Directory scalability in multi-agent based systems

Shahid Hussain, Hassan Shabbir
This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 40 weeks of full time studies.

Contact Information:
Author(s):

Shahid Hussain
Address: Folkparksvagen 18:03, Ronneby 37240, Sweden
E-mail: sh_hussain@hotmail.com

Hassan Shabbir
Address: House 376, Street 29, G-8/2 Islamabad, Pakistan
E-mail: mr.hassanshabbir@gmail.com

University advisor(s):
Peter Tröger
School of Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : +46 457 102 45
Simulation is one approach to analyze and model the real world complex problems. Multi-agent based systems provide a platform to develop simulations based on the concept of agent-oriented programming. In multi-agent systems, the local interaction between agents contributes to the emergence of the global phenomena by getting the result of the simulation runs.

In MABS systems, interaction is one common aspect for all agents to perform their tasks. To interact with each other the agents require yellow page services from the platform to search for other agents. As more and more agents perform searches on this yellow page directory, there is a decrease in the performance due to a central bottleneck.

In this thesis, we have investigated multiple solutions for this problem. The most promising solution is to integrate distributed shared memory with the directory systems.

With our proposed solution, empirical analysis shows a statistically significant increase in performance of the directory service. We expect this result to make a considerable contribution to the state of the art in multi-agent platforms.

Keywords: Agents, MABS, JADE, DSM, DF, Distributed shared memory, performance, scalability
ACKNOWLEDGEMENT

We would like to extend our sincere gratitude to our supervisor Dr. Peter Tröger for his timely guidance, and worthy criticism and for enormous support extended by him during the course of this research work. We are grateful to him for his support and appreciation throughout the thesis that helped us to achieve our goal and finish it in time.

We are also thankful to Mr. Dawit Mengistu for helping us to identify the research area. We also appreciate his efforts and positive feedback during whole thesis. Our friends and colleagues at Blekinge Institute of Technology also deserve thanks. Some of our friends contributed more than others especially we would like to admire the effort of Mr. Fahad Khalid for sparing time to review our thesis.

Our learning experience at Blekinge Institute of Technology has been very good. We are grateful to the faculty and staff at the institute. In addition, we want to appreciate the efforts and prayers of our parents who constantly motivated us during our stay in Sweden.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACC</td>
<td>Agent Communication Control</td>
</tr>
<tr>
<td>ACL</td>
<td>Agent Communication Language</td>
</tr>
<tr>
<td>AMS</td>
<td>Agent Management Service</td>
</tr>
<tr>
<td>CT</td>
<td>Container Table</td>
</tr>
<tr>
<td>DF</td>
<td>Directory Facilitator</td>
</tr>
<tr>
<td>DSM</td>
<td>Distributed Shared Memory</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation for Intelligent Physical Agents</td>
</tr>
<tr>
<td>GADT</td>
<td>Global Agent Descriptor Table</td>
</tr>
<tr>
<td>GUID</td>
<td>Global Unique Identity</td>
</tr>
<tr>
<td>IMTP</td>
<td>Internal Message Transport Protocol</td>
</tr>
<tr>
<td>JADE</td>
<td>Java Agent Development Environment</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>LADT</td>
<td>Local Agent Descriptor Table</td>
</tr>
<tr>
<td>MABS</td>
<td>Multi-Agent Based Simulations</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RTT</td>
<td>Round Trip Time</td>
</tr>
<tr>
<td>STM</td>
<td>Software Transactional Memory</td>
</tr>
</tbody>
</table>
Table of Contents

1 INTRODUCTION........................................................................................................................................... 1
  1.1 THESIS STRUCTURE .......................................................................................................................... 2

2 BACKGROUND ........................................................................................................................................... 3
  2.1 AGENT.............................................................................................................................. 3
  2.2 MULTI-AGENT SYSTEM – THE CONTEXT ................................................................................ 3
  2.3 SIGNIFICANCE OF MULTI-AGENT SYSTEMS ........................................................................ 4
  2.4 JADE AS MABS PLATFORM .................................................................................................. 4

3 PROBLEM STATEMENT ............................................................................................................................ 6
  3.1 AIMS .............................................................................................................................................. 6
  3.2 OBJECTIVES ............................................................................................................................. 6
  3.3 RESEARCH QUESTIONS ............................................................................................................ 7
  3.4 EXPECTED OUTCOMES ........................................................................................................... 7
  3.5 RESEARCH METHODOLOGY ...................................................................................................... 8
    3.5.1 Problem Analysis / Literature review - Qualitative method ........................................... 8
    3.5.2 Prototype development/Simulation - Quantitative method ............................................ 9
    3.5.3 Results/Conclusion - Implementation ........................................................................... 9

4 JADE ARCHITECTURE ANALYSIS ....................................................................................................... 10
  4.1 JADE ARCHITECTURE .................................................................................................................... 10
    4.1.1 Main Container ................................................................................................................ 11
    4.1.2 Directory Facilitator (DF)............................................................................................... 11
    4.1.3 Agent Management Service (AMS) ................................................................................ 11
    4.1.4 Container Table (CT) ...................................................................................................... 12
    4.1.5 Local Agent Descriptor Table (LADT) .......................................................................... 12
    4.1.6 Global Agent Descriptor Table (GADT) ......................................................................... 12
  4.2 AGENT COMMUNICATION LANGUAGE .................................................................................... 12
    4.2.1 KQML ............................................................................................................................. 13
    4.2.2 FIPA .................................................................................................................................. 13
  4.3 JADE MESSAGING SERVICE ........................................................................................................ 13
    4.3.1 Message Queue ............................................................................................................... 14
    4.3.2 Communication Interfaces ............................................................................................. 14
  4.4 AGENT TASKS EXECUTION MODEL IN JADE ............................................................................ 14
  4.5 RELATED WORK .......................................................................................................................... 15
  4.6 PROBLEM IDENTIFICATION - QUALITATIVE APPROACH ...................................................... 17
  4.7 EMPIRICAL ANALYSIS OF JADE PERFORMANCE .................................................................... 17

5 PROPOSED SOLUTION .............................................................................................................................. 19
  5.1 DESIGN APPROACH ...................................................................................................................... 19
    5.1.1 Solution #1: Shared Memory Approach ........................................................................ 19
    5.1.2 Solution #2: Distributed Shared Memory Approach ................................................... 21
  5.2 IMPLEMENTATION STRATEGY .................................................................................................... 23
    5.2.1 Implementation #1: Shared Memory Approach ............................................................. 23
    5.2.2 Implementation #2: Distributed Shared Memory Approach ........................................ 24
    5.2.3 JSTM Implementation .................................................................................................... 25

6 EXPERIMENTS ........................................................................................................................................ 26
  6.1 SETUP .......................................................................................................................................... 26
    6.1.1 Environment ...................................................................................................................... 26
    6.1.2 Variables ........................................................................................................................... 26
  6.2 EXPERIMENT: SCALABILITY ACCORDING TO NUMBER OF DIRECTORY ITEMS .................. 26
INTRODUCTION

The topic of this thesis is to investigate scalability and performance issues of the directory service in JADE (Java Agent Development Environment) platform. In JADE, an agent provides services to other agents so that these agents can perform different tasks to obtain some predefined goals.

For interaction, an agent uses protocols to communicate and coordinate with other agents. This protocol is known as the Agent Communication Language (ACL) [1][2]. There are multiple standards available from different organizations for agent communication protocols e.g. Foundation for Intelligent Physical Agents (FIPA) is one of them. FIPA founded in 1996, defines the standards concerning how a multi-agent platform should work. These standards enable agents from multiple vendors to collaborate and exchange information. JADE is one implementation of FIPA agent platform and complies 100% with the FIPA standards [3][21].

JADE provides a complete framework to develop agent-based applications, which follows the standard FIPA architecture [2][3][12]. The main components of JADE are agent management system, directory service and agent communication control. The platform itself is very stable and has incorporated number of features over time. The platform is also available as an extension for mobile devices, where the memory is too small [7]. Keeping in view all the relevant applications, the platform design is simple and robust. To acknowledge the dynamic nature of applications, the platform comes with built-in APIs to modify, update and extend the services. This helps researchers and the developers to perform a number of experiments using simulations in various fields.

