A Multi-agent Persistency Framework using Cloud-Based Agent Mirroring

By

Nauman Khalid

2011-NUST-MSCS-025

Supervisor

Dr. Peter Bloodsworth

NUST-SEECS

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science (MS CS)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(May 2015)

Approval

It is certified that the contents and form of the thesis entitled "**A Multi-agent Persistency Framework using Cloud-Based Agent Mirroring**" submitted by **Nauman Khalid** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Peter Bloodsworth**

Signature: ____________
Date: ________________

Committee Member 1: **Dr. Muhammad Sohail Iqbal**

Signature: ____________
Date: ____________

Committee Member 2: **Dr. Anis ur Rahman**

Signature: ____________
Date: ____________

Committee Member 3: **Mr. Shamyl Bin Mansoor**

Signature: ____________
Date: ____________

Abstract

This thesis introduces a flexible cloud-based persistency framework that can be used in a wide range of Multi-agent systems. The recent emergence of the Internet of Things has made it possible for us to monitor and manage our environment like never before. Small embedded devices can run an array of sensors and communicate with each other using mechanisms such as Bluetooth SMART. Alongside such hardware developments, intelligent data processing and control software also needs to be created. Multi-agent systems provide a well-tailored architecture for such challenges. They are however often limited in terms of persistency and scalability. In our work we have designed and implemented a new in memory agent persistency framework. In order to evaluate proposed approach two prototypes have been implemented, one using the proposed solution and the other using an established agents persistency environment. Experimental results confirm that our framework is more scalable than existing approaches whilst providing a similar level of persistency.

Certificate of Originality

Author Name: **Nauman Khalid**

Signature: ________________

Acknowledgment

Up and above everything all glory to **ALMIGHTY ALLAH**. The Beneficent, The most Merciful and Most Compassionate. It's a great blessing from Almighty Allah that gives me the health and strength to do this research work.

I would like to special thank the Supervisor Dr. Peter Bloodsworth for his guidance, useful input and availability throughout my research work. His support and encouragement was remark able and responded to my questions and queries so promptly. I have to appreciate his enthusiasm and instant review of my thesis which able me to complete my research work.

I would like to express my deep gratitude to Dr. Muhammad Sohail Iqbal, Dr. Anis ur Rahman and Mr. Shamyl Bin Mansoor for being my committee members. I would also like to thank Dr. Kamran Munir and Dr. Nauman Qureshi for assisting me during my time as their student.

I wish to thank my lovely parents who always stood by me in time of need. I also appreciate their love and complete support in completing my thesis work. I will also thanks to my brothers and sister for their ultimate motivation and support during all this time.

It gives me immense pleasure to thank my friends M. Iqbal Aziz, Usama Shah, Tariq Habib Afridi, Naeem Ahmad, Fahad Shah and Anayat Malik for sharing their knowledge. Their discussion and motivations were always useful for me. They motivated me during the tough time in thesis journey. I would like to thank everyone who is directly or indirectly contributed in my thesis work.

Nauman Khalid

Contents

List of Abbreviations

Abbreviations	Descriptions
JADE	JAVA Agent Development Framework
MAS	Multi-agent Systems
JDK	Java Development Kit
AMS	Agent Management System
MTP	Message Transport Protocol

List of Figures

List of Tables

Chapter 1

INTRODUCTION

In Multi-agent systems, several software agents work individually to solve a certain problem. Software agents are computer programs that work autonomously on behalf of their user. Agent-based systems has been introduced as new paradigm for designing and implementing software using agents that learn from environment, interact with each other and use that knowledge to work autonomously for their users across open and distributed environments. Multi-agent systems are useful in problem domains where computations, planning or decision support are needed autonomously without the need for the involvement of people.

The agents in Multi-agent systems can lie on several nodes on a network or over the internet and the move among nodes. Unlike a centralized system, Multi-agent systems comprise a group of intelligent software agents which are distributed across a network. Therefore a Multi-agent system does not suffer with; resource limitations, critical failures or single point of failure, resource allocation deadlocks or bottlenecks and improves computational efficiency, reliability, flexibility, robustness and performance efficiency of the overall system. This composition of Multi-agent systems makes them more vibrant and resource efficient but also increases complexity. Collaboration, maintenance and synchronization of agents on different devices or network are real challenges in these systems. In this modern era people are using applications in mobiles, personal computers and other devices to manage their lives. These applications are growing in complexity, and due to this dependence of human beings on the internet. The internet of things (IOT) has already produced 50 petabytes of data on the internet. According to the worlds leading information technology research and advisory company Gartner, there would be connectivity of approximately 26 billion IOT devices with internet by 2020. The result of this is the networks become heterogeneous where different devices will be connected to the internet for different purposes. This will lead to scalability problem and even manipulating data among such networks will be challenging using monolithic systems.

Network capacity and data volumes are growing everyday and robust systems with self-learning capabilities having parallelism are required. In these circumstances multi-agent systems are well suited because an agent has the ability to perform actions on behalf of the user. And if a single agent is mapped to a single user, this concept will result in a huge number of agents in the system which will put significant burden on the underlying agent platform. Agents will perform many tasks such as communication, migration, cloning and execution. They are running on different devices with different specification and configuration with respect to hardware, software and network. Agents are therefore more likely to unexpectedly crash. Managing agents which crash in the complex interrelated system is very challenging in this at present, it puts limitations on the take up of agent-based systems. The crash agent can be recreated, but it will be amnesic agent. It will not start its execution from where it had crashed. It will unable to understand the updated status of the system. For example, A agent is sorting array of thousand integers. Considering the case in which the agent has sorted hundred integers and crashed. If the new agent is created, it will start sorting the array form start. It would not know that hundred integers has been sorted. The detailed explanation will be given in chapter 2.

1.1 Research Areas

Multi-Agent Systems (MAS) emerged as a scientific area in artificial intelligence and distributed computing. These days it is popular in research domains includes aircraft maintenance, electronic book buying coalitions, military demining, wireless collaboration and communications, military logistics planning, supply-chain management, joint mission planning, financial portfolio management. All listed research domains are distributed where the agent resides on several computers and these agents communicates for sharing information. For example supply-chain management which needs to oversights the various entities from supplier to manufacture and manufacture to wholesales and eventually whole sellers to retailers. In case of multi-agent systems, the agents would intelligently work on behalf of each of them.

Agents are supposed to be on different devices and network. These agents can crash because of changes in the environment of any device and network

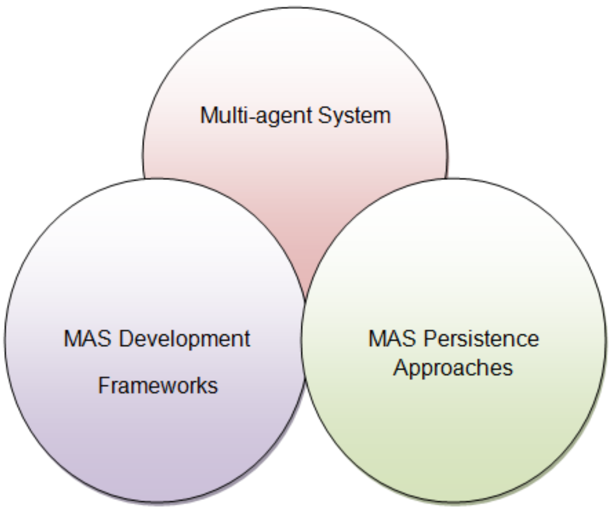


Figure 1.1: Research Areas

such as network failure, system shutdown. This will increase the complexity and make such systems more prone to failure. There is need therefore for a framework for multi-agent systems which is able to capture and persist agents states and recreate them when crashes occur. The proposed solution is a cloud base approach for persisting latest states of agents within an agent development platform. The complexity of applications has been increased and therefore there is need of introducing frameworks using new technologies.

