (R1 to R25), and 39 link agents (shown as connected lines with their cost values). Out of 25 router agents, only 17 of them are connected and actually participate in the experiment. There is no limit for number of client, web server, router, and link agents. Topology of the network could be any complex rectangular graph. The experimenter only determines the number of rows and columns in the graph and the interconnecting links as well as number of clients, and web servers. The rest will be created and initialized by IA. The only limit is the number of proxy agents that could be only one. The number beside each link is its cost value (see [2, p374-375]) that is defined for each link as a separate property in configuration file. These values are used by IA to create the routing tables for each router agent. For instance the routing table for router agent number 11 will be generated like Table 4.1. This routing table is used by Router Agent 11 during run time to pass the incoming messages to the right destination.

In majority of test cases and for simplicity of comparison the web servers have been configured in such a way that web service number one (aka *WS1*) has the fastest response time and web service number five (aka *WS5*) has the slowest response time. In test cases

Table 4.1: Routing Table of Router Agent Number 11

Receiver Host IP Address	**Receiver Host Agent Name**	**Destination Link Agent Name**
130.100.1.1	WS1	L_R11_R7
130.100.1.2	WS2	L_R11_R7
130.100.1.3	WS3	L_R11_R7
130.100.1.4	WS4	L_R11_R12
130.100.1.5	WS5	L_R11_R13
100.10.1.1	C1	L_R11_C1
100.10.1.2	C1	L_R11_C2
110.10.1.1	P1	L_R11_P1

Table 4.2: Route Paths and Costs from Proxy Agent to all Web Server Agents

Web Server	**Route Path**	**Route Total Cost**
WS1	R11-R7-R8-R4-R5	8
WS2	R11-R7-R8-R4-R10	10
WS3	R11-R7-R8-R14-R15	12
WS4	R11-R12-R18-R19-R20	8
WS5	R11-R12-R18-R24-R25	9

that require a different order the new values will be explicitly mentioned in the context. The network topology is completely configurable as a matrix of m by n in which m and n are read from configuration file and initialized in startup. The configured network and link costs for this experiment has 5 rows and 5 columns as shown in Figure 3.4. The routing of messages is static and so each web service call will follow a static route to the destination. Based on configured-routing-tables the paths from the proxy agent to each web server and their associated cost value are shown in Table 4.2.

As its seen in Table 4.2, web servers 1 and 4 have the shortest path (lowest cost) to proxy agent and web server 2 has the longest path (biggest cost) to proxy agent. The cost directly affect the delay time in run time. The time that it takes for a message to

Table 4.3: Choosing Algorithm Tested in the Experiment

#	**Algorithm Name**	**Description**
1	Random	Chooses the replicas randomly
2	Best Last	Chooses the replica that had the best last response time
3	Best Median	Chooses the replica that has the best median response time in last k responses. k is a positive number that is set in configuration
4	Parallel	Sends the requests to services in parallel. The winner is the replica that responds first

get from proxy agent to one of these web servers depend on route cost, link delays, router delays, and message length. The message length as explained in Chapter 3 is constant.

4.2 Testing Phase

This experiment analyzes several *Replica Choosing Algorithms*. A brief explanation of utilized algorithms by the simulator is shown in Table 4.3.

Each choosing algorithm is tested separately started from Random selection algorithm. For each algorithm three sets of tests are executed where each set sends five hundred web service requests from clients to proxy agent. Proxy agent decides to which replica the request should be forwarded. The result of each test is recorded by the program and will be put in tables for further analysis and comparisons. Table A.1 lists the collected attributes in each test case. The test is conducted by *Tester Agent* which gets its testing commands from the operator. Figure 4.1 shows the high level UML sequence diagram of test process. Table A.1 lists all the test case scenarios that have been executed in the experiment. In total twenty–one test cases were executed to evaluate and validate the behavior of the simulator. The remainder of this chapter presents and analyzes the

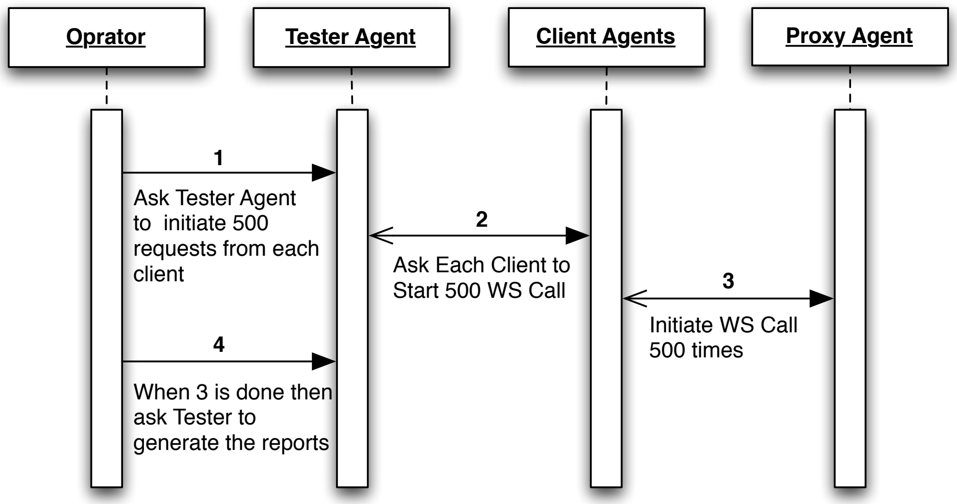
Oprator
Tester Agent
Client Agents
Proxy Agent
1
Ask Tester Agent to initiate 500 requests from each client
2
Ask Each Client to Start 500 WS Call
3
Initiate WS Call 500 times
4
When 3 is done then ask Tester to generate the reports

Figure 4.1: Test Sequence Diagram

result of these test scenarios.

4.3 Random Selection

In *Random* selection each replica is chosen randomly by *Proxy Agent*. Figure 4.2 and Tables A.3 show the results where the Dynamic Processing Behavior (DPB) is off for web servers. Figure 4.3 and Table A.3 show the result where the feature is on. The numbers shown on top of each bar indicates the number of times that each replica was selected by the proxy agent. PA uses a random number generator function to choose each replica and so the distribution of selections should be almost uniform in this algorithm. Each replica is expected to be called equal number of times per each test run. The result of random selection will be used as a reference to compare and analyze the performance of other selection algorithms.

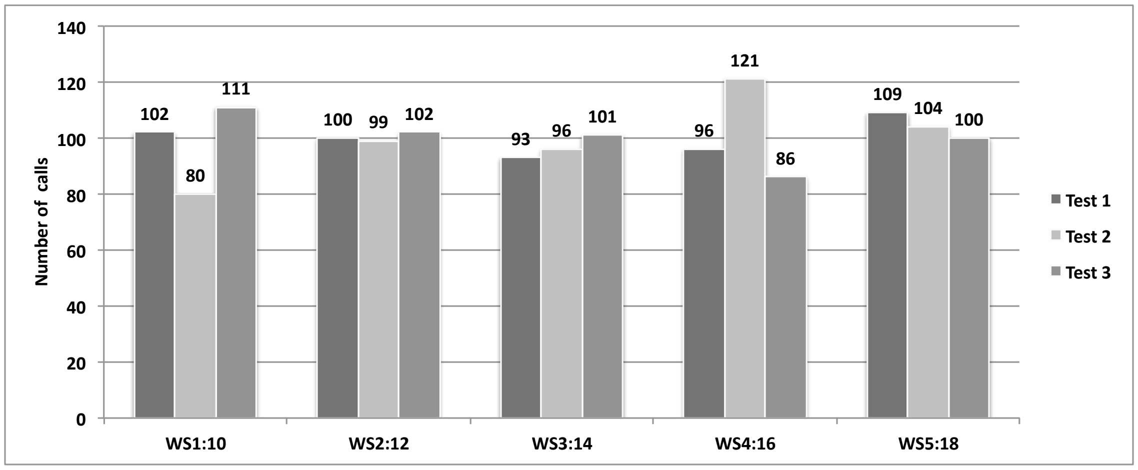

Figure 4.2: Random Replica Selection with Dynamic Processing Behavior Turned Off

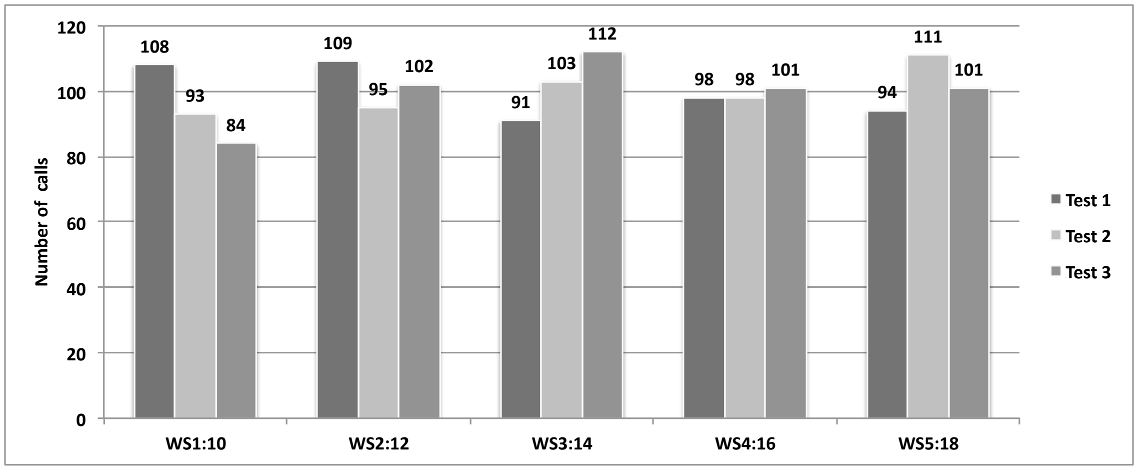

Figure 4.3: Random Replica Selection with Dynamic Processing Behavior Turned On

4.4 Best Last Selection

In *Best Last Selection* algorithm the proxy agent chooses a replica that has the best last response time in the previous request. Proxy agent does not have the response time of replicas at the beginning and needs to collect them in initial requests. In first few requests (in this experiment 5 because the number of replicas is 5) it selects the replicas one by one to collect the response times. Comparing to random selection this algorithm leans towards the web service that provides the best response time. The value of the response time that is perceived by proxy agent is an aggregation of the response time of the replica, plus the response time of hosting web server, and the network delay time that includes the link and router delay values. From client's point of view the final result should provide an overally better response time. Several different test cases have been executed for this algorithm. In the first one the dynamic processing behaviour of web server agents has been turned off that means the processing time of each web server is constant as configured in startup and never changes by time. Figure 4.4 and Table A.5 clearly show that almost every request has been routed to web server 1 that hosts WS1 replica. As expected the client's overall response time have been improved which looks better than random selection as seen in Table A.5. Figure 4.5 and Table A.6 show the result of best last choosing algorithm where DPB is turned on.