In multi agent platform, agents update their knowledgebase by scanning the environment. Environment scanning is a process in which each agent seeks information about other agents from directory service and then waits for the results. At times, the delaying of results due to load on directory service would affect the platform performance.

Directory service in JADE works as a single threaded application and in case of multi-agent based simulation, the communication between directory service and the agents increases tremendously with an increase in the number of agents [2][8]. Multiple operations at a single time like communicating, parsing, searching and then relaying back the results to the agent causes degradation of service in the directory service agent.

There are a number of parameters available to do a comprehensive study about the platform. The goal of this thesis is to look into this matter and enhance the scalability and performance issues related to directory service agent. We have initiated the concept of implementing multiple directory service agents by introducing multiple approaches (shared memory approach and distributed shared memory) towards the scalability and performance boost in JADE platform.

One of the application domains, which could benefit from increased resource availability, is Multi-agent Based Simulations (MABS). Mr. Dawit Mengistu investigates this particular issue in an ongoing Ph.D. research at BTH. One future possible implementation for this is on TAPAS, i.e. the simulation for traffic.
1.1 Thesis structure

The next chapter covers background information and the fundamental concepts behind agents and their environments. Chapter 3 starts with problem domain that comprises of aims, objectives, research questions and the last section of this chapter covers the research methodology part. Chapter 4 provides the qualitative analysis about the JADE platform and the related material about JADE performance and directory service. Chapter 5 presents the proposed solution and implementation strategy. Chapter 6 describes the experiment designs and results. In chapter 7 we layout the conclusion and the future work to be done.
2 BACKGROUND

In this chapter, we provide background knowledge about agents and multi-agent based systems. We then move to the significance of multi-agent systems and finally discuss the JADE platform.

2.1 Agent

The term agent is getting popular in the software industry and is widely used in different horizontal and vertical markets in fields like artificial intelligence, operating systems, simulations and networks. The hidden power of an agent lies in its characteristic to mimic the behavior of complex real life systems such as societies, economies and other ecological systems.

Wooldridge and Jennings defines agent as

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." [1]

As given in the above definition, an agent acts at its own to achieve some goals. An agent makes decision either to cooperate with other agents, or to compete with each other to satisfy objectives. Each agent is social as it communicates with other agents, and it is reactive as it responds to other agents or to the environment in a timely manner. In addition, an agent is proactive as it makes decisions based on scanning the environment and performs tasks to achieve its aims without any external interventions. An agent can also learn to adapt itself to a new environment and work in a rational way to accomplish its tasks [1][2].

A multi-agent system (MAS) is a platform where agents interact with each other to achieve common goals. In a multi-agent environment, every agent satisfies its own goals as well as the collaborative global goals [1]. Examples of areas include web crawlers, e-commerce, intelligent transport systems, simulations, mobile devices, aerospace industry etc. [3].

2.2 Multi-agent system – The context

A multi-agent system is a software environment for deployment and execution of agents. It provides the infrastructure for the agents to coordinate with each other and to live like a community in that environment. A well-known definition given by Woolridge and Jennings about multi-agent system is:

"A set of agents that interacts in a common environment to achieve common or individual goals is called a multi-agent system (MAS)." [1]

In general, the services provided by such an environment are agent management services for registration, security profiling, directory service for lookups ontology keeping and remote monitoring of agents [2]. A Multi-agent system acts as a middleware between the operating system and the agents running on it and provides mobility to agents. In MAS, an agent has its own memory area and a set of defined protocols to communicate with the environment and with other agents. Agents use that protocol for message passing to communicate with other agents on the same platform, or with agents on other connected platforms.
The context is the real meaning behind multi-agent systems; the context where the agents are created and allowed to work. It is similar to the soccer field, where players are encapsulated to play football. The multi-agent systems provide that encapsulation for agents to be in relevant domains. However, if an agent deployed directly on the operating system, porting it to a different operating system would require significant changes (due to different underlying resource, access interfaces and mechanisms). Due to the fact, the available solution is to use an abstract layer like MAS to run agents on any operating systems.

2.3 Significance of multi-agent systems

Multi-agent systems are gaining popularity in many disciplines. People are thinking of changing their normal procedural systems with agent-based systems. The normal thoughts about agent systems are that they can change the living paradigm and provide more ease in the working environment of daily life.

The high demand in multi-agent systems has gained attention in the past decade, and a number of platforms have been developed to satisfy the needs of both academia and the industry [1]. These platforms provided a rich growth in agent based application in areas like auction systems, national hockey leagues, social science simulations, terminal container simulations, critical infrastructure platforms etc.

The promising results of agent-based systems are because of a number of real life problems, which is not feasible to model them mathematically. As a result, the best thing is to make an agent-based model of the problem domain and run a simulation multiple times by introducing multiple variables or by changing the roles of agents. At the end, the system provides results, which are beneficial for analyzing and forecasting real life problems.

2.4 JADE as MABS platform

In 1998, Telecom Italia (formally CSELT) developed a new multi-agent platform called JADE. The initial goal was to validate FIPA’s agent framework specifications and to provide a common platform, which would act as a middleware between agents and the operating system. JADE became open source in 2000 under the LGPL license that paved its way for researchers and organizations to take it further ahead [2].

A simple architecture and strong interaction design made JADE a good competitor in agent-based platforms. It initially evolved as a research tool for building agent based simulations, proofs of concepts and modeling complex problems. Due to its immense popularity among researchers as an open source agent platform, many people participated to add modules and add-ons to make it more robust and scalable for any type of agent-based application.

We selected JADE platform due to the following reasons as also specified by GT Nguyen et al [18].

Platform maturity

While considering other platforms we selected JADE as it is more mature and still well known for developing multi-agent based systems in the research community.
Openness

The platform is open source and comes with a huge built-in API collection that helps to develop agent applications in a fast-paced mode.

This study will help other researchers to increase the performance of simulations developed on the JADE platform.
3 Problem Statement

Now-a-days multi-agent platforms are gaining popularity in the development of simulation applications called Multi-agent Based Simulations (MABS) [9][10][11]. JADE is one such platform that provides rich interfaces to implement simulation applications. JADE provides a reliable and scalable platform to develop multi-agent based simulations [12].

The agents in the platform require information from the Directory Facilitator (DF) service agent about other agents [2][3]. At times the directory service agent takes a little longer time to respond to the agents’ requests due to multiple factors. Our focus is to study those factors and come up with a solution to enhance its performance.

3.1 Aims

Our aim is to study the JADE platform by analysing its performance and to look for problems in this domain.

3.2 Objectives

By reviewing the JADE layout in (3.1), we notice that DF agent serve all the agents of the platform and it stores information in a sequentially access memory. It might be possible that this service causes delay in the overall system performance.

The objective of this research is the scalability of directory service in the multi-agent platform, under consideration of different research questions, given in next section. To improve the functionality of directory service, we propose the concept of incorporating distributed shared memory and to improve the JADE architecture by incorporating more DF agents, which will improve its performance and scalability.
3.3 Research questions

- What are the performance bottlenecks in the directory facilitator agent in the JADE platform?

- What are the available options to enable the directory service to handle multiple requests?

- How can JADE-based applications transparently benefit from incorporating shared memory for the DF agent in order to increase its performance?

- What are the relevant performance metrics to rate the difference between local and distributed JADE execution?

3.4 Expected outcomes

The goal of this master thesis is to gather knowledge from the available literature and then transform it into the development of an architectural and prototypical JADE extension for running applications in the new environment. This includes the evaluation of related work, the development of resource usage strategies and the consideration of related distributed shared memory as blackboard agent or the replacement of DF agent. Experimental evaluation and measurements based on the identified performance metrics will validate the proof of concept.
3.5 Research Methodology

Our research is based on qualitative and quantitative research methods. We have carried out empirical analysis in the form of experiments in order to validate the effectiveness of the proposed solutions. The research methodology consists of following steps as shown in fig (3.1).