1.2 Research Hypothesis

Multi-agent systems are very useful and reasonably resource efficient for developing applications in distributed environments, but complex as discussed in the previous section. The agents in these systems need to be very stable and responsive. Crashes of agents in multi-agent system will like result in the failure of entire system. A persistence framework is required to recreate crashed agents with syncronized states. There are several persistence frameworks have been introduced to persist agent states, but it is felt that they need improvement given recent advances in Cloud and other technologies. This thesis proposes a cloud based framework to address this persistence problem. The basic approach is based on a communicative system in which a group of mirror agents reside in the cloud and agents share their internal states with them. When an agent or several agents fail their mirror agents will recreate them whilst persisting and recreating their internal state information.

The hypothesis of this thesis is

It is possible to build a widely applicable persistence framework for Multi-agent systems that saves the internal states of agents and can recreate them if they die.

Previously proposed persistence frameworks have not become popular because of some drawbacks. These are mainly due to changes in technologies and networks. The technologies and networks are more advance and powerful. Also the nature of software applications is changed and they become more complex. Thus, there is need to put forward new approaches for persisting agents' states and recreating them when crash. In order to support the hypothesis some research questions are summarized below.

Question no 1

Which agent states exist and how can they be classified for Multi-agent systems?

An agent has several states having different representation. States actually make agents able to work intelligently and function in the external environment. There is a need to understand the semantics of these agent states and classify them.

Question no 2

How can we propose a flexible persistence framework for capturing and persisting states of agent at the application level?

Agent states are actually information that it holds which can be collected at application level. This information can be stored using any persistence approach. The flexible persistence framework can posibily offer.

Question no 3

What should be the behavior of multi-agent systems when several agents or several clonel agents die in system or agent states change frequently?

The persistence approach is proposed to increase the stability and dependability of Multi-agent systems. This can be achieved when system is capable of recreating crashed agent with synchronized states. This question can answer in chapter five.

Question no 4

What would be the performance and scalability of the proposed approach?

The proposed approach require more memory resources rather than computational resources, therefore it would has possibly good performance. Also proposed approach is communication oriented which theoretically proves that it would considerably scalable.

1.3 Proposed Approach

In Multi-agent systems, agents take tasks and execute them intelligently. These agents can be on same or different locations. This makes application boundary of Multi-agent systems based software larger involving more networks, devices and computers. This distributed nature of such systems increases the performance but in mean time there are more chances of losing agents which can eventually cause system failure. The approach should able to recreate agents with synchronized states that crash while executing their tasks. In order to recreate an agent with synchronized state there is need to persist agent states. These states of agents may change frequently or after long time. The frequent chaning of agent states makes it really challenging to persist them. The previously proposed approaches and traditional way of persisting data are unable to handle it. In this thesis a new flexible persistence approach is proposed to improve stability of multi-agent systems.

States of agent can be captured at the system level or at application level. Unfortunately, there is no Java virtual machine based agent framework that provides a state capturing feature at the system level. The state of agent can be captured on application level and several methods have been proposed. The previous techniques used for persisting agent states are not very beneficial and are discussed in chapter two in greater detail. Among several persistency approaches, one of them is storing and updating agent states in database. This allows system to recreate agent with same state if it dies. Using a database limit the scalability of Multi-agent systems and writing and reading from database reduces the performance and can cause serious limitation to scalability. Database systems are complex, difficult and timeconsuming to design. As database systems are usually centralized damage to or the crashing of database may impact entire system. They are slow and can crash in many scenarios. Such as when there are huge numbers of agents with frequently changing their states. Similarly other pervious systems also have such problems.

Somehow when crashes occur and certain agents die, system can recreate them but they would be possibly suffered with amnesia. The recovered agents would unable to know their position in overall system which can cause unexpected system results and performance. These agents can contribute in system when their states would synchronize with the states they had when crash occurred. Also the recovered agents need to know about the changes occur in overall system during their absence. Actually when these agents get crash, other agents in the system continue their execution. These tasks sometimes make changes in the environment of system. Therefore a persistency framework is required for synchronizing these states.

In this thesis a cloud based solution is proposed for capturing and restoring the agent states. Agent states will be stored in memory which will obviously give faster access to data. The in-memory read write operations are time effective but memory is volatile. States will be lost when agent crashes or system shutdown. Therefore an agent will share its states with another agent which will be called clone agent. The clone is a replica of the agent who keeps changing states of the respective agent. Clone agent can reside anywhere on same machine, cloud or other machine in same or different network. The agent will share its state via communication. The communication is faster relatively other options for persisting data because networks are fast and fairly reliable nowadays .

The agent or its clone can crash and proposed approach will able to get them back with latest states. Considering a case agent crashes, it will be recreated using states saved by clone agent. Similarly if clone agent dies, it will be recreated using states of its respective genuine agent. Communication protocol will use to share agent states between agent and its clone. This communication will be humanitarian where they will share information with each other as human do. This communication and in-memory persisting of agent states will significantly improve the performance of persistence approach. The in-depth discussion on architecture of proposed approach will be presented in chapter three.

1.4 Research Methodology

The research began with a literature review which helped to scope and to concretely identify the problem. It was apparent that current persistence frameworks for maintaining and synchronizing agent states have a number of limitations. Existing approaches that were proposed for making the internal states of agents persistent were analysed. The detailed discussion on previous approaches and their shortcoming is provided in chapter two. The internal structure of agent is complex because of its multi-threaded nature, communication and learning capabilities and independent execution. There is a possibility that agents states changes frequently, also sometime does not change often. Therefore states of the agent cannot persist using conventional approaches. The steps involve in research methodology is shown in figure 1.2

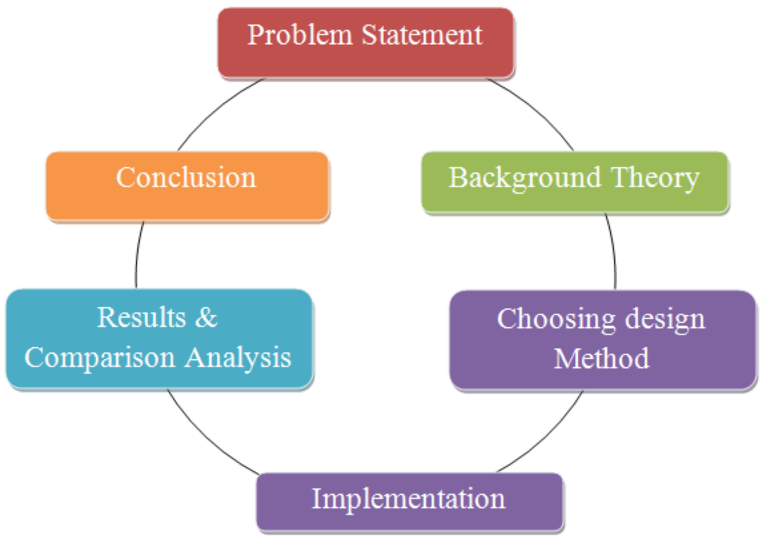


Figure 1.2: Steps in research methodology

After the study of shortcoming of previous persistence framework and their internal complexity, a new cloud based approach is designed to improve previous approaches. In order to evaluate this approach, it is compared with one of most recognized previous persistence frameworks. There were two prototypes developed, one used the old JADE persistence framework and second is implemented using the newly proposed approach. We chose the auction system because it is complex, real-time, agents comprehensively communicate for effective bidding, Also the number of agents can be increased to evaluate them in term of scalability. In the evaluation phase a number of metrics were used which included the response time of system, memory usage, recreation time and the expandability of the system. These are compared as the number of agents in the system increases.