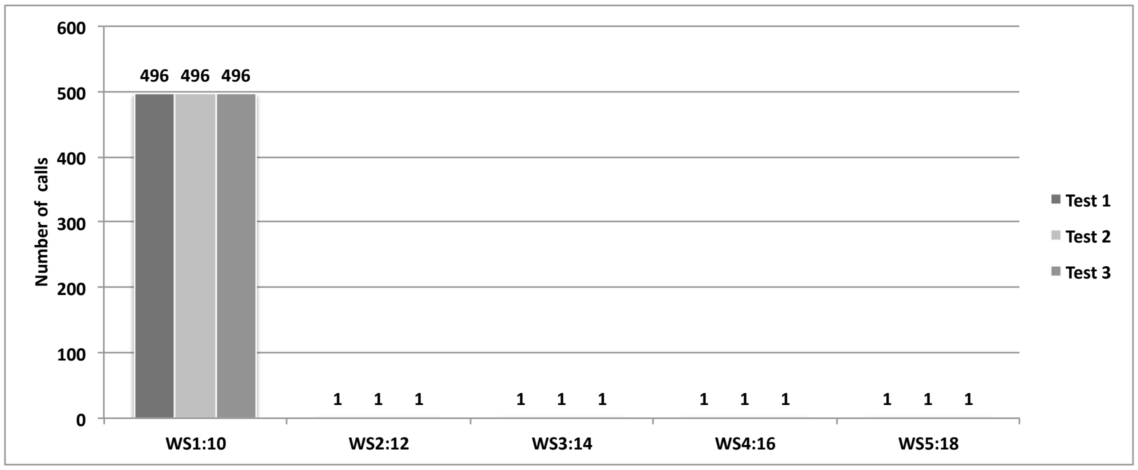

Figure 4.4: Best Last Replica Selection with Dynamic Processing Behavior Turned Off

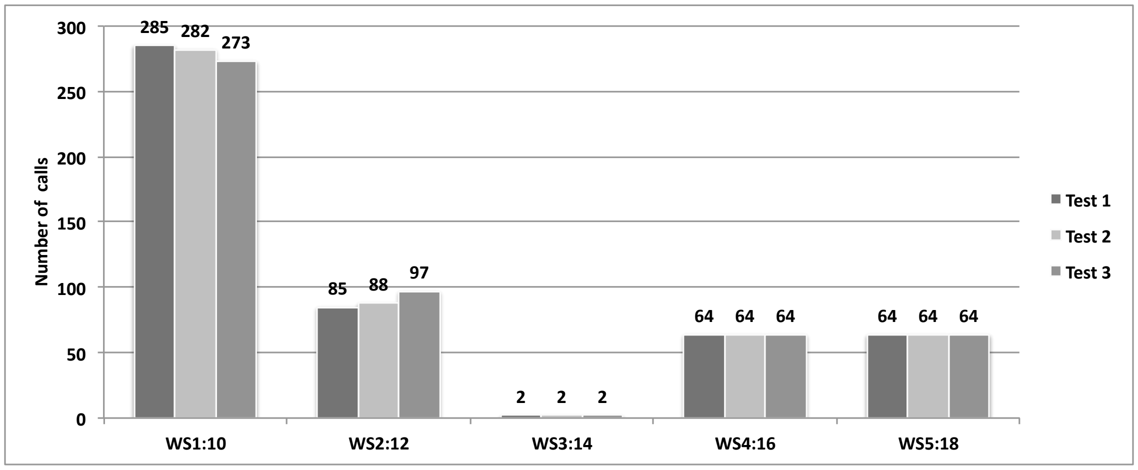

Figure 4.5: Best Last Replica Selection With Dynamic Processing Behavior Turned On

4.5 Best Median Selection

In Best Median Selection the proxy agent first calls each replica one by one at least k times to collect enough history data to calculate the mean values and then it selects the replica that provides the best last mean value. k can be set a value between 1 and total number of tests per each text case (500 in this experiment). Figure 4.6 shows the result for value of k of 1, 5, and 9 where the dynamic processing behavior is turned off. Figure 4.7 shows the result where the dynamic processing behavior is on for value of k equal to 1, 3, 5, 7, and 9. For values of k = 1 as it is seen the behavior is very similar to best last algorithm. Figure 4.8 shows the result where processing speed of web servers are different.

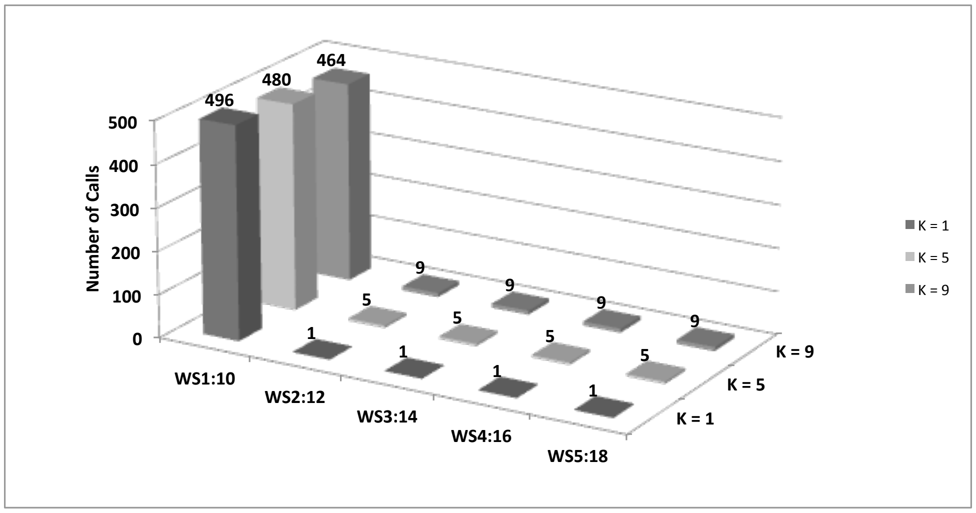

Figure 4.6: Best Median Replica Selection With Dynamic Processing Behavior Turned off for values of k equal to 1, 5, and 9

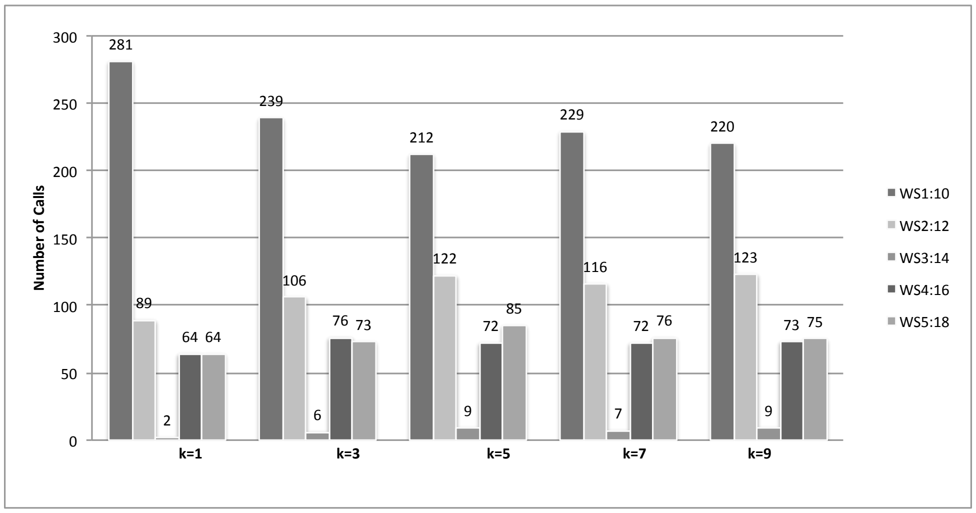

Figure 4.7: Best Median Replica Selection With Dynamic Processing Behavior Turned ON for values of k equal to 1, 3, 5, 7, and 9

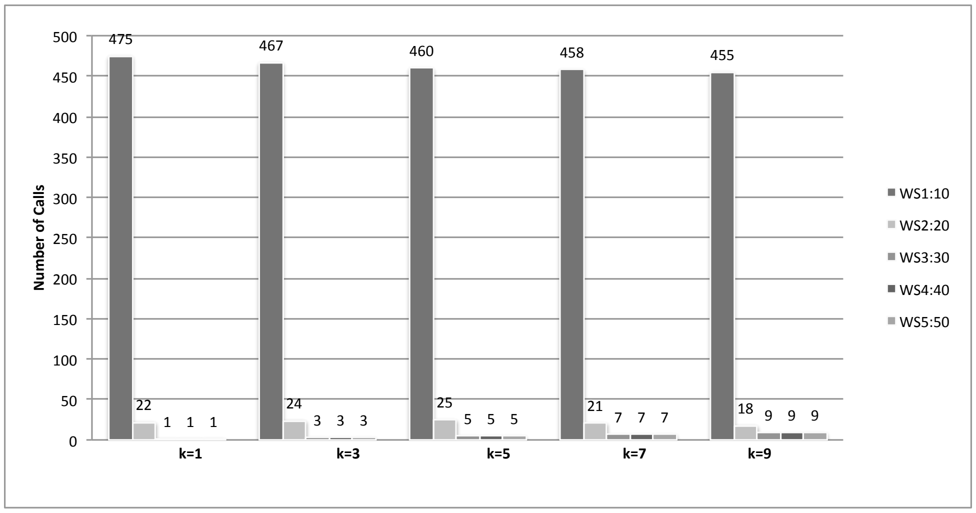

Figure 4.8: Best Median Replica Selection With Dynamic Processing Behavior Turned ON for values of k equal to 1, 3, 5, 7, and 9 and different static processing time

As k increases the distribution of service calls become more even because proxy agent has to collect more statistical information from each replica. Best median response algorithm does provide any advantage to best last algorithm in terms of client Average Response Time (ART). This could be observed by comparing client ART values in Table A.6 and Table A.10. Best median is actually a little bit slower to clients because proxy agent has to do more calculations to choose the right replica. On the other hand best last median is less sensitive than best last to abrupt changes in response times due to events happen in web servers or the residing network between the server and the client. This makes this algorithm more biased toward the already selected replica unless there is a big change in response time of that replica to persuade proxy agent to switch to another replica. This feature will be shortly analyzed in section 4.7.

4.6 Parallel Selection

In Parallel Selection Algorithm the proxy agent sends the client's requests to all web servers at the same time. The first response that comes back to proxy agent will be the winner and that response will be forwarded back to the client agent. Proxy agent collects and saves the statistics and responses of all replicas even though only one of them is the winner. Figure 4.9 and Table A.11 show the results for parallel selection algorithm where the dynamic processing behavior is off. Figure 4.10 and Table A.12 show parallel selection where the dynamic processing behavior is on. Figure 4.11 shows the result for parallel selection with dynamic processing behaviour turned on similar to Figure 4.10 but with different web servers speeds.

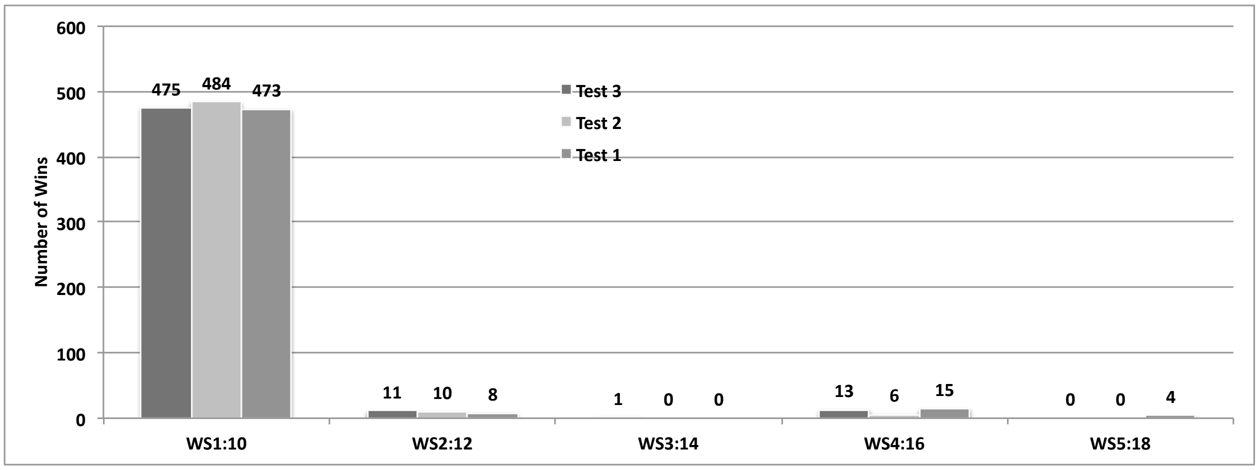

Figure 4.9: Parallel Replica Selection With Dynamic Processing Behavior Turned Off