- **Step involved**
  1. Problem analysis
  2. Literature review
  3. Problem identification
  4. Prototype development
  5. Implementation
     a. Single platform
     b. Multi platforms
  6. Experiments and Results

3.5.1 Problem Analysis / Literature review - Qualitative method

The study of literature provides us with the basic knowledge of agents and multi-agent based systems. The main idea is to gain knowledge about the JADE architecture as a multi-agent platform. Other thoughts are to look into the messaging mechanism and the DF functions, and finally to investigate what are the performance bottlenecks and how can we address those issues to improve the performance and the scalability of the platform.
3.5.2 Prototype development/Simulation - Quantitative method

To validate and testify our results we proceeded with a number of tests in single and multi-platform environments. The results can be certifiable by a number of iterations and by changing different variables that give us confidence for the proof of our study.

3.5.3 Results/Conclusion - Implementation

The developed applications will help us in providing the answer to the questions, whether there are any significant performance differences in the JADE DF service with incorporating our design approach. To achieve our goals, we conducted extensive experiments by combining JADE and distributed shared memory. The results are collected and we used statistical characteristics of the samples to establish the significance of our findings (in terms of differences).
This chapter will elaborate the core concepts about the JADE architecture, agent communication language, JADE messaging, agent task model and related work done in the concerned area. The topics provide us the base for qualitative analysis of our thesis and guide us in understanding the problem domain.

4.1 JADE Architecture

JADE is a complete distributed agent platform with each agent running in a separate thread having the capability of transparently communicating with-in the platform or with other available platforms. Peer-to-peer communication among agents outperforms many other similar platforms based on its scalability and architecture [12].

The rich APIs facilitate developers to develop simple agent prototype applications as well as complex agent negotiations. Its built-in support for ontology and content languages provides a transparent mechanism of converting data into XML or RDF format.

Internally, the JADE platform is a collection of agent containers that can also work in distributed environment over the network. Agents live in containers that provide each agent a run-time environment and services for performing different tasks. Each container also provides a local directory look-up service and a communication mechanism to send/receive messages to/from agents.

Fig 4.1 gives a detailed overview of the JADE platform architecture. A separate sub-section provides detail about each component.
4.1.1 Main Container

The main container acts as the front-end interface for any agent entering into the platform and provides the bootstrap point for the platform. It is the first container launched in the platform and all other containers must join the main-container by registering with it [2].

Main Container hosts the following sub-services.

- Agent Management Service (AMS)
- Directory Facilitator (DF)
- Container Table (CT)
- Local Agent Descriptor Table (LADT)
- Global Agent Descriptor Table (GADT)

The normal naming convention used for other containers is like “Container-1”, “Container-2”, unless specified to some other name programmatically or by using command line parameters.

Main container is a centralized point of contact for all agents and this infrastructure lacks fault tolerance, failure in this service causes abnormal behavior for the platform [6]. To overcome this problem JADE platform supports replication services for the main container. In that scenario, the replicated main-container takes control in case of failure [2]. This replication provides the system with fault tolerance capabilities but the system still lacks load balancing.

4.1.2 Directory Facilitator (DF)

DF agent provides directory facilitator service. It resides in the main container and starts with the bootstrap of JADE. It provides yellow page services to all the agents of the platform.

The yellow page service allows agents to publish the description of the services they are offering to other agents. This helps the other agents to search and locate them and ask for further processing of tasks. Any agent can register as publisher (services provider) or discoverer (search for services). Registration, deregistration, modification and searches can be performed at any time during an agent’s lifetime [2].

DF uses ACL messaging style communication to coordinate with other agent of the platform. The internal architecture of the directory service uses sequential search mechanism for retrieving information from the memory storage [8][17].

4.1.3 Agent Management Service (AMS)

AMS is the core of JADE and runs in the main container of the platform. It automatically starts with the platform and provides different services to the agents for using the platform. It primarily manages the registration and the life cycle of all the agents registered at the platform [2].

AMS act as a supervisory authority by performing platform management operations such as creating and deleting the agent from the platform, deleting containers and shutting down the platform [2]. Other agents send request to AMS to perform agent management functions. AMS provides a naming service to make sure that each agent on the platform has a unique name in order to make the agent globally unique in any distributed environment [5]. AMS also assigns a unique identity to each agent named as Global Unique Identity (GUID).
The “white pages” service is the core of the JADE, where the platform maintains the complete information about each agent present on the platform. The service updates the information whenever agent creation or termination takes place. The AMS also maintains other connected platforms information to ensure proper handling of external entities for the messaging service [12].

AMS also offers on demand subscription services to other agent for notification of events like creation/deletion of agents and for container’s event notifications.

4.1.4 Container Table (CT)

Container table contains object references and transport addresses of all containers, which are part of the platform. The platform uses it to for locating the containers and the associated information. It is a tightly integrated with the main container and is only available for internal working purposes [2].

4.1.5 Local Agent Descriptor Table (LADT)

This table contains information about the agent in the local container, so if the agent sends a message to another agent, LADT is the primary source for searching agents in the platform, if it fails then the request is forwarded to GADT. The LADT comprises of fast searching algorithms to maintain efficiency in locating the agent [2].

4.1.6 Global Agent Descriptor Table (GADT)

This table acts as a registry for all agents available to the platform. The table consists of agent status, location information and it permanently resides in the main container. JADE uses a local cache mechanism where each container keeps a copy of the GADT with it, which synchronizes with the GADT in case of a miss. The cache replacement uses the Least Recently Used (LRU) algorithm that optimizes the cache performance [2].

4.2 Agent Communication Language

The agents require some common mechanism to communicate and coordinate with the other agents in order to perform their work. Normally there are methods available such as RMI, CORBA and .Net-Remoting, but they are the means of communication. Even though all these are ways for interaction, they lack the interaction protocol. To cater this, different organizations developed different ACLs that act as communication protocols for the agents to interact with external entities. ACL provides types of messages and their meaning to an agent. The message can be as simple as a string and as complex as rules, prepositions and semantics [2][5].

There are two known ACL platforms, which covers these areas. The first is KQML (Knowledge Query and Message Language) and the other standard is FIPA (Foundation of Intelligent Physical Agent).
4.2.1 KQML

KQML is like an object wrapper class, where each message start with the name and has a number of parameters sub-divided into attributes and values, similar to the XML structure, normally the syntax looks like the following example given by [1]:

Request
```
( ask-one
  : sender customer
  : content (PRICE IBM? price)
  : receiver stock-server
  : language LPROLOG
  : ontology NYSE-TICKS
)
```

This is an example of message packet format for inter agent communication. The conversation is in request/response protocol format.

4.2.2 FIPA

FIPA developed standard for agent interaction and also provided the basic framework for a multi-agent system, providing communication protocols, ontology and services for the scalability of multi-agent systems [2][4]. In addition to the old syntax of KQML, FIPA added action type and reserved actions for the agents’ registration and life cycle tasks [21]. JADE is using FIPA as ACL on intra-platform and inter-platform agent communication.

Example of the FIPA message is presented below:
```
(inform
  : sender agent-1
  : receiver agent-2
  : content (price dell6400 450)
  : language s1
  : ontology ebay-auction
)
```

The message starts from the word “inform” which tells the other agent about the action to be performed, rest of the message contains variables and their values. FIPA artifact action into a term called “performative” which is standardized for cross platform compatibility and understanding [21]. FIPA also provides a set of architectural framework for developing agent platforms [3].

4.3 JADE Messaging Service

The JADE messaging is the core of the platform; the platform uses queue mechanism for proper management of messages and uses multiple communication layers for sending and receiving messages.
4.3.1 Message Queue

The platform provides messaging queue to handle messages. The queue maintains the incoming messages, so an agent can get its messages from the queue. After processing each message, an agent can read the next message from the queue and performs the task, hence, this process run in a cycle until the queue is empty or an agent terminates [2].