1.5 Thesis Outline

Chapter two contains detail literature review providing background for hypothesis and it revealed the importance of research problem. The research problem is explained with precisely discussing proposed solution for best understanding of problem. The critical analysis of older approaches proposed for similar research problem through published facts, summary and comparison. The literature review showed that proposed work has not been done before and there is significant scope for research in this area.

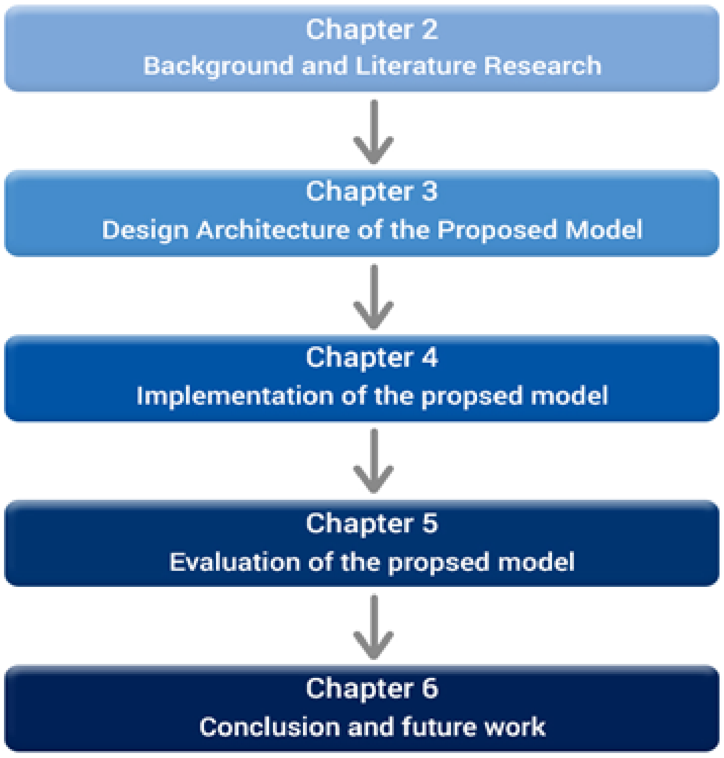

Chapter 2
Background and Literature Research
Chapter 3
Design Architecture of the Proposed Model
Chapter 4
Implementation of the propsed model
Chapter 5
Evaluation of the propsed model
Chapter 6
Conclusion and future work

Figure 1.3: Thesis outlines

Chapter three considers the design and architecture of proposed system.

There are two prototypes designed and implemented, one with older approach and other with proposed approach. The prototypes are actually auction systems in which older and proposed approaches are used to save internal states of agents. The chapter briefly present the criteria of selecting older approach and why auction is selected for developing prototypes. The finalized architectures for evaluation are concluding in this section.

Chapter four includes a consideration of the tools and systems which can be used for the implementation of the architecture proposed in chapter three. The implementation of each module of proposed architecture elaborated. It is ensured to explain with evidence that implementation of prototypes are according to the proposed architectures.

Chapter five presents the evaluation of both prototypes and their comparison. In order to better understand the results the experimental setup is elaborated. The test cases of qualitative tests are drawn and each test metric used to evaluate prototypes and their results are explained comprehensively. There is conclusion including the depth analysis of results.

Chapter six presents the future work directions. There are various possible research problems that can be carryout on basis of this thesis. It includes exciting applications that can used proposed persistency framework. The conclusion is also listed in end of this chapter.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

Chapter two presents an in-depth literature review for the research problem and related approaches that are being used for addressing it currently. The literature review considers previously proposed approaches and their weaknesses which provide an indication of the scope for research in this area. Test applications have also been developed to study certain previous approaches and analyse their shortcomings and performance. After this comprehensive study of previous approaches, a solution is proposed to overcome the weaknesses of current approaches. The chapter also has detail of relative frameworks and techniques are also discussed for better understanding of their drawbacks.

2.2 Multi-agent Systems

The usage of multi-agent systems for solving complex problems is increasing at a faster rate than ever before. This increase in their use demands more comprehensive and concrete development support for developers. Introducing new add-on, libraries, frameworks and approaches can be of significant help for developers to develop scalable and stable multi-agent systems. Multi-agent systems are complex comparative to other development approaches because of their internal multi-threaded nature, communication requirements complex behaviours, internal logics and distributed platforms.

An agent in a Multi-agent system is a thread and big systems include huge number of agents. Multi-threaded architectures are considered to be highly complex. Multi-agent systems have agents which can be on single machine or spread across network on several hosts. These distributed agents some time work collectively to perform certain task where they need strong coordination. This coordination can be achieved by using effective communication. In this scenario agents need to keep record of other platforms and agents working with them. This makes their internal state more complicated. More-over these agents can crash due to any disturbance in network host or channel, they will lose all their information. So far there is no trustworthy approach to recover crashed agent with all its information. Such crashes can degrade the overall performance of system and eventually result in crash of complete system.

2.2.1 States of Multi-agent systems

Multi-agent system could have many agents and these agents will probably have certain states. Such states can be captured either in system level or application level. The capturing of state at the system level will result in a very generic and concrete solution but it is really tough and it would need some comprehensive coding in the Java virtual machine (in case of JADE) or agent development framework. Another way of capturing and persisting agent states is at the application level where the states of system are captured at run-time. Multi-agent systems have agents which collaboratively work to complete certain tasks which mean they share their some states with other agent in multi-agent system; this sharing creates collaboration between agent which can be said multi-agent environment. This sharing requires the classification of states with respect to responsibilities of agent.

Bracciali et al proposed a model to classify states of agents of multi-agent systems. The model obtains the all possible states in multi-agent systems and broadly categorizes them in two sets, Environmental states and Agent states. Environmental states are states of agents which are shared with other agent in multi-agent systems while states of agents are specifically belong to that agent itself only. The model for modelling the agent state should be generic so in this paper agent is consider as black box which means states are model regardless of working of agent. The actions of agents are being observed to capture states and these actions also change the states of agent. These actions of certain agents are observed by other agents who in response take a specific course of action. Each agent maintains its mental state containing its beliefs, desires and what it intends to do. Further in the proposed model there are fully transparent Multi-agent system and partially transparent multi-agent systems. In transparent Multi-agent systems each agent knows the mental state of all other agents in system while in partial

Multi-agent systems each agent has certain limited visibility of mental state of other agent[10]. The work is significantly useful for classifying agent states. Although this process is very complicated, this increases the complexity of Multi-agent systems. Using this approach in ads on or frameworks will result in increasing complexity of systems. Similar approach with some modification can work notable well.