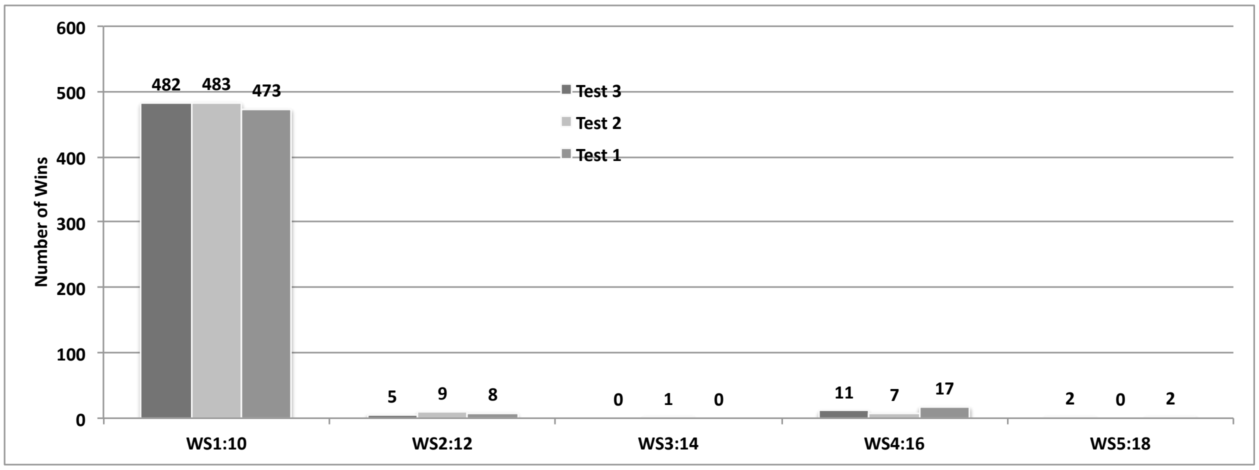

Figure 4.10: Parallel Replica Selection With Dynamic Processing Behavior Turned ON

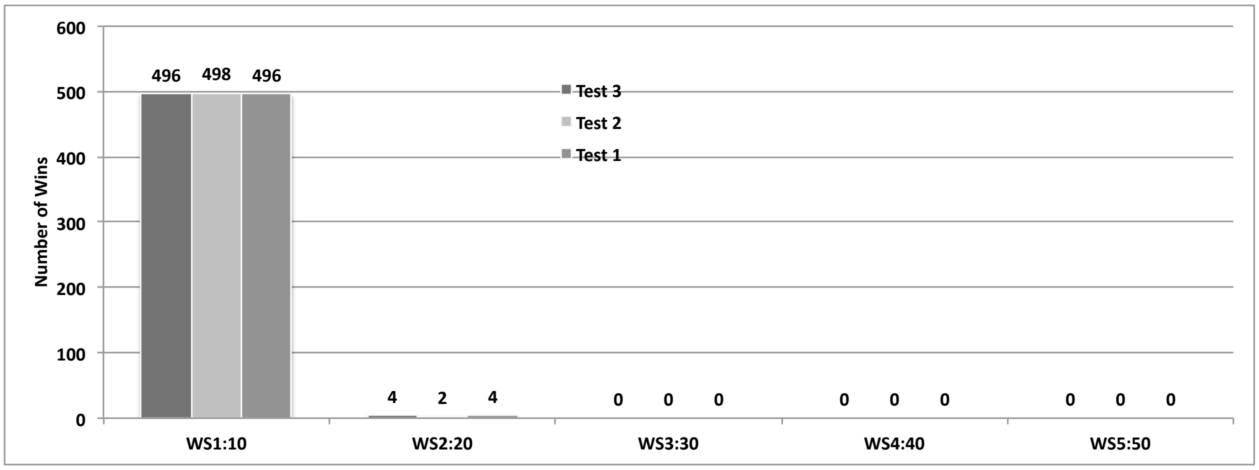

Figure 4.11: Parallel Replica Selection With Dynamic Processing Behavior Turned ON for different web server speeds

In parallel selection all the web servers receive web service requests from proxy agent. This makes this algorithm more fair than previous algorithms in terms of replica selection but the number of requests that are sent to servers put lot of traffic on network and web servers that could affect the overall response time as clearly shown in Table A.12 that shows considerably more client ART value comparing to best last, best median, and even random selection algorithms. The great advantage of parallel algorithm is on service availably. If for any reason the web servers become unavailable due to server failure or network issues, the parallel selection algorithm is the most reliable choice. It is more unlikely that all servers go down at the same time and so at least one of the servers could respond to client.

4.7 Simulator Evaluation

As indicted before the purpose of this experiment is not the evaluation of choosing algorithms, a work that has been done in many other experiments including [19, 18]. The intention of this experiment is to evaluate the simulator application behavior. Thanks to agent systems, the simulator program is highly flexible and and could provide test case scenarios that are extremely hard or impossible to achieve in real scenarios. For each test scenario that follows the behavior of the simulator is analyzed and verified.

We start from *Last Best Algorithm*. It is known and also logically easy to conclude that this algorithm does not provide a fair distribution when it comes to replica selection [19, 18]. This means a great majority of requests will be forwarded toward the replica that provides the fastest recent response to proxy server. On the other hand Last Best is a biased algorithm, a characteristic that has been shown in Figure 4.4. In reality, the distribution of calls is not very biased to only one web server as shown in Figure 4.5 where the dynamic processing behaviour is turned on for all web server agents. This means each web service response time dynamically changes based on the number of incoming requests. The more requests that flows toward a web server the more time it needs to process it.

Other requests have to wait to take turn and that is what happens in reality. If a web server receives a great number of requests per unit of time its response time will increase. This behaviour has been simulated in each web server agent. The result shown in Figure 4.5 and Table A.6 show a better distribution in terms of replica selections comparing to Figure 4.4 and Table A.5, which does not necessarily mean a better overall response time from client's perspective. Being biased toward a specific replica may not be desirable for the selected web server that would face lot of network traffic flow but will be desirable to clients of that server if it is much faster than other hosting servers.

An interesting observation in Figure 4.5 shows that web servers 4 and 5 provide better overall response time comparing to web server number 3 even though the web server 3 is faster than the other two. This is the effect of network response time that lies between proxy agent and these web servers. Web server 3 has a slower network connection to proxy agent (total cost of 12) than web servers 4 and 5 (see Table 4.2). Web server 2, even though has a slower connection than number 4 and 5 but is fast enough to beat them. Obviously the proxy agent perception is important here that determines the final winner. This test shows how network latency affects the overall response time, a known fact, and a confirmation of a valid behavior of router and link agents.

A little change in processing power of each web server shows how that affects the replica selection pattern. In next experiment shown in Figure 4.12 and Table A.7 web servers 4 and 5 have been configured to be much slower than web servers 1, 2, and 3. The result shows how the faster web servers steel the requests from slower web servers even though the network costs are in favour of slower web servers. Look at majority of calls that have been sent over to web servers 1 and 2 in Figure 4.12 and compare it with results shown in Table 4.5. This is expected and indicates another correct behavior of the simulator.

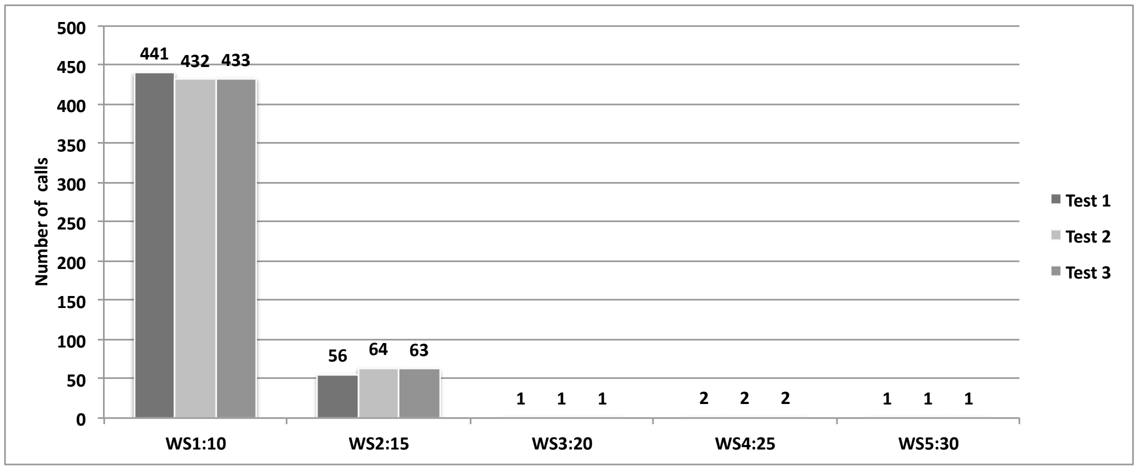

Figure 4.12: Best Last Replica Selection With Different Server Response Times and Dynamic Processing Behavior On

In another interesting test case the processing time of routers 4, 7, and 8 has increased from 2ms to 10ms to show how the network response time could affect the replica selection pattern. By looking at Table 4.2 it is clear that routers 4, 7, and 8 will be used to send messages from proxy agent to web servers 1 and 2, the two fastest servers in this distributed web server system. The results of this change, shown in Figure 4.13 and Table A.8, indicate how differently the replicas are selected with this change. The majority of requests are now forwarded to web servers 4 and 5 even though their response time are much slower than web servers 1 and 2. This experiment shows how easily an MAS–based simulator could be tuned to execute test scenarios that are impossible to run in real environments. In real environments the components that constitute the network are out of reach and could not accessed and configured by the experimenter.

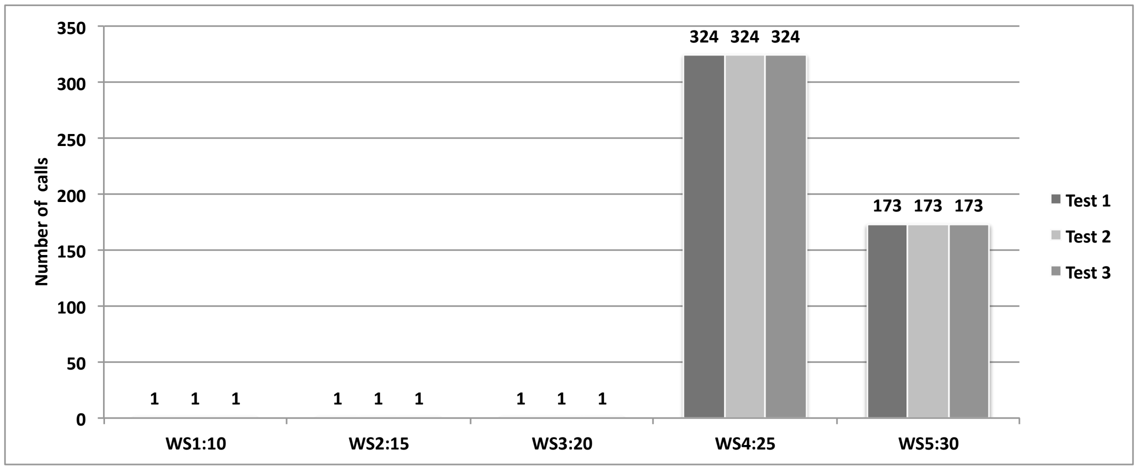

Figure 4.13: Changing the Response time of Routers in Best Last Replica Selection

It was indicated before that Best Last algorithm inclines to the replica that has had the best response time that means Best Last forgets about the other replicas as long as the first chosen replica provides the best response time. This could have a positive or negative impact on client's overall response time that could be easily affected due to fluctuations in network traffic or web servers availability.

If these dynamic fluctuations provides a faster response time for the already selected replica then the algorithm sticks even more to that replica and so the overall response time will improve (decrease) but if these fluctuations results in faster response times for other replicas, which cannot be perceived by the proxy agent, then the overall response time will be longer (increase) even though the other replica could have provided better response times if they were selected.