4.3.2 Communication Interfaces

JADE messaging service is divided into two major groups

- Intra-Platform communication
- Inter-Platform communication

Intra-platform communication

Intra-platform messaging is required when two or more agents communicate within the same platform. The JADE platform implements Internal Message Transport Protocol (IMTP) for such kind of message transfer. IMTP uses two types of methodologies for sending and receiving messages, depending on the placement of the agents [2].

- Event passing
- RMI

If the sender and receiver agents both are on the same container then IMTP uses event-passing mechanism for sending messages to the other agent. If the sender and receiver agents both are in different containers then IMTP uses Remote Method Invocation (RMI) for sending messages to the receiver agent. Both the mechanisms are well tested, robust and give scalability and performance even in case of high communication load [12][13].

Inter-platform communication

Inter platform communication is performed by a service called Agent Communication Control (ACC). ACC provides this facility using application level protocols known as Message Transport Protocol (MTP). MTP provides multiple protocol choices for carrying message packets from one JADE platform to another platform. The most commonly used MTPs are HTTP, CORBA or IIOP. There are advantages and disadvantages of using each technique, but HTTP is preferred over rest of the protocols as it provides smooth and seamless communication over internet. CORBA and IIOP offers performance boost, if the platforms are directly connected with each other. The conclusion depends upon different case scenarios, but by looking at the real usage of internet based communication, the recommendation is to select HTTP MTP [2][13].

4.4 Agent Tasks Execution Model in JADE

An agent performs tasks within certain boundaries known as behaviors. These behaviors follow agent execution model. Agent execution model symbolizes control flow of the execution of an agent that mainly carries out in a thread [2]. An agent internal architecture uses single thread to process all its tasks as shown in the Fig (4.2).
An agent initializes its working environment in the setup method that runs only once in its lifetime. The developer can add as many behaviors in this method; each behavior executes depending on its priority or as defined by the developer of the agent. In each behavior, there is an action method, where an agent lists all actions to perform. After each action, an agent notifies the platform to do post action tasks. In case of agent termination, the last method known as “takedown” is used for agent cleanup tasks.

The architecture looks simple but provides a strong base for developing agents. An agent can define simple rules and actions within its framework to fabricate complex negotiation in simple and logical ways.

4.5 Related Work

This section will cover the related work done by others researchers in the area of JADE and multi-agent systems. During the last decade enhancement in MAS makes this field very pertinent to the research community. Each year the agent platforms are getting robust and reliable due to the increased demand in this technology. Few of the relevant papers are discussed below:


Fabio Bellifemine et al [3] presented a good understanding of JADE platform. The author briefly outlined the idea behind the development of JADE, the standards, FIPA framework, and the systems services provided by the platform. The choice of java language was based
on the following features that include object serialization, reflection API and RMI. This choice of language made the platform scalable for multiple operating systems. This paper provided a complete overview of JADE architecture and the core technology used in it.

**Fabio Bellifemine et al (2007) [12]**

Fabio et al (2007) [12] presented new JADE architectural framework and the modification done in the platform during the last nine years. The author introduced each component and its implementation, moreover the paper also provided with the information about different add-ons like web services, BDI component (JADEX), etc. and some performance metrics about JADE communication.

**Edward Curry et al (2003) [13]**

Edward Curry et al (2003) [13] implemented Java Messaging Service (JMS) as new message transport protocol (MTP) in JADE platform. This new MTP component improved the platform messaging performance and enrolled it into enterprise level agent based messaging platform. The approach provided the offline messaging and high volume of messages with reliable delivery of service. This paper also provided a comparison with other available MTPs like HTTP and IIOP, and concluded that this MTP will give the performance similar to HTTP, however by implementing JSM the platform can also work in offline messaging mode.

**E.Cortese et al (2001) [14]**

E.Cortese et al (2001) [14] presented simple approach to evaluate the scalability and performance of JADE message transport. The paper provided a description of messaging architecture, the caching techniques and the inter-platform message performance metrics based on Round Trip Time (RTT). The measurements were calculated and the results illustrated the linear graphs of the message transport layer in JADE platform that highlighted it as the future platform for multi-agent based system.

**Krzysztof Chmiel et al (2004) [15]**

Krzysztof Chmiel et al (2004) [15] provided extensive experimental result about JADE platform, the performance of JADE between agent-agent communication as well as the whole performance of the platform was scrutinized using different test cases on different operating system. The results supported the claims by E.Cortese et al (2001) [14]

**Kresimir Jurasovic et al (2006) [16]**

Kresimir Jurasovic et al (2006) [16] provided a general comparison of JADE platform with Grasshopper platform. The authors compared JADE with Grasshopper platform in term of performing an agent tasks like message sending and receiving, data payload comparison and an agent migration performance. The result showed that JADE performed better as compared to Grasshopper in sending and receiving messages and in data payload comparison. However, in an agent migration the average transfer size of an agent in JADE is 2.1 times more than that of Grasshopper, as Grasshopper used agent cache technique and only sends internal state of the object to migrate. In JADE, the whole object migration took more time and more processing as compared to Grasshopper platform.

Geon-Ha Lee et al (2008) [17] worked on the designing of intelligent directory service for an agent based service discovery. Using this approach the agents owns set of services and stored that information into their repository with tree based search mechanism. This paper described context-aware service matching by implementing context-filtration of data in order to segregate the data that optimized the search.

Dong-Uk Kim et al (2007) [8]

Dong-Uk Kim et al (2007) [8] presented improvement architecture to overcome directory bottleneck and the sequential search problem. The paper describes a way to use the content filters and inference engine to map user-predefined policies by implementing category tree structure. The result showed improved searching mechanism for agent based systems.

4.6 Problem Identification - Qualitative approach

By systematically reviewing related material, we arrive at a conclusion that JADE platform is well built and suitable for agent based application [12] [13] [14] [15]. However according to other related work [8][17], the following bottlenecks are well identified:

- Network Overloading
  o DF agent is single service on the platform, which handles "yellow page" information, when running an agent application where the number of agent increases to some hundred, the DF agent performance get degraded [8].
- Single threaded
  o Single task can be processes at a single point of time and the system needs parallel execution of task to handle multiple requests.
- Sequential memory storage access
  o The memory structure used by the DF agent uses sequential way to read, so every times the agent searches for some specific service, the DF has to traverse the whole array of all the register agent [2] [8].

4.7 Empirical analysis of JADE performance

Communication performance is one metric to gauge the efficiency of a multi-agent based platform. Multiple research studies were conducted to measures the performance of agent-based systems by changing different dependent and independent variables. The normal norm is to calculate the Round Trip Time (RTT) of messages between two agents [8][12]. The other parameters is to change the message packet size or by sending specific number of messages between two agents.

We also analyzed the scalability and performance of JADE by performing multiple experiments. To test the JADE platform scalability we used one master agent to submit tasks to all other worker agents. The worker agents perform their task and report to the master agent. The total execution time is calculated and by taking the mean value, the graph is plotted as shown in the fig 4.2.
The graph shows that JADE performs smoothly until we got the bottleneck after 430 agents, the JVM shows “out of memory error”, which further strengthen our facts that JADE performs well and is scalable enough to perform multi-agent based simulation. The only problem is with the hardware resources that can be upgraded depending upon the application requirements. Therefore, the only solution for scalable MABS is distribution, which is limited to directory service bottleneck.
5 PROPOSED SOLUTION

Multi-agent platforms have high communication-to-computational ratio [23], which causes degradation in service especially when large number of agents send their requests for specific information from a single agent. That will cause an agent to perform slower and this becomes a bottleneck or one point of failure in the platform.

One common solution is to hold the data to create a chunk of messages and then transfer it in bulk, but that will be a solution for non real time application where the application can allow delay in communication. The other solution is to localize the communication or improve the communication layer, by incorporating new communication protocols or changing the application flows. The third solution is to implement concurrency or parallelism in the architecture and we select concurrency model in our solution design.

We propose two design models for enhancing the performance and the scalability of DF service in JADE platform. To validate our design we come up with the prototype of our designs to get the results and see how distributed shared memory plays its role in validating our concept.