2.2.2 Multi-agent systems Frameworks

There are many agent development frameworks which are available for constructing Multi-agent systems. Popular agent development frameworks include AAA(Agent Anytime Anywhere), ABLE (Agent Building and Learning Environment), Cougaar [33], JADE [12], WADE [31], Sim-Agent[32]and many more. These frameworks provide an environment for developing Multi-agent systems, some of them are domain specific and some are generic. The problem is being discuss in here is integral part of all domains that are supposed to solve using Multi-agent systems. In broader scenario, focus is on distributed development where agents are on several devices having different states. Such systems require interpretability and platform independence. These agents supposed to communicate and move across different platforms. Considering these requirements, JADE is very preferable solution

JADE (Java Agent Development Framework) is a software framework which provides a very convenient means of developing multi-agent systems. It is implemented in JAVA following FIPA specifications which make it more standardized and several tools that provide debugging and deployment phase support. The multi-agent systems are supposed to be distributed across machines on different operating systems and hardware architectures. Remote GUI facilitates for configuring and managing agent platform across distributed machines even on runtime by stopping and creating agents, moving agents among platform, inspecting agent communication and analysing overall placement of agents (agents in each container). [12]

JADE is platform centric middleware that allows agents to share their services and resources on runtime. JADE platform also provide distributed environment with many run time containers. Each container can has several agents and each environment has one or more container. These containers provides run time environment to agents and developer can organize agents on bases of their behaviours and states among these containers. For example; Java Virtual Machine hosts several containers and agents are distributed among container according to application logic. The Main Container always launches before any other container in platform because it has FIPA services agents and all other containers register with main container. Main container gathers and manages the information of all other containers with in platform either they are running locally or remotely. [6][7][12]

JADE offers messaging based flexible and efficient communication architecture and manages queue of incoming ACL messages for each agent. Agent can access these ACL messages using different modes include blocking, polling, non-blocking and pattern matching based. It is based on FIPA communication model and all its components are integrated in JADE such as protocols, envelope, ACL, content languages, encoding schemes.The agent can communicate to agents in same container or in other container or even in other platform using these FIPA specifications however the underline process of JADE communication in different cases are shown in figure below

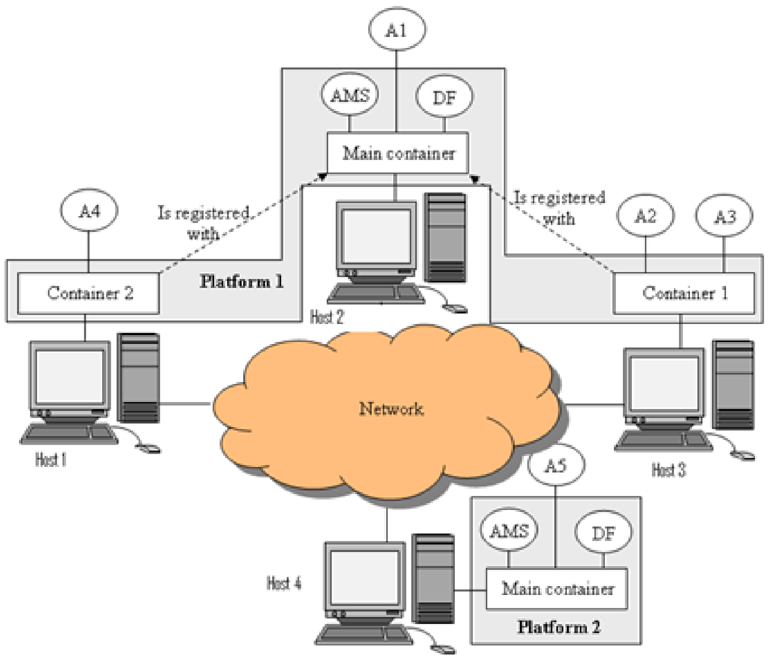


Figure 2.1: JADE based Agent Architecture [12]

2.3 Persistence approaches and Analysis

There are several approaches for addressing the persistency problem but persisting agents states are more complex. The states of an agent can change frequently and even in parallel under some conditions. Each agent or group of agents may have different states, hence managing this data (corresponding to the various agent states) is challenging. Approaches which are currently being used in order to address this problem include: Java Persistence API, Serialization mechanism, DBMS and JADE Persistence Services. These persistence frameworks are discussed in detail below along with their shortcomings. In chapter three the details proposed approach is explored which will overcome most of these issues.

2.3.1 Persistence using the Java Persistence API

The Java persistence API is a framework for developing business logic of enterprise application that edges the speed, security, and reliability of server-side technology [12]. It makes it easier for developers to manage relational data than ever before. The large number of supporting libraries in JPA enables them to develop applications in less time and with a significantly reduced effort. Although the Java persistency framework provides a structural way for persisting data in a relational database it is not necessarily always that scalable. When it comes to the management of frequent chaning agent states and if there are many agents operating in a system then this can become a serious issue with this approach.

Caire et al developed agent based mobile application for hybrid networks which are combination of several networks including GPRS, UMTS and WLan. They implemented an application in J2ME and used JADE as the agent development framework. They intensively checked scalability and stability of the application. They deployed application on network of wireless devices. In order to check scalability, interoperability and fault tolerance, the information in MTP table, Container tables and Global Agent Descriptor Table are evaluated. In terms of persistency they preferred the Hibernate framework [16] which provides object relational mapping (ORM). Hibernate has been proven to be able to successfully provide flexible and stable persistency because of its having support for the object query language. [9]

The Java persistency framework did improve the flexibility and stability of an agent-based system. Its scalability however was not satisfactory for agents who frequently change their state and thereby send huge numbers of requests to the persistency framework. Hibernate has the capability to perform secure and structural transactions but its performance degrades as the burden on it increases. It was added as add-on in the JADE platform for providing application level persistency but the developers have described that there is still need to improve this add-on for highly scalable agent based systems.[19] Simply efficient persistency mechanism is required.

2.3.2 Persistence using Serialization

Serialization is a mechanism for representing an object in the form of bytes which contain information about the object and the type of data that is stored. In serializing the object is cast to stream of bytes which can be read and write on any storage such as a disk, file, memory or database as shown in figure 2.2. In de-serializing, the byte stream is received as byte stream from where it was saved after serializing. This byte stream is cast as a respective object. As a serialized object is a byte stream this makes it independent from the underlying Java virtual machine. Therefore we can say that object serialized on one virtual machine can be de-serialized from any other machine which is helpful for exchanging data in distributed environments.