In best last algorithm the Proxy agent only collects history information of the replicas that it communicates with. If Proxy agents forward all requests to one server then it can only collect the responses from that server so the history data for other replicas becomes out dated over time. This means the proxy agent will never switch to another replica as long as the response time of the chosen replica is better than the response time of other replicas. This behaviour is actually one of the drawbacks of this algorithm. This drawback could be compensated to force proxy to communicate and ping other server periodically. This is doable but will consume a portion of processing time of Proxy agent. This behavior could be considered as a different choosing algorithm which could be easily implemented using this platform but will be left as one of future tasks.

This mentioned drawback could be viewed from another perspective. Suppose for any reasons the proxy agent perceives a negative spike in response time of an already selected replica. That spike should persuade proxy agent to switch to different replica that has the second best response time in its history data. Best last always inclines to replica that has the best last response time so due to this spike it stays with the second replica even though the spike could be temporary and the original replica could provide the best response time afterward. This behavior could be easily simulated by our platform. A spike or communication turbulence could be simulated by changing the behaviour of the related agents. For instance we could enforce WS1 Agent (fastest agent in the experiment) to artificially prolong its response time for a short period of time or for a specific request payload, then we could observe and measure the impact of spike. Figure 4.14 and Table A.9 show how the algorithm will react to this spike in WS1. The proxy agent selects the next best responsive agent (WS2) and it inclines to stay with WS2 afterward in majority of remaining requests. The overall client response time (average and worst response times) have also increased a little bit that is because of slower response time of WS2. As it is shown in Figure 4.14, in the first test case the spike is programmed to happen in 20th requests. In the second test case the spike happens in 40th request, and in the last test case the spike happens in request 100 (information such as request number is fully logged by the simulator application in each test run).

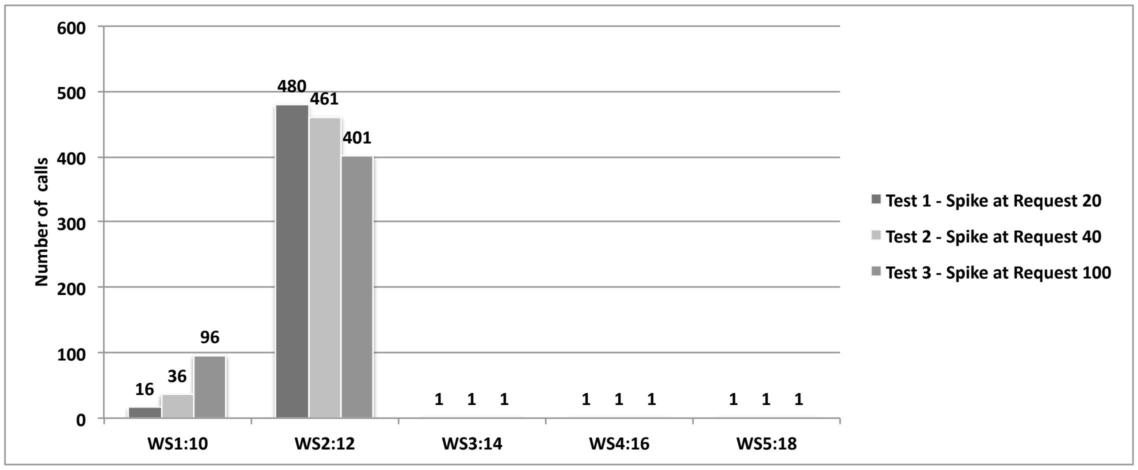

Figure 4.14: Effect of Spike in Best Last Selection Algorithm - Spikes happen in WS1 with a jump of 20 ms in predefined request messages

If the same experiment is executed for Last Median algorithm, that injects a spike of 20ms in request number 20 (shown in Figure 4.15) the replica selection patterns changes in favour of WS1. Depending on the value of k the proxy agent tries to forgive the unexpected rise in response time in WS1 and chooses that replica because now it looks at the last average value of k response times and not only the last response time. For k =2 the system stays with WS2 longer until request number 252 and then it switches back to WS1 in request number 253. For k = 3 the proxy agent stays with WS1 when the spikes occurs at request number 20 because the last mean value of WS1 stays lower value than last mean value of WS2 even though the spike is a big amount.

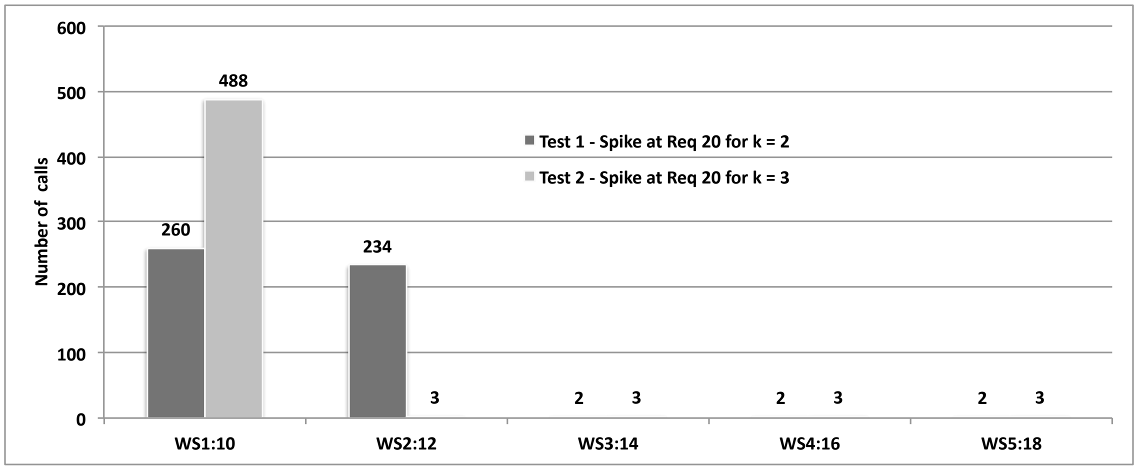

Figure 4.15: Comparison of Spike in Best Last and Best Median Selections - For k=2 the proxy agent switches back to WS1 at request number 253. For k=3 the proxy agent stays with WS1

Figure 4.17 compares the client response times for all choosing algorithms while the dynamic processing behavior is off and Figure 4.18 shows the comparison when it is on. From client perspective Best Last and Best Median provides the best response times and Parallel selection provides the worst response times even comparing to Random selection. The results for parallel selection are not surprising because in this approach the proxy agent sends requests to all replicas and so all hosting web servers will receives equal number of requests. This algorithm sends a lot of messages to all web servers that causes lot of pressure on each server. This in return causes a much slower response time. In Random selection only one of replicas receives the request so the servers will not become overloaded. In parallel selection the proxy agent also becomes busier because it needs to take care of much more responses (five times comparing to best last, best median, and random methods in this experiment). That puts pressure on Proxy agent that contributes negatively to overall response time perceived by the client.

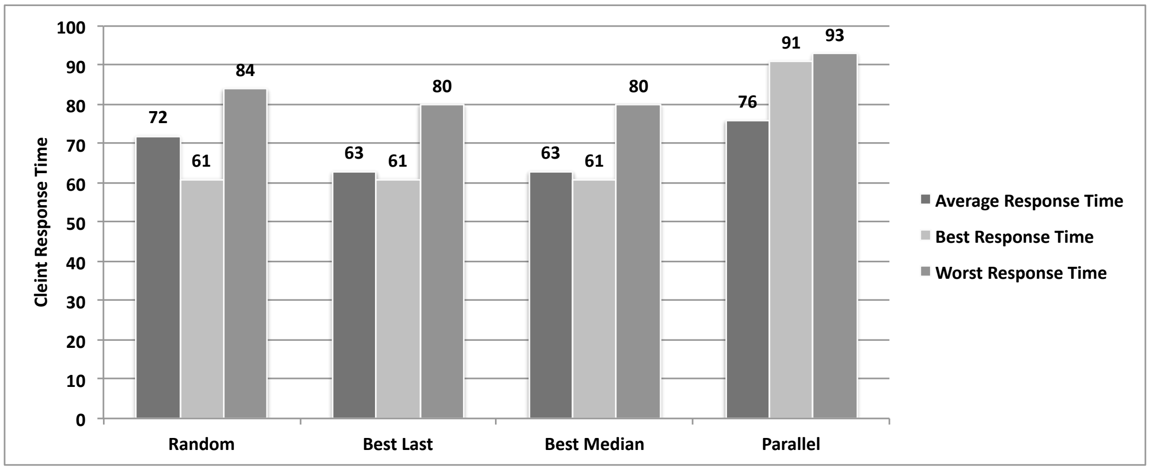

Figure 4.16: Comparison of Single Client Response Times when Dynamic Processing Behavior is Off

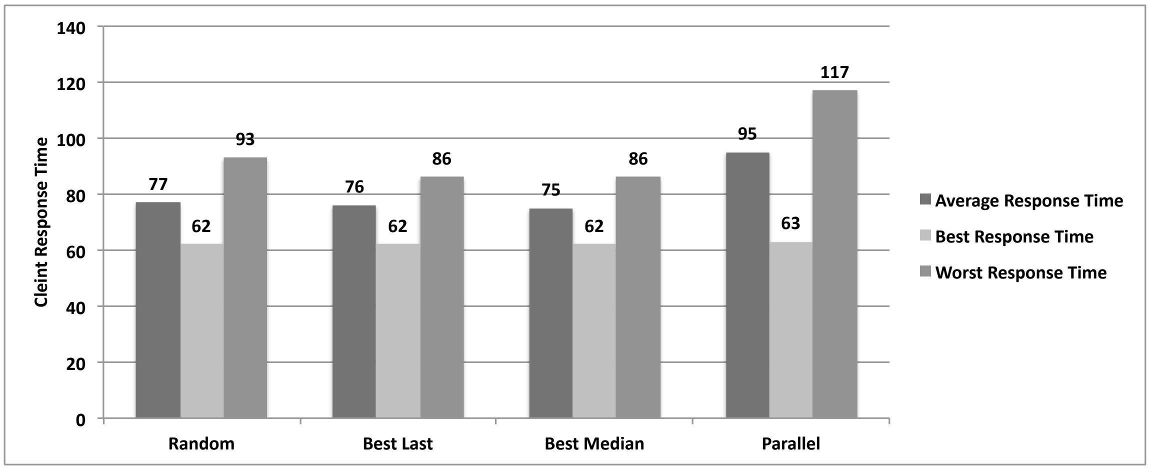

Figure 4.17: Comparison of Single Client Response Times when Dynamic Processing Behavior is On

4.8 Effect of More than One Client

The simulation platform is capable to simulate an unlimited number of clients. In this test scenario two client agents send the same number of requests (500) to proxy agent. This experiment has been executed for all selection algorithms. Figure 4.18 and 4.19 show the results when there are two clients and the choosing algorithm is Random. The first one is for dynamic processing behavior turned off and the later one when this feature in on. Figure 4.20 and 4.21 show the same results for Best Last selection and Tables A.13 to A.16 show the experiment results for all choosing algorithms while the dynamic processing feature is turned on. From clients point of view the response times are the best when selection algorithms are best last and best median but worst when the selection algorithm is parallel. General response times is worse than one client scenario which is expected. The more number of clients means the worse response time in overall because all servers are busier and hence slower (see Figure 4.22).