5.1 Design approach

We propose couple of solutions to handle this bottleneck and to lower down the communication vs. computational ratio. This will give us the performance boost in the platform as well as the scalability of DF service. The solutions are as follows:

- Shared Memory Approach
- Distributed Shared Memory Approach

5.1.1 Solution #1: Shared Memory Approach

The first approach is to create directory facilitator (DF) agent in every sub container. Each agent created in this sub container will register itself to the local DF agent, which is on the same container. DF agent uses a shared memory to shares the same information among all other DF agents created on the platform.

This will lead us to have concurrent threading model by placing DF agents into multiple containers, and the shared memory concept enhances the performance of DF agent. Each DF agent only focuses on individual write mechanism and it can read the whole memory of all agents. This approach minimizes the write operation for individual DF agent, as the local DF agent is responsible to store data about its own container agents. On the other hand, it can have the search space of all the agents residing in the platform as described in the Fig (6.1).

The worker agent send messages to DF agent via event passing and DF uses the same mechanism to send the data back to the worker agents.
Benefits in the design

The design provides multiple benefits in sense of network performance and memory consumption.

- The placement of DF agents in the sub container will lower the communication load as the presence of local DF service will allow other agents, to directly access with local DF agent without any message payload and RMI calls.

- The use of shared memory will allow concurrent DF agents to do read and write operations locally that will provide a virtual parallel threading model and will enhance the performance of the platform.

Risks

- Due to the shared memory, the agents that are residing on another machine can only access the agent list of their residing machine, not the list of all agents in the whole platform. However, this solution can work for single JADE platform running on single JVM.

- The other risk is of memory consistency, we have implemented locking strategy on write operations. When a thread executes a synchronized method for an object, the thread immediately acquires the object's lock before the execution and releases it after the execution, which makes impossible for the two invocations of synchronized methods on the same object to interleave [2]. As a result, all the other threads, which invoked the same synchronized method, have to suspend execution and they are moved to block state until the first thread releases the lock. The whole procedure ensures that at most only one thread can execute the synchronized method or block.
Secondly, when a synchronization method exits, it establishes a “happen-before” relationship with any subsequent invocation of a synchronized method which guarantees that changes to state of the object are visible to all the threads [2].

We address these risk factors in the next solution by incorporating distributed shared memory.

5.1.2 Solution #2: Distributed Shared Memory Approach

The second approach is to introduce the distributed shared memory concept. The need of transfer of data from one system to another created the concept of distributed application. In distributed systems, the data-passing model is the de-facto approach for exchanging and sharing information [20]. The exchanged information can be as small as a single character to as complex as live streaming of data from the imagery satellite. We see hundreds of application using this concept and providing huge amount of information.

The other approach is to use a shared memory model, which provides shared address space between all the connected applications. This approach comes from the concurrent and parallel architecture, where application programs uses this shared address space in the same way as they use normal local memory [19]. However, in software this approach requires an efficient and reliable communication mechanism. Now a day, the current network infrastructure can facilitate the applications with reliability and performance without any degradation in service.

A layer of software’s provides the shared address memory in distributed systems. There are two ways to make this type of software layer. One way is to modify the operating system kernel to provide this functionality and the other way is to write runtime library that provides seamless integration and the communication with the application [19]. This sort of shared memory model is known as Distributed Shared Memory (DSM).

Mostly the implementation of distributed shared address memory is transparent to the developer, so they do not have to focus on the internal working of it. The underlying layer also provides transaction based updates, as we used in databases. Therefore, in case of exception or failure, the object returns to their previous old state and notifies the application. This approach is widely accepted in concurrent programming as Software Transactional Memory (STM) [14]. STM is an attempt to simplify the parallel programming and provides robustness to the applications without much overhead.

We designed distributed architecture for incorporating DSM. We uses client server model, where we write a server agent that act as a replication agent between all DF agents. DF agent acts as a client and connects to replication server to synchronize with other DF agents as shown in the Fig 6.2.
Each DF agent acts as a client machine to the replication agent. At DF startup, the agent connects to the replication server and automatically gets the copy of already existing shared objects from the server. The local DF will get the same number of objects that are present in the server memory. The DSM package will take care of the communication and transactional problems and will notify the connected DF agents if any agent is registered on any other connected DF. However, in our case we have segregated the write operation within individual DF to their specific objects, so adding the same object is not possible in our solution, which solves the consistency problem.

In this approach, each DF agent will have to write to the local memory in the form of transaction. The server will update rest of all the connected DF agents automatically. Therefore, at every moment each DF agent has the copy of all the memory objects as a local copy. Moreover, if some problem occurs in any DF agent, it will only require a restart to get the whole memory information available locally to the local DF agent.

Benefits

This approach will provide fail-safe mechanism to the platform itself as the failure in DF agent will cause only a single container agent to stop getting information to and from the DF.
agent, whereas if the DF agent fails in JADE platform all the agents in the platform will show abrupt behavior.

5.2 Implementation Strategy

5.2.1 Implementation #1: Shared Memory Approach

For single platform we have proposed a solution for JADE in the form of Shared DF. We have developed our own class with the name of DF_Container. This class uses internal data structure to keep the records of agents registered with the Shared DF. We have also adopted a memory locking mechanism as described below:

Locking Mechanism in Shared Memory
Java programming language [24] has a built-in support for multi-threaded programming, in which, threads access “shared data” and “shared memory”. This form of communication is very efficient and leads to different types of errors like memory consistency errors (occur when multiple threads access shared data), Thread Interference errors (results from the inconsistent view of shared memory) [25]. To overcome these problems, we use Java class libraries to provide seamless synchronization mechanism to our application.

Class functions
We have developed our own functions to perform different operations in the shared DF. We have kept the signature of these functions same as that of the original JADE DF. They are given as below:

- Register agents with local DF Container
  static void Register(String name, String service);

- DeRegister agents and services from local DF
  static void DeRegister(String name);

- Search Agents on the basis of services
  static String[] Search(String service);

- Modify service of an Agent
  static void Modify(String name, String serviceToChange);

Configuration
In order to use shared DF in an application, user has to import the package of SharedMemory as shown below:

import SharedMemory.SharedDF.*;

For each container a local DF should be created. The local DF at each container can be enabled as shown below:

Method call to create DF at each container
AgentController localDF = ac.createNewAgent(NameOfLocalDF, SharedMemory.SharedDF.DF_Container.class.getName(), new Object[0]);

Method to start Agent
localDF.start();

By following these steps SharedDF can be used in an application.
5.2.2 Implementation #2: Distributed Shared Memory Approach

For distributed platform we have proposed a solution for JADE in the form of Distributed DF, which is based on XSTM framework [22]. The XSTM project is the combination of JSTM, NSTM and GWM. We have used JSTM as it is a Java implementation of XSTM, whereas rest are the implementation for .NET and web platform. Basically, XSTM is an open source library which enables high performance object replication between processes and is termed as an object oriented Distributed Shared Memory or a Distributed Object Cache [22]. In order to use it for our proposed solution, we have developed our own classes with the name of “JstmServer” and “DF_Container”. Here the mechanism is different from the implementation mechanism of Shared DF because for Distributed DF we maintain a local DF at each sub-container. So that every agent which is created on a specific container has to register itself with the local DF. This local DF then provides an interface, with set of functions, which are used to replicate objects. The JstmServer class runs the Server side code of JSTM whereas client side is managed by the DF_Container class. The DF_Container class uses internal data structure to keep the record of the Agents registered with the Distributed DF.