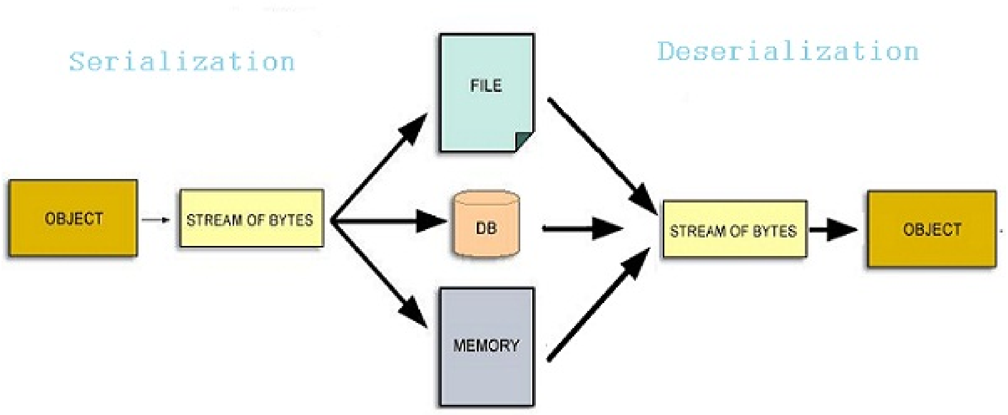


Figure 2.2: Serialization/ De-serialization Process of object [19]

Wong et al introduced a middleware application between the network and devices connected to the network called Concordia. This application was complete framework for developing and administrating mobile agent based application for accessing information from all device connected to network either they are wired or wireless. The framework allows agent administration, communication, migration among devices connected to network and persistency. The persistency framework in Concordia was able system to recover the agent and its states when it died. It uses Java object serialization for preserving the states of agent. The object having states of agent is serialized to preserve the agent and recover agent when it dies by de-serialized the object having its states.[20]

There are benefits and limitations to using any approach. Similarly serialization is good for preserving states for short durations and exchanging small data in distributed application among different modules. It is not used as complete persistence framework because preserving complex data is not possible. In serialization each object is preserved as single entity which does not have relationships with other serialized objects. In Java not all the objects are serializable, it should implement the serialized interface to be serialized. Also the reading and writing serialized object will result in a significant time overhead for frequently updating states. Due to above discussed reasons the use of serialization is not reliable for persisting updating states or saving states for the long term.

2.3.3 Persistency Using Databases

Databases are another way of persisting information. This is being used in different persisting approaches for Multi-agent systems. Relational databases save data in form on row and column in table. The data in tables are store in relations which actually create link between tables. A table is identified by field which is shared with other table to create relation this key is known as primary key. These relations help to retrieve recodes that are saved in more than one table. These relational databases are very popular because there are several database management systems that are available to interact with the data.

Databases can be used to store and persist agent states. States can be stored in a database which then allows them to be fetched, updated and deleted. Databases provide a very structural way for these operations. In the recent past, much work has be carried out on relational database management systems by extended them towards handling less structured data. The purpose of databases is to provide storage and increase in number of request to database consequences in slower database response.

DBMS are used in many enterprise business applications because they offer powerful information management tools. But for complex data, designing a database is not an easy task because it stores data in tabular form and single record could be in more than on table. Implementing relations among tables, primary keys, unique constrains, cascading are necessary for performances and consistency of database. Similarly Normalization of database is also a non-trivial process. It is hard for an inexperienced programmer to design database by implementing all these concepts.

Along with design complexities databases are also slow and their purpose is to store information. They receive the requests, process them and generate results accordingly. Results are sent to the requesting applications. This process of request handling to result generation takes time depending upon the structure and amount of data. In case of a database which has a complex schema most of the response time is taken by fetching data from many tables. Often increase in number of requests results in increase in the database response time. On receiving significant requests, it reduces response time and crashes especially in bulk operations.

If we consider the use of databases for persistency in a MAS there is a significant level of complexity. Each agent has its own internal state which may or may not different from the other agents in the environment. In a database schema these states will be attributes of table and agent states possibly change on changing the responsibilities of agent. Such responsibilities will increase or decrease numbers of states which will eventually change the schema of database which results in broken keys and records. This will actually increase the complexity for database because this cannot achieve using simple design. This complexity will increase with the passage of time and consequences in crashes and instability. Hence this change in responsibilities results in change of attributes of table, such updating attributes inconsistence the database and generates errors.

Another reason for not selecting databases as a persistency mechanism for multi-agent systems is their response times. Databases are used for storing data and their performance is hardware depended. Consider the case of a Multi-agent system which could have several thousands of agents distributed across a network. The agents in multi-agent system in some cases frequently change its states results in a massive amount of requests to the database. This response time of database can increase by enhancing hardware resources but it will be very costly.

2.3.4 JADE Persistence Services

JADE persistency services (JPS) has core module for providing persistency services known as JADE object manager. It is responsible for managing and persisting Java object using object oriented JADE database. It has full object oriented database management system (OODBMS) with full transaction control, multi-user object access, concurrency control with object-level locking, publish-and-subscribe events, and automatic object caching across distributed application servers. Java application can use all the features of JADE object manager using JAVA API.

The JPS divided persistency features into two broader classes. One deals with system level features such as whole containers and persistently storing agents, which has not been implemented thus far. The other deals with application level features related to the persistent storage of entities specific to applications. [2]. Neither the Java virtual machine nor any agent development framework offer a persistency framework for persisting agent states at the system level. Several application level persistency framework has been introduced using traditional persistency approaches, one of them is JADE persistency services.

The JPS are available as add-on of JADE which is not complete so it is not feasible for using in large applications. It does not support system level persistence yet, it is an add-on which eventually saves agent states in Database using an object relational model.[12] So far it does not support system level persistence and also application level persistency is not fully implemented and need some concrete implementation of some modules. It does not provide sufficient support so as to be used in enterprise multi-agent systems. The feedback of persistence services are not satisfactory and review of different forum showed that using persistency services is not a better choice.

2.4 Agent Mobility in JADE

Agents are capable of moving or copying themselves (Cloning) from one network host to any other network host. JADE also supports some degree of agent mobility using single method such as domove() or doclone(). The JADE migration or cloning of an agent is very simple but so far it only allows intra platform migration. In intra-platform migration agents can move among containers of a single JADE platform. This is a significant limitation because containers could be located on different physical machines across network. There is no proper library for migrating agents from one platform to another platform in JADE.

An agent has code and states where code is actually behaviour of agent and states are information of agent. These states shows current internal state of agent and it uses these states for decision making. Secure migration of agents is more challenging with code and states information. Migrating agents with code and states information could result in more content on a channel which will eventually result in latency and delay. The proposed approach for migrating agents from one platform to other is very effective because it is communication based and agents are good in learning from environment via communication. Also FIPA specification contains interaction protocols, content language representation, blocking and unblocking communication support which improves the communication capabilities of agents.

E.Cortese et al have carried work on the scalability and performance of the underlying message transport system of JADE. They designed JADE Message Architecture having multiple platforms and tested this design with a range of different scenarios. They tested communication performance between agents in the same container, agents in different containers but the same platform and agents on different platforms. They started with single sender and receiver and calculate the scalability of platform by increasing the sender and receiver parties. They observe the change in round trip time to draw their conclusions. The results state that the degradation of communication performance of design in inters and intra platform communication on increasing number of agents is negligible. Even use of IIOP and MTP protocols for communication among agents on different platform does not suffer in terms of performance. [7] The proposed approach in this thesis is communication oriented and could generate significant load on the communication aspects of the underlying multi-agent platform. The experiments that were carried out in this paper show JADE to be a scalable and efficient in communication performance for multi-agent systems with large numbers of agents. This is major reason for using JADE among many other reasons.