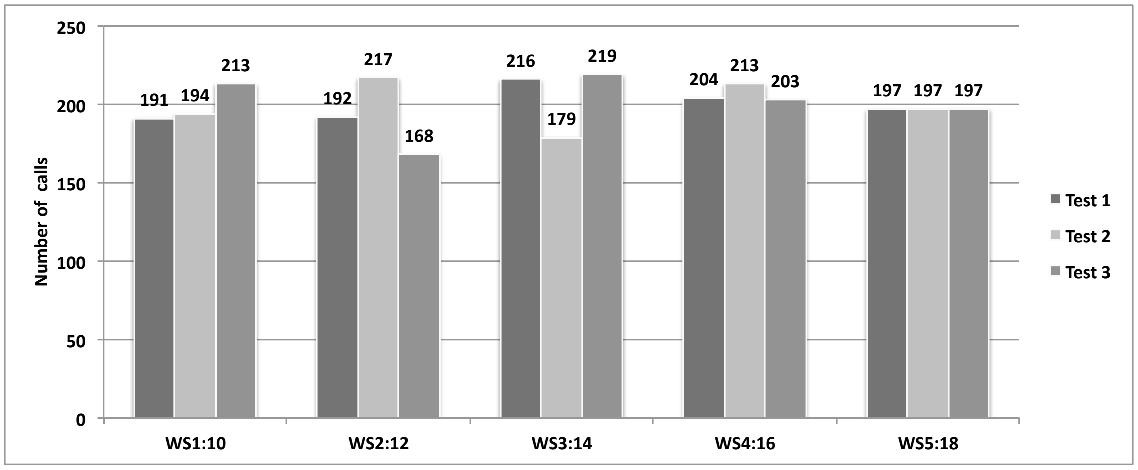

Figure 4.18: Random Selection: Two Clients, Dynamic Processing Behavior = Off

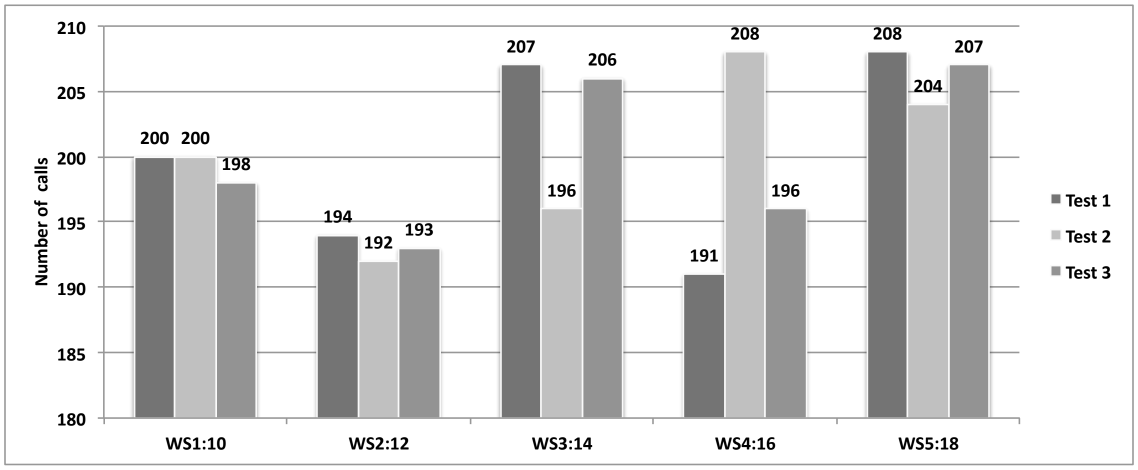

Figure 4.19: Random Selection: Two Clients, Dynamic Processing Behavior = On

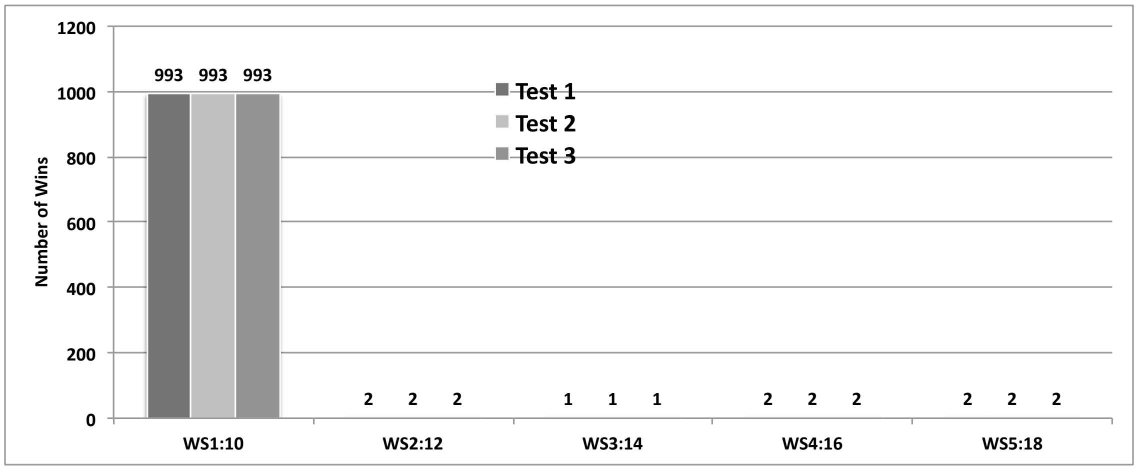

Figure 4.20: Best Last Selection: Two Clients, Dynamic Processing Behavior = Off

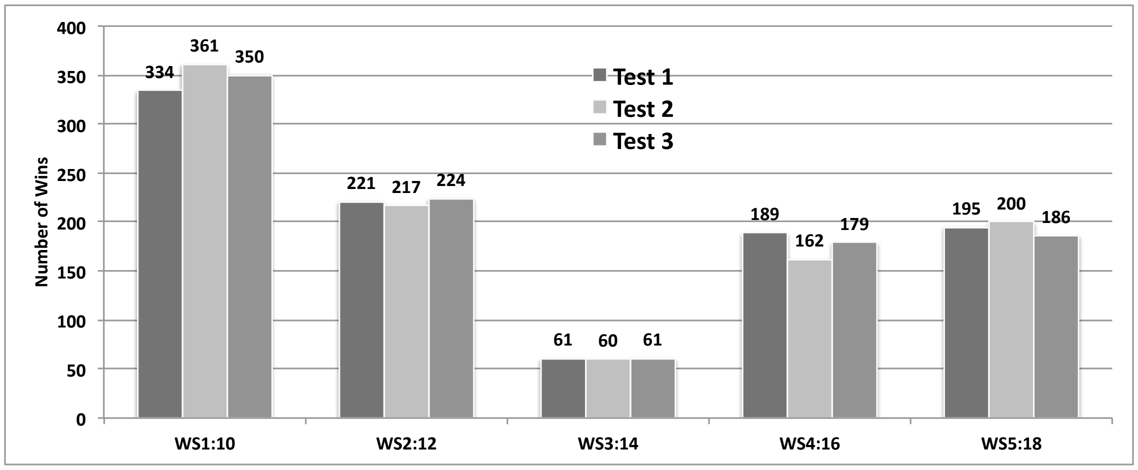

Figure 4.21: Best Last Selection: Two Clients, Dynamic Processing Behavior = On

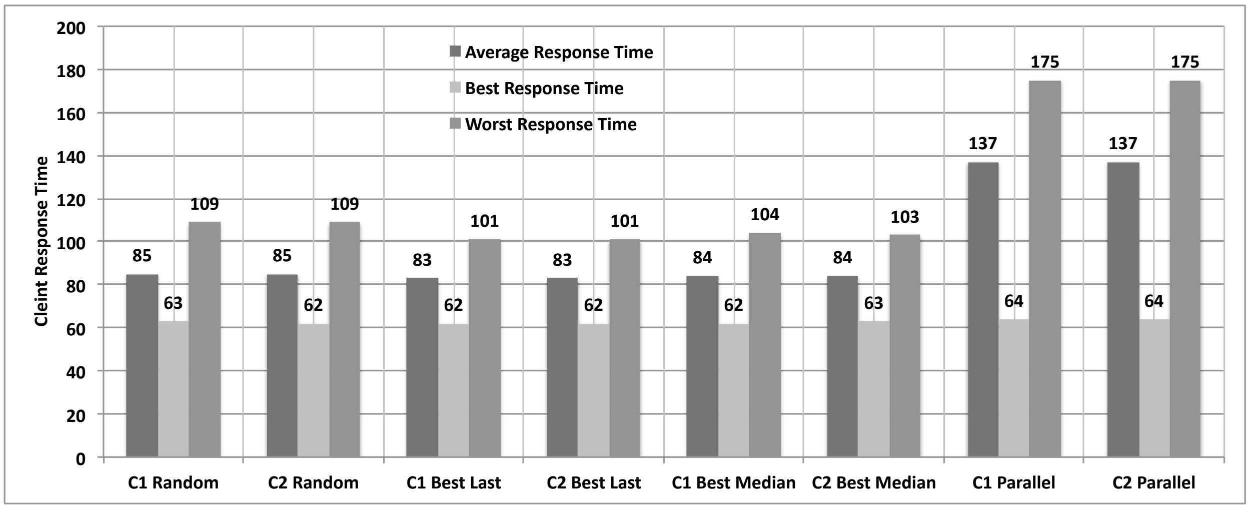

Figure 4.22: Comparison of Two Client Response Times when Dynamic Processing Behavior is On

Chapter 5

Conclusion and Future Work

Agent systems have been utilized in many applications particularly in simulation but a very few works show they effective usage in simulation of distributed applications and computer networks. Development and maintenance of agent systems is fast and easy thanks to tools like JADE; however the agent runtime environment requires lot of processing power. A big concern about Multi-Agent Systems and their usage in simulation of large scale systems like computer networks and distributed systems is their performance and scalability. This concern has not been clearly addressed or even mentioned in the related literature and works, at least the ones that were reviewed for this thesis. The outcome of this thesis has helped to evaluate and clarify this concern.

5.1 Conclusion

Design, development, and testing of distributed systems such as a web service based application is a complex process that demands a lot of resources. From software engineering perspective, each stage in development of a distributed application, from design and architecture to implementation, deployment, and testing phase is time consuming, resourceful, and expensive. Many design and development tools have been created to improve this process but the growing complexity of such applications demands for more innovative and effective methods and tools. Simulation of such systems could be an effective method to help architects and developers to evaluate their designs before starting the real implementation but a big challenge in simulation of distributed systems is the underlying computer network.

The main purpose of this experiment was to show how effectively a multi-agent system could be used to simulate the behavior of a computer network and complex distributed application. A replica based web service system was chosen as an example to show how this simulation could be realized. This thesis could be considered as an extension to a previous work presented in [24]. Contrary to that approach that used utility functions to simulate a network, in a single thread environment, in this model the agents were used to handle this task. Each agent is running in its own thread of execution as an independent and autonomous entity that makes it very suitable to simulate the behavior of decentralized processing nodes in a distributed system. The underlying network was modelled as a matrix of nodes and links where each node represents a router and each link represents a communication link. The core functionality of real routers and communication links were simulated by agents.

Higher level components in web service application like clients, web servers, and proxy servers were also modelled and constructed as agents. Each of these components in a distributed system are host nodes. A host node in a computer network is a node that is either initiates a request message or consumes a request message. The host node agents in this simulation platform handle two major tasks. The first task enables them to connect and deliver message to the network or receive a message from the network and the second task simulates the real business logic of the simulated entity. The first task does not change per application but the second one could be changed per application requirement. For a new application only the second task should be changed and this makes this architectural model very flexible in simulation of a verity of distributed applications.

The simulator application used to execute a series of well known and already tested replica choosing algorithms to evaluate its bahavior as a distributed application. Agents are extremely easy to develop, maintain, and configure and during the development or experiment phase these features became very handy and useful. The challenge was to make each agent similar to its related real entity as closely as possible. If modelling of each component is done correctly then the overall behavior of mutli-agent system converges to behavior of the real system. Even though the simulation of each individual entity was not perfect and comprehensive, the obtained results were very promising and showed that we were in the right direction.

The simulated network consisted of 67 agents in total. Each test case took between 40 to 50 seconds for processing of 500 requests in a notebook computer. A preliminary test showed that the simulator application, on the current running environment, could accommodate at least couple of thousands of agents with simulation time not more than several minutes. These numbers were promising and indicated that agent systems are scalable enough to be used in simulation of more complex networks and distributed systems.