Class functions
We have developed our own functions to perform different operations on the shared DF that are given below:

- Register agents with local DF Container
  static void Register(String name, String service);

- DeRegister agents and services from local DF
  static void DeRegister(String name);

- Search Agents on the basis of services
  static String[] Search(String service);

- Modify service of an Agent
  static void Modify(String name, String serviceToChange);

Configuration
In order to use “DistributedDF”, JADE must be configured properly, then this class can be configured by importing the package of Replication into the application as shown in the statement below:

    import Replication.DistributedDF.*;

Then user has to enable “DistributedDF” by calling its server side and client side code for JSTM as mentioned below:

Server side code for “DistributedDF” must be enabled while creating Main-Container for JADE. It will be activated by creating and starting “JstmServer” agent as shown:

Method call to create JstmServer Agent
    AgentController jstm = ac.createNewAgent(NameOfServerAgent,
                                          Replication.DistributedDF.JstmServer.class.getName(), new Object[0]);

Method call to start agent
    jstm.start();
Then user has to maintain the local DF at each sub-container. The client side code will be run in each DF and it will act as a client when deals with the JSTM Server. When a new sub-container is created the local DF at the container must be enabled as shown:

Method call to create local DF at sub-container

```java
AgentController localDF = ac.createNewAgent(NameOfLocalDF,
Replication.DistributedDF.DF_Container.class.getName(), new Object[0]);
```

Method to start Agent

```java
localDF.start();
```

After these steps user will be able to replicate objects by calling above mentioned functions.

### 5.2.3 JSTM Implementation

JSTM is one such package that provides the implementation of software transactional memory in distributed way. JSTM provide seamless replication of object between multiple applications in distributed environment. The performance is very promising, as in JSTM, the object is transferred only once and onward it only uses ID or reference for update that save the communication overhead, therefore, it is much faster than RMI. RMI serializes whole object and transfer it to other machine at every update. The JSTM library claims double performance boost as compared to RMI implementation [22].

### 5.2.4 Others Software Transactional Memory Frameworks

There are many implementations of STM applications, which focus on the distributed aspect also. Here is the list of few with their salient properties as defined below.

TinySTM: TinySTM is a well robust application written is C++ and has implementation over multiple OS platforms; the performance measure based on transaction is very encouraging as mentioned by Pascal Felber et al [26]. However, the implementation is only for a single machine and not for a distributed environment.

DiSTM: DiSTM [27] is another distributed shared memory application that is considered for this experimentation. The package consist of JAVA code and claim well robust transactional features, however the package still lack some implementation details and is more focused on clustered computing.

TL2: TL2 [28] is another implementation of software transaction memory that outperforms many other implementations [26]. The code is written in C++ and has well documented APIs, however, the package is destined for single machine which make is difficult to adopt it for distributed platforms. Moreover, the interfacing between JAVA and C++ will also create an overhead in the experiments

The selection for JSTM is based on the JAVA language, JSTM APIs simplicity, and a well rich documentation provided by author. Moreover, the consideration is also based on the distributed nature and the scalability of platform choice will make it more suitable for our experimentation.
6 EXPERIMENTS

This chapter provides detail about the experiments conducted, the setup information and the environment for the experiments. To test the scalability of directory service we conducted two types of experiment.

- Scalability according to number of directory items
- Scalability according to agent numbers

6.1 SETUP

The following environment has been setup for the experiments.

6.1.1 Environment

We have setup a computer with windows XP and Java 1.5 as java runtime libraries. The system has the following configuration. We have close down all the running application in order to get an accurate measures, for more precision in our experiment we have taken three readings.

- Product: IBM Think Pad T43
- Processor: 1.8GHz (single core)
- RAM: 512MB

6.1.2 Variables

We have conducted the above experiment with the following variables configuration.

<table>
<thead>
<tr>
<th>Experiment Types</th>
<th>Independent variables</th>
<th>Controlled variables</th>
<th>Dependent variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability according to the directory search</td>
<td>Number of entries in DF</td>
<td>• Number of containers =2</td>
<td>RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of agents performing queries to DF =1</td>
<td></td>
</tr>
<tr>
<td>Scalability according to searching agent numbers</td>
<td>Number of agents performing queries to DF</td>
<td>• Number of containers =2</td>
<td>RTT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of entries in DF =50</td>
<td></td>
</tr>
</tbody>
</table>

6.2 Experiment: Scalability according to number of directory items

6.2.1 Experiment – Normal DF Search

DF agent resides on the main container and it keeps track of all the registered agents and their services. We create search agent in one sub-container and added working agent into this sub-container, and then create another container with some more agents added to this new container. The searching agent sends request to DF. DF returns the results to the searching agent. Time is recorded before the agent send message to DF agent and the receive time is recorder when the search agent received the results. The difference between send and
receive time is calculated, which is used as a measure for all the experiments. To show the linearity we started with 10 worker agents and one search agent. We measure by incrementing the 10 worker agents in each iteration and plotted a graph against the RTT value.

**Results**

The results present a linear graph as shown in the Fig (6.1)

![DF Performance Fig (6.1)]

The graph shows that the curve is closer to linearity and it verifies that DF service in JADE performs well, however, the amount of time taken increases with the number of agents. Therefore, if the agent gets the result of hundred searched items from DF agent the time taken will be around 90 milliseconds that is one of the causes of delay in multi-agent based simulation.

### 6.2.2 Experiment - Shared memory approach

The same experiment is conducted again with the same parameter, but with the new approach of incorporating shared memory.

**Results**

The results shows significant improvement over the original DF service of JADE as the total RTT value of 100 searched items is less than one millisecond.

![Shared memory Performance Fig (6.2)]
6.2.3 Experiment – Distributed Shared memory approach

The second solution is more focused to incorporate distributed shared memory concept, which has its own built-in communication requirements.

Results

Fig (6.3) elaborates the results, which show significant performance improvement as compared to original DF. These results will form the basis of our concept that incorporating DSM into multi-agent platform can add significant performance boost as well as make DF service more scalable for multi-platform applications.

![DSM Performance Fig (6.3)]

6.2.4 Performance Comparison Original DF vs. DSM vs. Shared memory

Fig (6.4) presents the performance metrics of all the technologies used. The significance is visible and the results are very promising to introduce distributed shared memory in multi-agent based systems.

![Performance Comparison DF vs. DSM vs. Shared memory Fig (6.4)]
6.2.5 Experiment – Performance metrics on Single vs. multi platform

We have done multiple experiments on accessing the performance degradation of JADE in multi-platform environment. For multi-platform environment, we run two instances of JADE on the same machine, in which one act as a host and the other as remote client. HTTP is the default MTP used in this experiment.

Results – DF performance

![DF Performance on Single vs. Multi platform](image)

DSM performance comparison Fig (6.5)

The result in Fig (6.5) shows that the communication overhead plays important role in degrading the performance of the JADE DF service. This experiment conducted is on a single machine and the communication is dependent on the operating system and the system performance. There is no physical network (routers, switches, etc.) involved so we conclude that adding network resources will generate more performance degradation in the JADE DF service while running on a real distributed environment.

Results – DSM performance

![DSM Performance on Single vs. Multi platform](image)

DSM performance comparison Fig (6.6)

The graph in Fig (6.6) shows the same results as we have seen in the case of DF, however the difference is not that significant as we have seen in the case of DF. These results are substantial to conclude that the DSM performs better than the RMI implementation of JADE.

6.3 Experiment: Scalability according to searching agent numbers
In this experiment, we fixed the number of entries in the directory service. We keep on incrementing the value of searching agent to get the performance of directory service by RTT value.

6.3.1 Experiment – Normal DF Search

The experiment provides interesting results, where we see a smooth graph and when we reached the figure of 50 searched agents the platform shows JVM “out of memory” error as shown in the Fig (6.7).

6.3.2 Experiment - Shared memory approach

Integrating shared memory to the directory service yields very promising results, and we see significant performance boost in the directory service of the platform.

Result

The graph in Fig (6.8) shows linear behavior of the shared memory approach and the time taken is not so much significant. The RTT value shows that this architect approach yield better results for multi-agent based simulations, however, we got the same JVM error when we reached the limit of 70 agents.

6.3.3 Experiment – Distributed shared memory approach

We conducted the same experiment by integrating DSM into the directory service.
Results

The graph (6.9) shows a linear graph, which shows that the DSM implementation will perform well in load. However, when we increase the number of searching agents to 80; the platform shows the same JVM error, but the performance is noteworthy than the directory service of JADE.

6.3.4 Performance Comparison Original DF vs. DSM vs. Shared memory

To sum all the figure, we draw a graph of all three experiment and plotted it in logarithmic scale.