Ametller et al have presented a design and implementation which is capable of allowing agent migration over FIPA ACL messages. They proposed a model in which migration is directed by an agent but supervised by Agent Management System (AMS). Each platform has an AMS and they talk to each other for decision making for transferring agent or not. Agent that initiates migration should send request to other platform for migration along with the necessary information. The remote agent management system may reject the request or agree depending upon the capacity of its platform. The agent request can be refused when it does not meet to minimum requirements that platform specified for executing it. In case both platforms are agreed to migrate agent then code and data of agent is sent to remote AMS for execution. [6] This approach engages AMS for transferring which degrade the performance of entire system. The other tasks of AMS are also disturbed which actually makes this techniques not suitable for transferring agents. The approach proposed in this thesis for transferring agent is somehow different which is explained in chapter three.

2.5 Conclusion

A Multi-agent system is a way of developing more complex and distributed systems. A multi-agent system has agents who work on behalf of human on basis of information it gets from its surroundings. Also agents reside on different physical locations with different hardware software and network configuration. This increases the possibility of agents crashing and losing all its states information. This states information can help to recover agents that crash during execution with latest states. There is need of mechanism that can persist the latest states of agents and can recreate crashed agent using states information.

The analysis of previous approaches shows that there is significant space for research in this area. The previous approaches for solving this problem are not effective as discuss in previous section with analysis so there is demand of new approach. The changes of states of agents may be very frequent or may not change for ages depending on task of agents. Also dumb storage such as database is not good solution for agent based systems. The agents learn from environments like human being, it can negotiate and communicate so using dumb storage for saving its states would not suitable. Also variations in states of agents make it different from saving agent states information than other such information. In contrast of these issues there should responsive and collaboration approach where agents actively share their states.

This thesis intends to put forward an information sharing approach that can be implemented on any Multi-agent system. JADE is well known framework for developing distributed agent based system and it not domain specific as well. Many other Agent development frameworks are available but they are domain specific. The proposed approach is developed intended to be an applicable in wild range of Multi-agent systems. JADE follows FIPA specifications which make easy to understand the working of JADE and also FIPA gives very proficient control on communications. The proposed solution is cloud base where states of agent would save on cloud. JADE platform support all required support for running JADE containers on cloud.

Chapter 3

PROPOSED APPROACH

3.1 Introduction

The literature review in chapter two includes a study of previous approaches for persisting the agent states and recovering it when crash occur. It has been concluded that relatively few approaches that are being used at present. These approaches are not really stable and robust enough to use for enterprise applications. Some are conventional techniques for persisting data. The agent oriented framework that is introduced in this thesis is specifically for multi-agent systems which need to persist agent states. The architecture of the proposed approach is very comprehensive having several agents. The overall of working of approach is divided among these agents and they are classified with respect to these tasks.

In this chapter, the model of proposed approach is presented in details. This model is designed after study of shortcomings in previous approaches and try to introduce new improved approach. The chapter starts with analysis of literature review including list of constrains on developing persistence framework for persisting agent states in multi-agent systems. The next section is abstract model containing details of agents that work in proposed approach. After abstract model detailed architecture and its working is discusses. Chapter ends with discussion having answer to some research question related with model. Conclusion includes the overall behaviour and utilitarian aspects of proposed approach in multi-agent systems.

3.2 Literature Review Analysis

The literature review conducted in Chapter two and analysis of research questions revealed some important evidence to propose persistency framework for Multi-agent systems. They are discussed in end to interlink them with proposed approach. This section presents reasons for introducing this approach and elaborates key outcomes of literature review.

There is no complete framework is available yet that provides application level or system level persistence of agent states. Different persistence techniques are being used to save agent states. These techniques are not particularly for storing agent states and may therefore reduce the performance of multi-agent systems. JADE is java based agent development framework which introduced an ad on for persisting agent states known as JADE persistence services. Detailed study of this ad-on showed that it is not fully functioning and need concrete development of some modules.

Agents are composed of two entities code and states. Agents can be recreated with code and states. The code is uniform for similar set of agent while state may differ. The agent from synchronized state can be created in case its code and synchronized states are available. So only synchronized states are required to create agent and start its execution from the point where it died. The persistence framework for multi-agent systems only needs to store agent states to recreate them. Agents are intelligent and consistently learn from its environment. They are responsive so their internal state extensively depends upon their responsibilities and environment. The agent can change its state frequently or update them rarely. In this scenario traditional persistence technique fails as discussed in chapter two.

Multi-agent systems are collection of agents who collaborate to solve distributed problem. The complex problem is decomposed in set of small tasks which are further allocated to agents to solve. Each agent accomplishes its task and result of all agent are put together to get cumulative result. This all process is very communicative and agent has collaborative behaviour. Agent is multi-agent systems work similar to human beings and they are responsive. On these grounds they must have some humanitarian framework where persistence unit (who stores information) should also response similar like agents.

The multi-agent systems has autonomous agents, each agent has its states that are reflecting overall context of system. Suppose an agent is recovered with its states, but there are also some information it possibly needs from the environment. This information will be states of some other agents in the system. For example an agent is bidding for a book, it bids 40USD and crashes. The other agent will keep bidding, let another agent bids 60USD. When the crashed agent recovers with all its states back and starts bidding again. It may has all other information but it would probably not know the amount of latest bid for the book that is 60USD. Because this bid was made in the absence of the crashed agent. The crashed agent could become active part of Multi-agent systems if it get back its synchronized states along with concern information from environment. This requirement can answer the research question RQ-1.

Q1. Which agent states exist and how can they be classified for Multi-agent systems?

The states in multi-agent systems are classified in environmental states and local states. The environmental state information of MAS is accessible by all other agents of system while local states are internal states of agent. These internal states may or may not visible to other agents in system depending upon requirements of system. The environmental states will be maintained separately and other agents in multi-agent systems would keep synchronizing them. When any agent dies, it gets back latest environmental states sent to it when it is recreated, as other agents are supposed to synchronize these states.

In large multi-agent systems there are enormous agents and each agent has its several states. The states of agents have to associate with respective agent, this can easily map in Relational Database management system (RDMS). But RDMS is not dependable solution; chapter two contains complete discussion on limitation of using RDMS and other previous approaches for persisting agents information. This thesis proposes a humanitarian approach where agents communicate to maintain synchronized states. The agent actually the clone of original agent saves data in memory. The communication and mobility of agents are real challenges because huge number of agent will create massive communication messages. These messages will put burden on underlying Multi-agent development framework. The intent of developing prototype is to provide such framework where communication and mobility would abstract. The end developer should not need to access underlying working details of framework regarding communication and mobility. The detail discussion on proposed approach is listed in next section.

3.3 Abstract Model

Agents in Multi-agent systems may distribute across networks to perform different tasks. In such distributed environment agent can crash which will eventually affect overall performance of multi-agent systems. This thesis proposed the approach for recreating agents that crashes during performing their tasks. This approach acquires certain set of agents for stable and robust working and without decreasing performance. This section contains brief literature to describe the model of approach and running. The start of this section includes the overview of proposed approached which would continue with discussion of each set of agents on second half. The next section is describing architecture and complete working of approach.