5.2 Future Work

Many enhancements could be done in future to improve the functionality and behavior of the simulator application. Here is a list of some of them:

I: **Creating of more complex router and link agents:** A challenging part of this experiment was creating a platform that could simulate the behavior of a real computer network. Simulation of a network is not an easy task because the network behavior is very dynamic and its performance depends on many dynamic variables. This dynamic and complex behavior is not predictable in runtime. The router and link agents used in this platform only simulate a portion of the job of their real counterparts. For instance the router agents in this experiment cannot provide dynamic routing. They also do not provide packet switching and handling. The router agents in this experiment do not fail but the real ones could! A great improvement in this simulation platform is to make the behavior of router and link agents more similar to behavior of their real counterparts. This does not require extensive work and is doable. In fact with a little effort the router and link agents could be packaged and used as a standalone network simulator.

II: **Creating of message protocols similar to OSI network layers:** Current simulation protocol uses a proprietary message protocol for host node, routers and link agents. That could be improved to be make it compliant with real message protocols used in Transport, Network, Link, and Physical layers in OSI or Internet protocol stack. This requires the implementation of more complex behaviors in host nodes to simulate transport layer functionality like packet retransmission and fail over mechanisms.

III: **Creating of more complex web server agents:** The behavior and interface of current web server agents and web services could be improved to imitate the behavior of their real counterparts. For instance we could enhance the messaging protocol to support real SOAP and HTTP messages.

IV: **Creating of more complex web services:** The web service that was used in this experiment was a simple length converter that accepts one decimal number and returns a decimal number. More complex web services could be implemented that accept more complex data structures and return larger data payloads so we could analyze the beahvior and performance of the simulator in processing of large message payload.

V: **Using more than one proxy agent in a clustered or peer-to-peer architecture:** The business logic of proxy agent as mentioned before is a portable code that could be used in real scenarios; however using only one proxy agent that handles all these tasks on behalf of clients produces a common design flaw: *Point of failure*. We could fix this flaw by having more than one proxy agent in a clustered or peer-to-peer architecture. In either approach the proxy agents need to communicate with each other to synchronize the history information. This approach should provide a better overall response time to client.

VI: **Deploying the simulator in a distributed running environment:** This experiment was conducted on a single computer and so all the agents were living and running in the same physical computing environment. MAS runtime environments including JADE allow a distributed running environment. For instance with little effort the proxy, client, and web server agents could be deployed in different computers or Java virtual machines. Routers and link agents could also be spread between these environments. This makes the simulation even closer to reality.

VII: **Creating a better graphical user interface for initializer and tester agents:** In this experiment the simulator is communicated through a default user interface that is provided by JADE. A better graphical user interface could be created to establish a easier and more intuitive communication with administrative agents (Initializer and Tester agents) and to configure and run the experiment faster.

Appendix A

Experiment Results

Table A.1: Test Scenarios Executed in the Experiment

#	**Test Case Description**	**Figure**	**Table**
1	Random selection with DPB turned off	4.2	A.3
2	Random selection with DPB turned on	4.3	A.4
3	Best last selection with DPB turned off	4.4	A.5
4	Best last selection with DPB turned on	4.5	A.6
5	Best median selection with DPB turned off for values of k equal to 1, 5, and 9	4.6	–
6	Best median selection with DPB turned on for values of k equal to 1, 3, 5, 7, and 9	4.7	A.10
7	Best median selection with DPB turned on for values of k equal to 1, 3, 5, 7, and 9 and different processing speeds for web servers	4.8	–
8	Parallel selection with DPB turned off	4.9	A.11
9	Parallel selection with DPB turned on	4.10	A.12
10	Parallel selection with DPB turned on and different web server speeds	4.11	–
11	Best last selection with DPB turned on and different web server speeds	4.12	A.7
12	Best last selection with DPB turned on and different router response time	4.13	A.8
13	Injecting spike in response time of the fastest web server for best last selection algorithm	4.14	A.9
14	Injecting spike in response time of the fastest web server for best median selection algorithm for two different values of k	4.15	–
15	Random selection with two client agents and DPB turned off	4.18	–
16	Random selection with two client agents and DPB turned on	4.19	A.13
17	Best last selection with two client agents and DPB turned off	4.20	
18	Best last selection with two client agents and DPB turned on	4.21	A.14

Table A.2: Meaning of Collected Attributes in each Test Scenario

Attribute Name	**Abbr.**	**Meaning**
Number of Requests	RN	Number of requests processed by the replica
Web Server Average Response Time	WS ART	Average response time of all requests processed by the web server
Web Server Best Response Time	WS BRT	The best response time provided by the web server
Web Server Worst Response Time	WS WRT	The worst response time provided by the web server
Proxy Average Response Time	Proxy ART	Average response time observed by proxy agent
Proxy Best Response Time	Proxy BRT	The best response time observed by proxy agent
Proxy Worst Response Time	Proxy WRT	The worst response time observed by proxy agent
Proxy Last Medium Response Time	Proxy LMRT	Last medium response time observed by the web server. This value is meaningful only in *Last Median* algorithm
Client Average Response Time	Client ART	Average response time perceived by client
Client Best Response Time	Client ART	Best response time perceived by client
Client Worst Response Time	Client ART	Worst response time perceived by client

Table A.3: Random Replica Selection with Dynamic Processing Behavior OFF

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	102	100	93	96	109
	T2	80	99	96	121	104
	T3	111	102	101	86	100
Proxy ART	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy LRT	T1	60	68	79	71	71
	T2	60	68	79	71	71
	T3	60	68	79	71	71
Proxy BRT	T1	57	65	76	68	68
	T2	57	65	76	68	68
	T3	57	65	76	68	68
Proxy WRT	T1	60	68	79	71	71
	T2	60	68	79	71	71
	T3	60	68	79	71	71
Proxy LMRT	T1	60	68	79	71	71
	T2	60	68	79	71	71
	T3	60	68	79	71	71
WS ART	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS BRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS WRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
Client ART	Test 1 = 72, Test 2 = 73, and Test 3 = 72					
Client BRT	Test 1 = 61, Test 2 = 61, and Test 3 = 61					
Client WRT	Test 1 = 84, Test 2 = 85, and Test 3 = 84					

Table A.4: Random Selection with Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	108	109	91	98	94
	T2	93	95	103	98	111
	T3	84	102	112	101	101
Proxy ART	T1	63	71	81	73	73
	T2	62	71	82	74	74
	T3	62	71	82	74	74
Proxy LRT	T1	66	71	85	75	75
	T2	61	76	85	73	80
	T3	64	72	85	77	76
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	68	78	86	80	78
	T2	66	77	90	79	80
	T3	68	75	88	79	78
Proxy LMRT	T1	65	71	84	75	76
	T2	64	75	82	75	77
	T3	65	72	83	76	75
WS ART	T1	19	22	23	25	27
	T2	19	21	23	25	28
	T3	19	21	24	25	27
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	24	27	26	30	30
	T2	22	26	30	30	32
	T3	23	26	28	30	31
Client ART	Test 1 = 77, Test 2 = 77, and Test 3 = 78					
Client BRT	Test 1 = 62, Test 2 = 62, and Test 3 = 62					
Client WRT	Test 1 = 91, Test 2 = 95, and Test 3 = 93					

Table A.5: Best Last Replica Selection with Dynamic Processing Behavior OFF

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	496	1	1	1	1
	T2	496	1	1	1	1
	T3	496	1	1	1	1
Proxy ART	T1	58	65	76	68	68
	T2	58	65	76	68	68
	T3	58	65	76	68	68
Proxy LRT	T1	58	65	76	68	68
	T2	58	65	76	68	68
	T3	58	65	76	68	68
Proxy BRT	T1	57	65	76	68	68
	T2	57	65	76	68	68
	T3	57	65	76	68	68
Proxy WRT	T1	60	65	76	68	68
	T2	60	65	76	68	68
	T3	60	65	76	68	68
Proxy LMRT	T1	60	65	76	68	68
	T2	60	65	76	68	68
	T3	60	65	76	68	68
WS ART	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS BRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS WRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
Client ART	Test 1 = 63, Test 2 = 63, and Test 3 = 63					
Client BRT	Test 1 = 61, Test 2 = 61, and Test 3 = 61					
Client WRT	Test 1 = 80, Test 2 = 80, and Test 3 = 80					

Table A.6: Best Replica Selection with Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	285	85	2	64	64
	T2	282	88	2	64	64
	T3	273	97	2	64	64
Proxy ART	T1	71	71	78	73	73
	T2	71	71	78	73	73
	T3	71	71	78	73	73
Proxy LRT	T1	80	80	80	80	80
	T2	81	71	80	80	80
	T3	81	80	80	80	80
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	80	80	80	80	80
	T2	81	80	80	80	80
	T3	81	80	80	80	80
Proxy LMRT	T1	80	75	78	76	76
	T2	80	74	78	76	76
	T3	80	75	78	76	76
WS ART	T1	28	22	20	24	26
	T2	27	22	20	24	26
	T3	27	22	20	24	26
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	2
WS WRT	T1	35	29	20	30	32
	T2	36	29	20	30	32
	T3	36	29	20	30	32
Client ART	Test 1 = 76, Test 2 = 76, and Test 3 = 76					
Client BRT	Test 1 = 62, Test 2 = 62, and Test 3 = 62					
Client WRT	Test 1 = 86, Test 2 = 86, and Test 3 = 86					

Table A.7: Best Last Replica Selection with Dynamic Processing Behavior ON and Different Web Server Processing Speeds

Attribute	**T#**	**WS1(10)**	**WS2(15)**	**WS3(20)**	**WS4(25)**	**WS5(30)**
Num of Hits	T1	441	55	1	2	1
	T2	432	64	1	2	1
	T3	433	63	1	2	1
Proxy ART	T1	76	73	83	79	81
	T2	75	73	83	79	81
	T3	76	73	83	79	81
Proxy LRT	T1	80	81	83	81	81
	T2	80	81	83	81	81
	T3	80	81	83	81	81
Proxy BRT	T1	58	69	83	78	81
	T2	58	69	83	78	81
	T3	58	69	83	78	81
Proxy WRT	T1	81	81	83	81	81
	T2	81	81	83	81	81
	T3	81	81	83	81	81
Proxy LMRT	T1	80	76	83	79	81
	T2	80	76	83	79	81
	T3	80	76	83	79	81
WS ART	T1	32	24	26	31	36
	T2	32	24	26	31	36
	T3	32	24	26	31	36
WS BRT	T1	16	21	26	31	36
	T2	16	21	26	31	36
	T3	16	21	26	31	36
WS WRT	T1	36	30	26	31	36
	T2	36	31	26	31	36
	T3	36	30	26	31	36
Client ART	Test 1 = 80, Test 2 = 80, and Test 3 = 80					
Client BRT	Test 1 = 62, Test 2 = 62, and Test 3 = 62					
Client WRT	Test 1 = 87, Test 2 = 87, and Test 3 = 87					

Table A.8: Best Last Replica Selection with Dynamic Processing Behavior ON and Router Response Time Changes - Response Time of R4, R7, and R8 increased from 2 to 10