The values in the in Fig (6.10) highlight the empirical evidence that DSM approach is much suited for application like MABS and it will enhance the scalability of directory service in multi-agent based platforms.
7 CONCLUSION AND FUTURE WORK

Based on our experiment results we concluded that there is enough claim for incorporating distributed shared memory into JADE is an appropriate approach to get a platform for scalable MABS applications. We found significant performance boost that proofs our distributed approach.

7.1 Answers to research questions

- What are the performance bottlenecks in the directory facilitator agent in the JADE platform?

  This question was answered in section 4.6 and the validation is done by performing experiment (see section 6.3)

- What are the available options to enable DF service to handle multiple requests?

  We have answered this question by providing solutions in chapter 5 and validated it in our experiment.

- How can JADE-based applications transparently benefit from incorporating shared memory for the DF agent in order to increase its performance?

  This question was answered in section 6.2 and 6.3.

- What are the relevant performance metrics to rate the difference between local and distributed JADE execution?

  This question was answered in section 6.2.5.

7.2 Result Summary

We achieved following research results in alignment to the stated problem description:

- We performed detailed study of performance issues in JADE and an empirical investigation shows that the JADE platform is well suited for agent-based simulation.

- Our qualitative and quantitative review of JADE DF service supported our claim that the platform performance depends on the DF as central bottleneck with an increasing number of requests or agent entries.

- We provided multiple architectural solutions to overcome this problem. By empirical study, we presented the increase in performance of DF agent with our solutions.
7.3 Future work

The performance provided by DSM is very significant. However, there is a need to do some performance analysis based on the number of reads/writes in the local shared memory as well as in distributed shared memory.

The other possible future work is to improve the sequential search mechanism with either ‘dynamic search algorithm’ or data structures.

7.3.1 Improvements

- An improvement will be required to integrate this DSM into the already existing DF agent.
- The DF agent, we implemented is simple in design and there is a need to add more functionality into it to run a full-scale simulation application.
- We used only RTT variable as a performance measure in our thesis, but more investigation are required to gauge individual packet send/receive time and the coding/decoding of packet computational time of the platform.
- To improve the linearity of graphs; the ideal condition is to use operating system with no installed daemon software or any running background processes, so an improvement is to recreate the test environment in Linux of Unix operating systems.
- To make the system fault tolerant, fail-safe mechanism (clustered server architecture) is well-suggested argument for a future improvement.
REFERENCES


APPENDIX ‘A’

The graphs showed are generated by the following list of tables.

Agent Performance – For JADE

<table>
<thead>
<tr>
<th>Agent Num</th>
<th>Factorial</th>
<th>Agent Start Time</th>
<th>Agent End Time</th>
<th>Time Diff</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>10000</td>
<td>1208353026437</td>
<td>1208353040015</td>
<td>13578</td>
<td>678.90</td>
</tr>
<tr>
<td>31</td>
<td>10000</td>
<td>1208353825187</td>
<td>1208353845531</td>
<td>20344</td>
<td>678.13</td>
</tr>
<tr>
<td>41</td>
<td>10000</td>
<td>1208354017843</td>
<td>1208354044703</td>
<td>26860</td>
<td>671.50</td>
</tr>
<tr>
<td>51</td>
<td>10000</td>
<td>1208354604156</td>
<td>1208354638390</td>
<td>34234</td>
<td>684.68</td>
</tr>
<tr>
<td>61</td>
<td>10000</td>
<td>1208354868000</td>
<td>1208354909562</td>
<td>41562</td>
<td>692.70</td>
</tr>
<tr>
<td>71</td>
<td>10000</td>
<td>1208354994250</td>
<td>1208354997875</td>
<td>48625</td>
<td>694.64</td>
</tr>
<tr>
<td>81</td>
<td>10000</td>
<td>1208355038015</td>
<td>1208355093593</td>
<td>55578</td>
<td>694.73</td>
</tr>
<tr>
<td>91</td>
<td>10000</td>
<td>1208355239531</td>
<td>1208355305640</td>
<td>66109</td>
<td>734.54</td>
</tr>
<tr>
<td>101</td>
<td>10000</td>
<td>1208355418890</td>
<td>1208355489015</td>
<td>70125</td>
<td>731.25</td>
</tr>
<tr>
<td>151</td>
<td>10000</td>
<td>1208355720296</td>
<td>1208355829218</td>
<td>108922</td>
<td>726.15</td>
</tr>
<tr>
<td>201</td>
<td>10000</td>
<td>1208355902562</td>
<td>1208356048500</td>
<td>145938</td>
<td>729.69</td>
</tr>
<tr>
<td>301</td>
<td>10000</td>
<td>1208356468953</td>
<td>1208356688234</td>
<td>219281</td>
<td>730.94</td>
</tr>
<tr>
<td>351</td>
<td>10000</td>
<td>1208358380031</td>
<td>1208358640000</td>
<td>259969</td>
<td>742.77</td>
</tr>
<tr>
<td>401</td>
<td>10000</td>
<td>1208357608109</td>
<td>1208357908937</td>
<td>300828</td>
<td>752.07</td>
</tr>
<tr>
<td>415</td>
<td>10000</td>
<td>1208361479921</td>
<td>1208361831031</td>
<td>351110</td>
<td>848.09</td>
</tr>
<tr>
<td>420</td>
<td>10000</td>
<td>1208362426500</td>
<td>1208362814828</td>
<td>388328</td>
<td>926.80</td>
</tr>
<tr>
<td>426</td>
<td>10000</td>
<td>1208363131859</td>
<td>1208363719984</td>
<td>588125</td>
<td>1383.82</td>
</tr>
<tr>
<td>430</td>
<td>10000</td>
<td>1208359011765</td>
<td>1208359838453</td>
<td>826688</td>
<td>1927.01</td>
</tr>
</tbody>
</table>

Table for Fig 4.2

Experiment #1: DF Performance

<table>
<thead>
<tr>
<th>Number of Agent</th>
<th>Reading I</th>
<th>Reading II</th>
<th>Reading III</th>
<th>Median Value</th>
<th>Mean Value (nanosec)</th>
<th>Mean Value (milliseec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>28604474</td>
<td>37925770</td>
<td>31960766</td>
<td>31960766</td>
<td>32830337</td>
<td>32.83</td>
</tr>
<tr>
<td>20</td>
<td>45579536</td>
<td>35003331</td>
<td>27572498</td>
<td>35003331</td>
<td>36051788</td>
<td>36.05</td>
</tr>
<tr>
<td>30</td>
<td>50423727</td>
<td>36266620</td>
<td>45377276</td>
<td>45377276</td>
<td>44022541</td>
<td>44.02</td>
</tr>
<tr>
<td>40</td>
<td>65195717</td>
<td>51812730</td>
<td>49794876</td>
<td>51812730</td>
<td>55601108</td>
<td>55.60</td>
</tr>
<tr>
<td>50</td>
<td>66462079</td>
<td>64256211</td>
<td>49528641</td>
<td>64256211</td>
<td>60082310</td>
<td>60.08</td>
</tr>
<tr>
<td>60</td>
<td>87671199</td>
<td>58268858</td>
<td>61836072</td>
<td>61836072</td>
<td>69258710</td>
<td>69.26</td>
</tr>
<tr>
<td>70</td>
<td>76366003</td>
<td>65644808</td>
<td>66674675</td>
<td>66674675</td>
<td>69561829</td>
<td>69.56</td>
</tr>
<tr>
<td>80</td>
<td>83499159</td>
<td>83216442</td>
<td>81116734</td>
<td>83216442</td>
<td>82610778</td>
<td>82.61</td>
</tr>
<tr>
<td>90</td>
<td>98138730</td>
<td>81885268</td>
<td>82232798</td>
<td>82232798</td>
<td>87418932</td>
<td>87.42</td>
</tr>
<tr>
<td>100</td>
<td>92335199</td>
<td>85775706</td>
<td>90872164</td>
<td>90872164</td>
<td>89661023</td>
<td>89.66</td>
</tr>
</tbody>
</table>