The architecture of proposed approach included many agents. These agents and their states are classified in different groups. The agents are classified with respect to their responsibilities in multi-agent systems. The states of these agents are grouped in contrast of their scope. Scope is visibility of an agent states to other agent in multi-agent systems. Several agents work together to run complete architecture of proposed approach. These agents are discussed below including their responsibility and location in architecture of proposed approach. The next section contains their working and how all of them collaborate to run the system. In proposed multi-agent architecture these agents are named as; working agents, clone agents, Local monitoring agents and Remote monitoring agents

Working agents in multi-agent systems have certain responsibilities that contribute in accomplishing overall objective of system. These are autonomous agents that perform their tasks independently and share their information with other agent using communication. These tasks are divided among agents depending upon requirements of system. Each agent in system may have different task or set of agents is assigned single task, so multi-agent systems has agents with different internal states. In this thesis these agents are putted in one category although they have different internals. Because approach is proposed to restore these agents when they die so these are most crucial part of multi-agent systems.

The clone agents are mirror agents of their respective worker agents. Each worker agent has its clone which keeps internal states of respective agent. A worker agent could possibility of having one clone agent. Clone agent does not have its own internal states; it keeps synchronized states of its working agent. The worker agent and its clone agent could be in same platform or in different platform. In proposed approach we resided clone agents in different platform so that each platform can send and receive request of other platform such as client server architecture. This will help to analyse performance of communication between clone agent and its worker agent. The in-depth scenario of their communication and re-creation of these agents including clone and worker agent is discussed later in this section.

Local monitoring agent controls and supervise platform having worker agents. These agents may be one or more than one, depending on number of working agent in platform. These agents are responsible for creating died agents and managing working agents and allow agents to communicate with agent in other platform. When agent died local monitoring agent is responsible to create agent with synchronized states. The synchronized states are provided by respective clone agent of died agent. These agents also manage the environmental states of multi-agent systems and other information including memory, CPU utilization, and platform information. Platform information includes different addresses of platform, number of container and agents in these containers.

Remote monitoring Agent resides on platform having clone agents. These agents creates clone on request of worker agent. Remote Monitoring agent may be one or more than one depending on scalability of multi-agent systems. Like local monitoring agent it collects information from its platform and share with other platform. The agents involved in working of this approach are discussed so far. The internal of agent consists of code and states. The states of these agents change due to change in environment as agents learn from its environment. The other possibility of change in agent states is due to task performed by agent. In contrast of these reasons in our proposed approach the states of multi-agent systems are broadly classify into two types. These states are divided with respect to their visibility in multi-agent systems. The types of states are Environmental states and Local states.

Environment states include information which is visible to all other agent in multi-agent systems. It is responsibility of local monitoring agents to manage and monitors these states and recreates them if they are lost. All agents in multi-agent can change the information of these states to keep them synchronize. These states persists addresses of monitoring agent, application oriented states and monitoring information. The addresses of monitoring agent are stored to allow worker agent to request them for creating clone or recreating clone. Application oriented states have information which is being retrieved and updated by several agents. The updated status of MAS can be fetched using these environmental states and they are saved using blackboard and other data structures.

The other types of agent states in multi-agent systems are local states. These states of agent are only visible by agent itself. These states are private and are not being shared with any other agent in Multi-agent systems. These states represent the internal state of agent including information such as agent name, its container and platform information, agent identification addresses and application oriented states. These states are used to re-create died agent with synchronized internal states and environmental states are fetched later form local monitoring agent. Both environmental and local states are compulsory to recover died agent with its synchronized internal states and able it to opt in updating environment.

Just logical divisions of states of multi-agent systems are not enough to manage such unstructured data. States of agent is associated with respective agent or monitoring agents in case of environmental states using object encapsulation. These states are encapsulated in object which is serializeable having methods to access and manage the states. The ACL message only allow serialized object as content to transfer it between agents. Serialized object which contains either environmental or local state can be sent to any other agent in ACL message. Object having environmental states information will associate with monitoring agents and these agents will allow all other agents to get and change values of these states. On the other hand local states will not share with even monitoring agent. These states will associate with agent and other group who would have responsibility to manage them. Working agent will share these states to its clone agent which will use them to recreate agent when it dies. Considering the case where clone dies; the clone will be recreated using synchronized states of respective working agent.

The proposed approach has two platforms and agent in them frequently exchanges messages. Agents are transfer when agent or clone died. Considering inter-platform migration the agent development frameworks does not allow agents such migration. Inter-platform means migration between two separate and independent multi-agent systems. Similarly in JADE, two multi-agent systems running different JADE platform cannot transfer agents among them. These platforms are possibly on; same network, different network or over internet. The proposed approach is supposed to be efficient for transferring agents among platform in either of listed case. Chapter two includes previously proposed process of agents which is fallow here as well but with some alterations. The detail insight of agent mobility is important for understanding working of proposed approach.

Agent Mobility technique being used in this thesis does not transfer agent with code and states. The agents are even not transported through any communication channel. The literature review in chapter two proved that the agent can be recreated with code and state. The code is behavior of agent which does not change while states change. States are information of agent which can be share with agents in same or different platform using communication protocol. An agent in each platform supervises the mobility which has code of all type of agent running in platform. When a platform wants to transfer agent it sends synchronized states and type of agent (which is being transferred) to other platform as communication message. The other platform would create agent with synchronized state provided using code of given agent type in message.

This technique is used in proposed approach in this thesis where there are two platform connected over internet. Both platforms have monitoring agents that initiate agents transfer between platforms as shown in figure seven. Let supposed agent of platform A is being transferred to platform B. The agent ask monitoring agent for AID of monitoring agent in platform B. The agent will send request containing its states to platform B. The monitoring agent in platform A would create agent with state provided in message. The newly created agent in platform B is supposed to retain accomplishing task from same point where it left in previous platform.

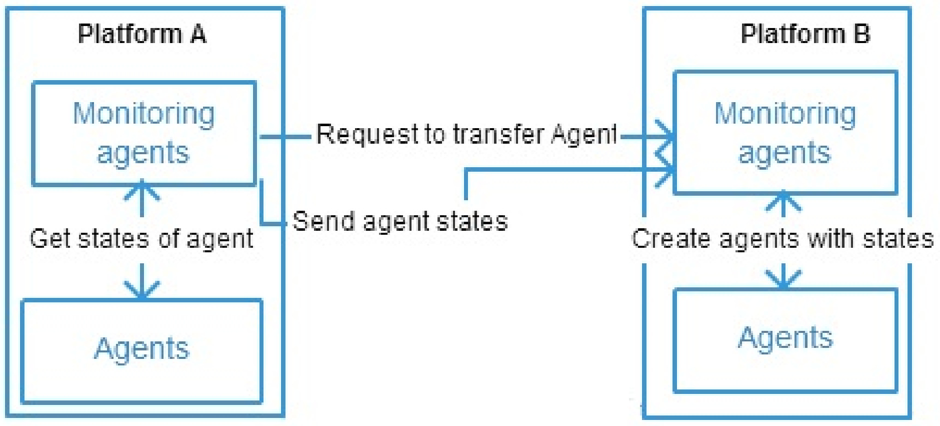


Figure 3.1: inter-platform mobility of agent

The challenge discussed in previous sections in mobility is changing states of agents and burden of request on agent development framework. The States of agent could frequently changes or may not change for ages depending upon domain and type of application. In chapter two we discuss on challenges using previously proposed approach for saving frequently updating states of agent. Old approaches does not handle this problem such as saving states in database slower response time and eventually crashes when agent sent sufficient updating requests. In proposed approach agent communicates with each other using ACL message which fallows FIPA specification. ACL messages do not put burden in underlying framework and they are good in communicating among agent on same platform or on different platforms . Also now a day networks are very faster so communication is not challenging.