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	1	1	1	324	173
	T2	1	1	1	324	173
	T3	1	1	1	324	173
Proxy ART	T1	106	109	115	92	89
	T2	106	109	115	92	89
	T3	106	109	115	92	89
Proxy LRT	T1	106	109	115	100	101
	T2	106	109	115	100	101
	T3	106	109	115	100	101
Proxy BRT	T1	106	109	115	78	81
	T2	106	109	115	78	81
	T3	106	109	115	78	81
Proxy WRT	T1	106	109	115	101	101
	T2	106	109	115	101	101
	T3	106	109	115	101	101
Proxy LMRT	T1	106	109	115	100	96
	T2	106	109	115	100	96
	T3	106	109	115	100	96
WS ART	T1	16	21	26	43	43
	T2	16	21	26	43	43
	T3	16	21	26	43	43
WS BRT	T1	16	21	26	31	36
	T2	16	21	26	31	36
	T3	16	21	26	31	36
WS WRT	T1	16	21	26	51	53
	T2	16	21	26	51	53
	T3	16	21	26	51	53
Client ART	Test 1 = 95, Test 2 = 95, and Test 3 = 95					
Client BRT	Test 1 = 82, Test 2 = 82, and Test 3 = 82					
Client WRT	Test 1 = 119, Test 2 = 119, and Test 3 = 119					

Table A.9: Best Last Replica Selection with Dynamic Processing Behavior ON and a Spike of 20ms in WS1 - Spike happens at Request# 20 in Test 1, Request# 40 in Test 2, and Request# 100 in Test 3

Attribute	T#	WS1(10)	WS2(12)	WS3(14)	WS4(16)	WS5(18)
Num of Hits	T1	17	480	1	1	1
	T2	36	461	1	1	1
	T3	96	401	1	1	1
Proxy ART	T1	60	66	76	68	68
	T2	57	66	76	68	68
	T3	57	66	76	68	68
Proxy LRT	T1	70	68	76	68	68
	T2	77	68	76	68	68
	T3	77	68	76	68	68
Proxy BRT	T1	57	65	76	68	68
	T2	57	65	76	68	68
	T3	57	65	76	68	68
Proxy WRT	T1	70	68	76	68	68
	T2	77	68	76	68	68
	T3	77	68	76	68	68
Proxy LMRT	T1	61	68	76	68	68
	T2	67	68	76	68	68
	T3	67	68	76	68	68
WS ART	T1	16	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS BRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS WRT	T1	25	17	19	21	23
	T2	35	17	19	21	23
	T3	35	17	19	21	23
Client ART	Test 1 (Spike at Req# 20) = 70 Test 2 (Spike at Req# 40) = 70 Test 3 (Spike at Req# 100) = 69					
Client BRT	Test 1 (Spike at Req# 20) = 61 Test 2 (Spike at Req# 40) = 61 Test 3 (Spike at Req# 100) = 61					
Client WRT	Test 1 (Spike at Req# 20) = 80 Test 2 (Spike at Req# 40) = 81 Test 3 (Spike at Req# 100) = 81					

Table A.10: Best Median Replica Selection with Dynamic Processing Behavior ON value of k equal to 1, 3, 5, 7, and 9

Attribute	**k**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	1	475	22	1	1	1
	3	467	24	3	3	3
	5	460	25	5	5	5
	7	458	21	7	7	7
	9	455	18	9	9	9
Proxy ART	1	77	76	93	93	101
	3	77	77	94	94	102
	5	77	77	95	95	102
	7	77	78	95	95	103
	9	77	79	96	96	104
Proxy LRT	1	80	81	93	93	101
	3	80	81	94	94	102
	5	80	82	97	97	104
	7	80	83	98	98	106
	9	80	77	100	100	108
Proxy LMRT	1	80	81	93	93	101
	3	80	80	94	94	102
	5	80	80	95	95	102
	7	80	80	95	95	103
	9	80	80	96	96	104
WS ART	1	33	27	36	46	56
	3	33	27	37	47	57
	5	33	28	38	48	57
	7	33	28	38	48	58
	9	34	29	39	49	59
Client ART	1			81		
	3			82		
	5			82		
	7			82		
	9			83		

Table A.11: Parallel Selection with Dynamic Processing Behavior OFF

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	500	500	500	500	500
	T2	500	500	500	500	500
	T3	500	500	500	500	500
Num of Wins	T1	475	11	1	13	0
	T2	484	10	0	6	0
	T3	473	8	0	15	4
Proxy ART	T1	69	76	87	76	76
	T2	69	76	87	76	76
	T3	69	76	87	76	76
Proxy LRT	T1	83	90	103	87	90
	T2	83	90	103	88	89
	T3	83	90	103	87	90
Proxy BRT	T1	57	65	76	68	68
	T2	57	65	76	68	68
	T3	57	65	76	68	68
Proxy WRT	T1	83	90	103	87	90
	T2	83	90	103	88	89
	T3	83	90	103	87	90
Proxy LMRT	T1	82	89	100	85	88
	T2	82	89	100	86	87
	T3	82	89	100	85	88
WS ART	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS BRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
WS WRT	T1	15	17	19	21	23
	T2	15	17	19	21	23
	T3	15	17	19	21	23
Client ART	Test 1 = 76, Test 2 = 75, and Test 3 = 76					
Client BRT	Test 1 = 61, Test 2 = 61, and Test 3 = 61					
Client WRT	Test 1 = 94, Test 2 = 94, and Test 3 = 92					

Table A.12: Parallel Selection with Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	500	500	500	500	500
	T2	500	500	500	500	500
	T3	500	500	500	500	500
Num of Wins	T1	482	5	0	11	2
	T2	483	9	1	7	0
	T3	473	8	0	17	2
Proxy ART	T1	89	96	107	96	96
	T2	88	96	107	96	96
	T3	88	96	107	96	96
Proxy LRT	T1	103	110	123	107	110
	T2	103	111	123	107	110
	T3	103	110	123	107	110
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	6
Proxy WRT	T1	103	110	123	107	110
	T2	103	111	123	107	110
	T3	103	110	123	107	110
Proxy LMRT	T1	102	109	120	106	108
	T2	102	110	120	106	108
	T3	102	109	120	105	108
WS ART	T1	34	36	38	40	42
	T2	34	36	38	40	42
	T3	34	36	38	40	42
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	40	42	44	46	48
	T2	38	40	42	44	46
	T3	39	41	43	45	47
Client ART	Test 1 = 95, Test 2 = 95, and Test 3 = 96					
Client BRT	Test 1 = 63, Test 2 = 62, and Test 3 = 62					
Client WRT	Test 1 = 112, Test 2 = 124, and Test 3 = 114					

Table A.13: Random Selection with Two Clients and Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	200	194	207	191	208
	T2	200	192	196	208	204
	T3	198	193	206	196	207
Proxy ART	T1	70	78	90	80	81
	T2	70	78	90	81	80
	T3	71	78	90	80	81
Proxy LRT	T1	79	84	95	84	91
	T2	77	84	99	89	85
	T3	82	86	97	85	86
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	81	91	101	89	91
	T2	80	89	103	90	91
	T3	83	91	104	92	89
Proxy LMRT	T1	78	85	96	83	90
	T2	76	85	98	88	85
	T3	82	85	96	84	85
WS ART	T1	23	25	27	29	31
	T2	23	25	27	29	31
	T3	23 25	25	27	29	32
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	29	30	34	37	38
	T2	31	30	35	35	37
	T3	28	32	34	36	38
Client 1 ART	Test 1 = 86, Test 2 = 85, and Test 3 = 85					
Client 1 BRT	Test 1 = 102, Test 2 = 64 , and Test 3 = 63					
Client 1 WRT	Test 1 = 108, Test 2 = 109, and Test 3 = 111					
Client 2 ART	Test 1 = 85, Test 2 = 85, and Test 3 = 85					
Client 2 BRT	Test 1 = 63, Test 2 = 62, and Test 3 = 62					
Client 2 WRT	Test 1 = 108, Test 2 = 110, and Test 3 = 109					

Table A.14: Best Last Selection with Two Clients and Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	334	221	61	189	195
	T2	361	217	60	162	200
	T3	350	224	61	179	186
Proxy ART	T1	74	78	83	80	80
	T2	74	78	84	80	80
	T3	75	78	84	80	80
Proxy LRT	T1	71	93	93	93	93
	T2	94	94	94	94	91
	T3	95	85	95	95	94
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	93	93	93	93	93
	T2	94	94	94	94	93
	T3	95	94	95	95	94
Proxy LMRT	T1	85	92	92	92	92
	T2	93	93	93	93	90
	T3	94	84	94	94	93
WS ART	T1	27	25	22	28	30
	T2	27	25	21	28	30
	T3	27	25	22	28	30
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	42	35	27	37	39
	T2	41	35	27	38	39
	T3	42	36	27	39	40
Client 1 ART	Test 1 = 83, Test 2 = 83, and Test 3 = 83					
Client 1 BRT	Test 1 = 63, Test 2 = 62 , and Test 3 = 62					
Client 1 WRT	Test 1 = 100, Test 2 = 101, and Test 3 = 101					
Client 2 ART	Test 1 = 83, Test 2 = 83, and Test 3 = 83					
Client 2 BRT	Test 1 = 62, Test 2 = 63, and Test 3 = 62					
Client 2 WRT	Test 1 = 100, Test 2 = 101, and Test 3 = 102					

Table A.15: Best Median Selection with Two Clients, Dynamic Processing Behavior ON, and k=3

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	347	205	69	187	192
	T2	365	189	67	189	190
	T3	367	248	48	155	182
Proxy ART	T1	75	79	85	80	80
	T2	75	78	85	80	80
	T3	75	79	84	80	80
Proxy LRT	T1	98	90	97	97	97
	T2	97	87	97	97	98
	T3	96	96	95	95	95
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	98	96	97	97	97
	T2	97	96	97	97	98
	T3	96	96	95	95	95
Proxy LMRT	T1	97	89	96	96	96
	T2	96	86	96	96	97
	T3	95	95	94	94	94
WS ART	T1	28	26	23	29	31
	T2	28	25	22	29	30
	T3	28	25	22	28	30
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	45	39	29	41	43
	T2	42	35	27	41	44
	T3	42	36	27	39	41
Client 1 ART	Test 1 = 84, Test 2 = 84, and Test 3 = 84					
Client 1 BRT	Test 1 = 62, Test 2 = 63 , and Test 3 = 62					
Client 1 WRT	Test 1 = 105, Test 2 = 105, and Test 3 = 104					
Client 2 ART	Test 1 = 84, Test 2 = 84, and Test 3 = 84					
Client 2 BRT	Test 1 = 63, Test 2 = 62, and Test 3 = 63					
Client 2 WRT	Test 1 = 104, Test 2 = 104, and Test 3 = 102					

Table A.16: Parallel Selection with Two Clients and Dynamic Processing Behavior ON

Attribute	**T#**	**WS1(10)**	**WS2(12)**	**WS3(14)**	**WS4(16)**	**WS5(18)**
Num of Hits	T1	1000	1000	1000	1000	1000
	T2	1000	1000	1000	1000	1000
	T3	1000	1000	1000	1000	1000
Num of Wins	T1	904	13	0	81	2
	T2	911	21	2	63	3
	T3	872	13	0	115	0
Proxy ART	T1	127	134	144	128	130
	T2	127	134	144	128	130
	T3	127	135	144	129	131
Proxy LRT	T1	158	164	177	153	159
	T2	158	164	177	153	158
	T3	157	163	176	152	157
Proxy BRT	T1	58	66	77	69	69
	T2	58	66	77	69	69
	T3	58	66	77	69	69
Proxy WRT	T1	158	165	177	153	159
	T2	158	165	177	153	158
	T3	158	164	176	152	157
Proxy LMRT	T1	157	164	174	151	155
	T2	156	163	173	150	154
	T3	156	163	173	150	154
WS ART	T1	55	57	59	61	63
	T2	55	57	59	61	63
	T3	56	58	60	61	63
WS BRT	T1	16	18	20	22	24
	T2	16	18	20	22	24
	T3	16	18	20	22	24
WS WRT	T1	82	83	80	77	76
	T2	83	84	79	76	76
	T3	84	86	82	78	76
Client 1 ART	Test 1 = 137, Test 2 = 137, and Test 3 = 138					
Client 1 BRT	Test 1 = 63, Test 2 = 62 , and Test 3 = 66					
Client 1 WRT	Test 1 = 174, Test 2 = 177, and Test 3 = 173					
Client 2 ART	Test 1 = 137, Test 2 = 137, and Test 3 = 138					
Client 2 BRT	Test 1 = 62, Test 2 = 65, and Test 3 = 64					
Client 2 WRT	Test 1 = 177, Test 2 = 173, and Test 3 = 174					

Appendix B

Configuration Information

Table B.1: Simulator Configuration Setup - Table 1 of 5

Property Name	**Description**
chossingAlgorithmMethod	Specifies the choosing algorithm method. Possible Values are: 0: Random 2: Best Last 3: Best Median 4: Parallel
bestMedian.k	*k* factor value: The value of *k* is only used for Best Median algorithm and specifies the number of last responses that will be used to calculate the best median value. As an example if *k* is 3 then the last three response times are used to calculate the median response time
numberOfRows numberOfCols	Network Topology: A matrix of [numberOfRows] rows and [numberOfCols] columns. Example: numberOfRows = 5 and numberOfCols = 5 means a network of 25 routers in a rectangular graph of 5 rows and 5 columns
numberOfClients numberOfProxies numberOfWebServers	Specifies Number of Client, Proxy, and Web Server Agents

web.server.delay.x	Static processing time power of a Web Server Agent in which 'x' identifies the number of web server.
	