Table for Fig 6.1
### Experiment #1: Shared Memory Performance

<table>
<thead>
<tr>
<th>Number of Agent</th>
<th>Reading I</th>
<th>Reading II</th>
<th>Reading III</th>
<th>Median Value</th>
<th>Mean Value (nanosec)</th>
<th>Mean Value (millisec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>14248</td>
<td>13969</td>
<td>14527</td>
<td>14248</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>16762</td>
<td>17041</td>
<td>15644</td>
<td>16762</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>26819</td>
<td>27099</td>
<td>27378</td>
<td>27099</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>31289</td>
<td>33524</td>
<td>30451</td>
<td>31289</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>37155</td>
<td>37435</td>
<td>37714</td>
<td>37435</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>45816</td>
<td>42184</td>
<td>42184</td>
<td>43395</td>
<td>0.04</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>49727</td>
<td>45816</td>
<td>46654</td>
<td>47399</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>50844</td>
<td>53917</td>
<td>50285</td>
<td>51682</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>56990</td>
<td>56432</td>
<td>56990</td>
<td>56804</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>61181</td>
<td>64812</td>
<td>61181</td>
<td>62391</td>
<td>0.06</td>
<td></td>
</tr>
</tbody>
</table>

Table for Fig 6.2

### Experiment #1: DSM Performance

<table>
<thead>
<tr>
<th>Number of Agent</th>
<th>Reading I</th>
<th>Reading II</th>
<th>Reading III</th>
<th>Median Value</th>
<th>Mean Value (nano sec)</th>
<th>Mean Value (millisec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>62857</td>
<td>62019</td>
<td>62299</td>
<td>62299</td>
<td>62392</td>
<td>0.06</td>
</tr>
<tr>
<td>20</td>
<td>83810</td>
<td>82692</td>
<td>83810</td>
<td>83810</td>
<td>83437</td>
<td>0.08</td>
</tr>
<tr>
<td>30</td>
<td>102527</td>
<td>103086</td>
<td>103645</td>
<td>103086</td>
<td>103086</td>
<td>0.10</td>
</tr>
<tr>
<td>40</td>
<td>122641</td>
<td>122641</td>
<td>129905</td>
<td>122641</td>
<td>125062</td>
<td>0.13</td>
</tr>
<tr>
<td>50</td>
<td>127111</td>
<td>126552</td>
<td>134654</td>
<td>127111</td>
<td>129439</td>
<td>0.13</td>
</tr>
<tr>
<td>60</td>
<td>130464</td>
<td>129067</td>
<td>129995</td>
<td>129920</td>
<td>129812</td>
<td>0.13</td>
</tr>
<tr>
<td>70</td>
<td>159796</td>
<td>159238</td>
<td>158958</td>
<td>159238</td>
<td>159331</td>
<td>0.16</td>
</tr>
<tr>
<td>80</td>
<td>168457</td>
<td>169295</td>
<td>168736</td>
<td>168736</td>
<td>168829</td>
<td>0.17</td>
</tr>
<tr>
<td>90</td>
<td>184101</td>
<td>182425</td>
<td>184102</td>
<td>184101</td>
<td>183543</td>
<td>0.18</td>
</tr>
<tr>
<td>100</td>
<td>189130</td>
<td>188292</td>
<td>189409</td>
<td>189130</td>
<td>188944</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Table for Fig 6.3
### Experiment #1: DF Performance on Single vs. Multi Platform

#### JADE DF Performance on Single and Multi platform

<table>
<thead>
<tr>
<th>Number of Searches</th>
<th>Single platform (nanoseconds)</th>
<th>Multi platform (nanoseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>32830337</td>
<td>35334497</td>
</tr>
<tr>
<td>20</td>
<td>36051788</td>
<td>38126827</td>
</tr>
<tr>
<td>30</td>
<td>44022541</td>
<td>48140196</td>
</tr>
<tr>
<td>40</td>
<td>55601108</td>
<td>63750060</td>
</tr>
<tr>
<td>50</td>
<td>60082310</td>
<td>69488321</td>
</tr>
<tr>
<td>60</td>
<td>69258710</td>
<td>76304313</td>
</tr>
<tr>
<td>70</td>
<td>69561829</td>
<td>86008526</td>
</tr>
<tr>
<td>80</td>
<td>82610778</td>
<td>95419566</td>
</tr>
<tr>
<td>90</td>
<td>87418932</td>
<td>105735714</td>
</tr>
<tr>
<td>100</td>
<td>89661023</td>
<td>117221481</td>
</tr>
</tbody>
</table>

*Table for Fig 6.5*

### Experiment #1: DSM Performance on Single vs. Multi Platform

#### DSM Performance on Single and Multi platform

<table>
<thead>
<tr>
<th>Number of Searches</th>
<th>Single platform (nanoseconds)</th>
<th>Multi platform (nanoseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>62392</td>
<td>64934</td>
</tr>
<tr>
<td>20</td>
<td>83437</td>
<td>84396</td>
</tr>
<tr>
<td>30</td>
<td>103086</td>
<td>106654</td>
</tr>
<tr>
<td>40</td>
<td>125062</td>
<td>147624</td>
</tr>
<tr>
<td>50</td>
<td>129439</td>
<td>164068</td>
</tr>
<tr>
<td>60</td>
<td>129812</td>
<td>171331</td>
</tr>
<tr>
<td>70</td>
<td>159331</td>
<td>182223</td>
</tr>
<tr>
<td>80</td>
<td>168829</td>
<td>207254</td>
</tr>
<tr>
<td>90</td>
<td>183543</td>
<td>230944</td>
</tr>
<tr>
<td>100</td>
<td>188944</td>
<td>254261</td>
</tr>
</tbody>
</table>

*Table for Fig 6.6*

### Experiment #2: DF Performance

#### DF Performance

<table>
<thead>
<tr>
<th>Number of Searching Agents</th>
<th>DF RTT value (nanosecond)</th>
<th>DF RTT value (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>84711269</td>
<td>84.711</td>
</tr>
<tr>
<td>10</td>
<td>121056497</td>
<td>121.056</td>
</tr>
<tr>
<td>20</td>
<td>136182280</td>
<td>136.182</td>
</tr>
<tr>
<td>30</td>
<td>203023792</td>
<td>203.024</td>
</tr>
<tr>
<td>40</td>
<td>208325438</td>
<td>208.325</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

*Table for Fig 6.7*
Experiment #2: Shared Memory Performance

<table>
<thead>
<tr>
<th>Number of Searching Agents</th>
<th>DF RTT value (nanosecond)</th>
<th>DF RTT value (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>25879</td>
<td>0.026</td>
</tr>
<tr>
<td>10</td>
<td>27109</td>
<td>0.027</td>
</tr>
<tr>
<td>20</td>
<td>27930</td>
<td>0.028</td>
</tr>
<tr>
<td>30</td>
<td>28314</td>
<td>0.028</td>
</tr>
<tr>
<td>40</td>
<td>29214</td>
<td>0.029</td>
</tr>
<tr>
<td>50</td>
<td>30209</td>
<td>0.030</td>
</tr>
<tr>
<td>60</td>
<td>32108</td>
<td>0.032</td>
</tr>
<tr>
<td>70</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table for Fig 6.8

Experiment #2: DSM Performance

<table>
<thead>
<tr>
<th>Number of Searching Agents</th>
<th>DF RTT value (nanosecond)</th>
<th>DF RTT value (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>123200</td>
<td>0.123</td>
</tr>
<tr>
<td>10</td>
<td>231370</td>
<td>0.231</td>
</tr>
<tr>
<td>20</td>
<td>295904</td>
<td>0.296</td>
</tr>
<tr>
<td>30</td>
<td>323064</td>
<td>0.323</td>
</tr>
<tr>
<td>40</td>
<td>384636</td>
<td>0.385</td>
</tr>
<tr>
<td>50</td>
<td>542421</td>
<td>0.542</td>
</tr>
<tr>
<td>60</td>
<td>591531</td>
<td>0.592</td>
</tr>
<tr>
<td>70</td>
<td>713972</td>
<td>0.714</td>
</tr>
<tr>
<td>80</td>
<td>0</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table for Fig 6.9