	Example:
	
	web.server.delay.1=10
	web.server.delay.2=12
	web.server.delay.3=14
	web.server.delay.4=16
	web.server.delay.5=18
	
	means that Web Server 1 has static processing time of 10ms. Web Server 2 has static processing time of 12ms and so on. The number of this properties must be equal to number of web servers defined for [numberOfWebServers] property.

Table B.2: Simulator Configuration Setup - Table 2 of 5

Property Name	**Description**
nameAddressMapper.client.x	Name to IPAddress Mapper for Client Agents. 'x' defines the client agent name. This property is used during creation and runtime to map a client agent name to its associated IP Address. Example: nameAddressMapper.client.C1 = 100.10.1.1 nameAddressMapper.client.C2 = 100.10.1.2 defines the IP Addresses for Client Agents C1 and C2. For each client agent we should have one property name of this type in configuration file.
nameAddressMapper.proxy.x	Name to IPAddress Mapper for Proxy Agents. 'x' defines the proxy agent name. This property is used during creation and runtime to map a proxy agent name to its associated IP Address. Example: nameAddressMapper.proxy.P1 = 110.10.1.1 defines the IP Address for Proxy Agent P1. For each proxy agent we should have one property name of this type in configuration file.
nameAddressMapper.router.x	Name to IPAddress Mapper for Router Agents. 'x' defines the router agent name. This property is used during creation and runtime to map a router agent name to its associated IP Address. Example: nameAddressMapper.router.R1= 120.11.1.1 defines the IP Address for Router Agent R1. For each router agent we must have one property name of this type in configuration file. In the conducted experiment we have 25 routers so 25 of these properties should exist in configuration file.

Table B.3: Simulator Configuration Setup - Table 3 of 5

Property Name	**Description**
nameAddressMapper.server.x	Name to IPAddress Mapper for Web Server Agents. 'x' defines the web server agent name. This property is used during creation and runtime to map a web server agent name to its associated IP Address. Example: nameAddressMapper.server.WS1 = 130.100.1.1 defines the IP Address for Web Server Agent WS1. For each web server agent we should have one property name of this type in configuration file.
router.delay.x	Static Processing Time Delay for each router agent. 'x' defines the router number. This property will be used by IA during RA creation and is used by RA during run time to calculate the router processing time when it receives a message. Example: router.delay.1= 2 set a value of 2ms for processing time to RA number 1. For each RA there should be one property name of this type in configuration file.
web.service.delay.x	Static Processing Time Delay for each WS. 'x' defines the WS number. This property will be used by IA during WSA creation and is used by WS during run time to calculate the server processing time when it receives a message. Example: web.service.delay.1= 5 sets a value of 5ms for processing time to WS number 1. For each WS (replica) instance there should be one property name of this type in configuration file.

Table B.4: Simulator Configuration Setup - Table 4 of 5

Property Name	**Description**
links.from.router.x	Defines all links started from router number 'x'. The link information for each router has the following format: links.from.router.S={PN,D,C,L,B}-{PN,D,C,L,B}-...-{PN,D,C,L,B} in which S: Source Router Index, a positive integer number PN: Name of Non-Host-Processing Node: Name of non-host-processing node. In this platform this could only be router that is shown as 'R' D: Destination Router Index, a number that specifies the destination router index C: Link cost that is a function of two subsequent fields (L and B) , a positive number L: Link Latency, measured by virtual millisecond, which is shown by 'ms', a positive number. B: Link Baud Rate, measured by bit/s. A positive integer Routers use the following formula to calculate the cost of Link shown as C. At this stage the L values are given as a static value at system start up. Routing tables are calculated and initialized based on L values: C = L + (100 / B) For simplicity C has been already calculated and put in property file. The values of L and B will be used by each Link Agent to provide a time based value that specifies the response time of link. Example: links.from.router.4={R, 5, 1, 0,1000}-{R, 8, 2, 1,1000}-{R, 10, 3, 1, 500} means the router R4 is connected to three other routers (R5, R8, R10) through three links. The link that connects R4 to R5 has a cost of 1, latency of 0ms, and baud rate of 1000 bits/s. Respectively the Link that connects R4 to R8 has cost of 2, latency of 1ms, and baud rate of 1000 bits/s. The last link that connects R4 to R10 has cost of 3, latency of 1ms and baud rate of 500 bits/s.

Table B.5: Simulator Configuration Setup - Table 5 of 5

Property Name	**Description**
router.Rx.routingInfo	Routing information from Router Rx to all destination host nodes. 'x' is the number of router. Example: router.R11.routingInfo=WS1:R7;WS2:R7; WS3:R7;WS4:R12;WS5:R12;P1:P1;C1:C1;C2:C2 means Router R11 should send any income messages to R7 if the receiver host is WS1, to R7 if the receiver host is WS2, to R7 if the receiver host is WS3, to R12 if the receiver host is WS4, to R12 if the receiver host is WS5, to P1 if the receiver host is P1, to C1 if the receiver host is C1, and to C2 if the receiver host is C3. For each router in the graph we need to have one entry like this. If there is no value assigned to this property that means that router is not connected to any node in the graph.
web.server.url.x	URL for each web server. 'x' is the number of web server
web.service.url.x	URL for each web service. 'x' is the number of web service.
client.delayBetweenEachWSCall.x	Delay value between each call that is sent from a client agent. 'x' is the number of client agent. Each client agent waits for the specified amount in ms between each calls. The default value is 5 ms and the value set in this experiment is 50 ms.

Appendix C

Runtime Environment

This experiment was executed on the following environment:

Hardware Overview	Model Name: MacBook Pro Processor Name: Intel Core 2 Duo Processor Speed: 2.53 GHz Number Of Processors: 1 Total Number Of Cores: 2 L2 Cache: 3 MB Memory: 4 GB Bus Speed: 1.07 GHz
Software Overview	Operating System: Mac OS X version 10.6.4 Java Runtime Version: Java(TM) SE Runtime Environment (build 1.6.0_20-b0227910M3065) JADE Version: Version 4.0

Bibliography

[1] Andrew S. Tanenbaum, Maarten Van Steen, *Distributed Systems: Principles and Paradigms.* PEARSON Prentice Hall, 2nd ed., 2007.

[2] James F. Kurose, Keith W. Ross, *Computer Networking, A Top-Down Approach.* Addison Wesley, 5 ed., 2010.

[3] Nikos Vlassi, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence.* Morgan & Claypool Publishers, 2007.

[4] Sattanathan Subamanian, "High-avaiable web service community," *Sixth International Conference on Information Technology: New Generation. IEEE Computer Society*, 2009.

[5] Zakaria Maamar, Quan Z. Ssheng, Djamal Benslimane, "Sustaining web services high-availability using communities," *The Third International Conference on availability, reliability and Security, IEEE Computer Society*, 2008.

[6] W. Kowalczyk, N. Vlassis, "Advances in neural information prcoessing systems 17," *MIT Press, Cambridge, MA*, pp. 713–720, 2005.

[7] A. L. Symeonnidis, P. A. Mitkas, ed., *Agent Intelligence Through Data Mining.* Springer, Berlin, 2006.

[8] V. Lesser, C. L. Ortiz Jr., M. Tambe, ed., *Distributed Sensor Networks: A Multiagent Perspective.* Kluwer Academic, Dodrecht, 2003.

[9] M. A. Paskin, C.E. Guestrin, J. McFadden, "A robust architecture for distributed inference in sensor networks," in *4th Int. Symp. on Information Prcoessing and Multiagent Systems, Los Angles, CA*, 2005.

[10] Maes, Pattie, "Artificial life meets entertainment: Life like autonomous agents," *Communications of the ACM, 38, 11, 108-114*, 1995.

[11] Michael Coen, "Sodabot: A software agent construction system," *Proceedings of the 1994 Conference on Information and Knowledge Management Workshop on Intelligent Proceedings of the 1994 Conference on Information and Knowledge Management Workshop on Intelligent Information Agents*, 1995.

[12] Wooldridge, Michael, Nicholas R. Jennings, "Agent theories, architectures, and languages: a survey," *Springer-Verlag, 1-22*, 1995.

[13] Brustoloni, Jose C., "Autonomous agents: Characterization and requirements," *Carnegie Mellon Technical Report CMU-CS-91-204, Pittsburgh: Carnegie Mellon University*, 1991.

[14] Stan Franklin, Art Graesser, "Is it an agent, or just a program?: A taxonomy for autonomous agents," *Proceedings of the Third International Workshop on Agent Theories Architectures, and Languages, Springer-Verlag*, 1996.

[15] Adelinde M. Uhrmacher, Danny Weyns, ed., *Multi-Agent Systems, Simulations and Applications.* CRC Press Taylor & Francis Group, 2009.

[16] Stuart Russel, Peter Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3rd ed., 2010.

[17] Gerhard Weiss, ed., *Multi Agent Systems, A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, 1999.

[18] Marco Conti, Enrico Gregori, Willy Lapenna, "Content delivery policies in replicated web services: Client-side vs. server-side," in *Cluster Computing*, vol. 8, pp. 47–60, Springer Science + Business Media, Inc., 2005.

[19] Nabor C. Mendonca, Jose Airton F. Silva, Ricardo O. Anido, "Client-side selection of replicated web services: An empirical assessment," *The Journal of Systems and Software 81*, vol. 81, no. 81, pp. 1346–1363, 2008.

[20] Kambiz Frounchi, Partheeban Chandrasekaran, Jawid Ibrahimi , Shikhharesh Majunmdar, Chung Hung Lung, Laura Serghi, "A QoS-Aware Web Service Replica Selection Framework For an Extranet," 2007.

[21] Fei Li, Fangchum Yang, Kai Shuangm Sen Su, "Peer-to-peer based qos registry architecture for web services," *IFIP International Federation for Information Processing*, pp. 133–138, 2007.

[22] Vladimir Stantchev, Miroslaw Laek, "Addressing web service performance by replication at the operating system level," in *The Third International Conference on Internet and Web Applications and Services*, 2008.

[23] Ziad Kobti, Ding Chen, "A multi-agent simulation platform for web service qos selection modeling," *CIMCA 2008, IAWTIC 2008, and ISE 2008, IEEE Computer Society*, 2008.

[24] Khashayar Habibi, Reza Ariaeinejad, Abdolreza Abhari, "Modeling and simulation of web service replica choosing algorithm," *SpringSim'09-Poster Sessions*, 2009.

[25] Mehdi Dastani, Rafael H. Bordini, Jurgen Dix, Amal El Fallah Segrouchni, ed., *Multi-Agent Programming Languages, Tools, and Applications.* Springer, 2009.

[26] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood, *developing multi-agent systems with JADE.* John Wiley & Sons, 2007.