The approach proposed in this thesis, each agent has its clone which can reside in same container, different containers or in different platforms. The recommended way of implementing this approach is to place agents and their clones on different platforms. And these platforms should be on different machines or networks; this will increase the robustness and stability of Multi-agent systems. The complete discussion on this scenario is discussed later in this chapter. So far we are considering two platforms one is local platform having agents named working agent and other is remote platform having clones of working agents named clone agents. These platforms could be on single machine or distributed over the network. The monitoring agents are responsible for managing overall platform including agent migration, agent or clone creation. These local and remote platforms are managed by local and remote monitoring agents respectively. These states of affairs will increases intra-platform agent mobility as well. JADE does not allow intra-platform migration or transfer of agent but agent will be transferred using approach discuss in chapter two.

The proposed approach has two JADE platforms; one platform is running on the local machine and other is running on cloud. These both platforms have monitoring agents which can be one or can scale up to depending on size of platform. The local monitoring agents maintain all the environmental states of local platform and make them available for all local agents. Along with these, local monitoring agents also allow local and remote platforms to communicate. Similarly the clones of agent which are running on cloud platform are maintained by Remote monitoring agents. They create clone of each agent in local platform on request of agent on local platform and synchronized the communication among agent and clone. It also fetches information of cloud usage such as instances currently running and number of agent in these instances and memory which is also shared local monitoring agent. This information helps local monitoring agent in managing the overall behavior of persistency system. The architecture of proposed approach is shown in figure 3.2.

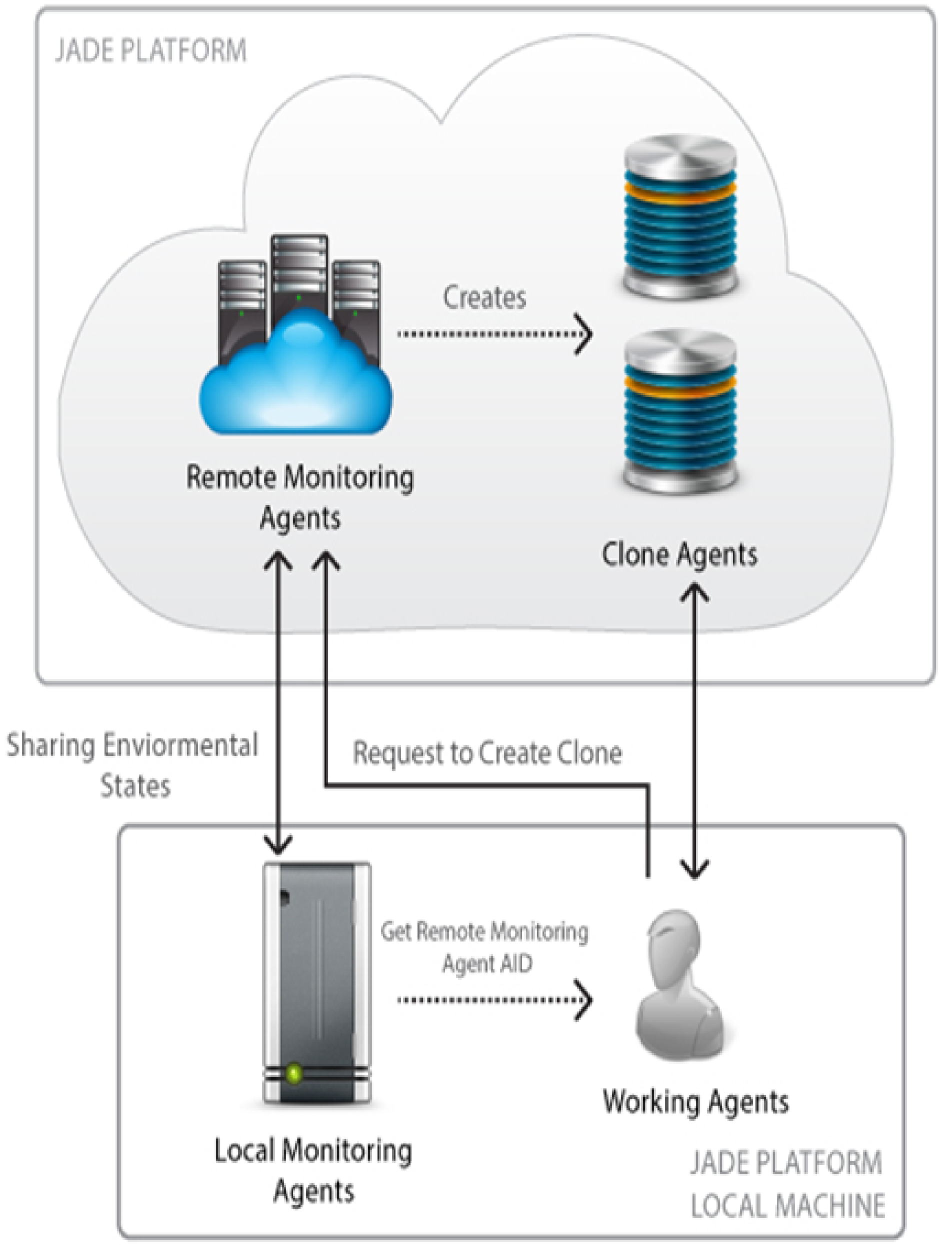


Figure 3.2: Architecture Diagram

Initially local and remote monitoring agent runs JADE platform on local machine and cloud independently and sharing their AIDs with each other. AID uniquely identifies the agent with in multi-agent systems; it also contains agent name and addresses. Both monitoring agents obtain each other AID and save it as environmental states. Any agent in local platform can communicate with remote monitoring agent after getting its AID from environmental states. Remote platform contains clones of working agents in local platform. These clones are actually mirrors of their respective working agent. Similarly these clones can communicate with local monitoring agent using its AID saved in their environmental states. The communication is carried out using ACL messages, these messages can have string value or serialized object as content, also ACL message has flag which tells the purpose of message called performativity. Platform uses String value to request other platform for any process while serialized object is used for sharing environmental and local states. Both platforms are fully adjacent now.

The working agent gets remote monitoring agent AID from its environmental states and send him ACL message. The ACL message contains serialized object of local states of respective agent and flag is set to create clone agent. Remote monitoring agent just parse the received message, read the reason of sending message which is obviously for creating clone agent. It creates clone agent and provides initial state from content of ACL message. Since clone and genuine agent has synchronized states and clone agent send message to genuine agent eventually both agents and its clone has AIDs of each. This is completely humanitarian way of exchanging information and now both agents begin direct communication and original agent keep sending stream of synchronized states to its clone. In this way all the working agents in local platform request for their clones to remote monitoring agent on remote platform on cloud, it creates clone which further synchronize with its working agent. The complete process is described in sequence figure 3.3.

The agent and its clone can crash unexpectedly; fortunately proposed approach is capable of re-creating any of them. In case clone crashes, the genuine agent will send request with synchronized states to recreate clone. The synchronized states would be serialized object containing local states of genuine agent. Remote monitoring agent will create clone and facilitate to synchronize communication with genuine agent. Similarly when clone agent discovers that its genuine agent is lost. It sends request including its internal states to local monitoring agent for creating new agent. The internal states of clone is obviously the synchronized states of genuine agent. Local monitoring agent creates agent with synchronized states provided by its clone agent.

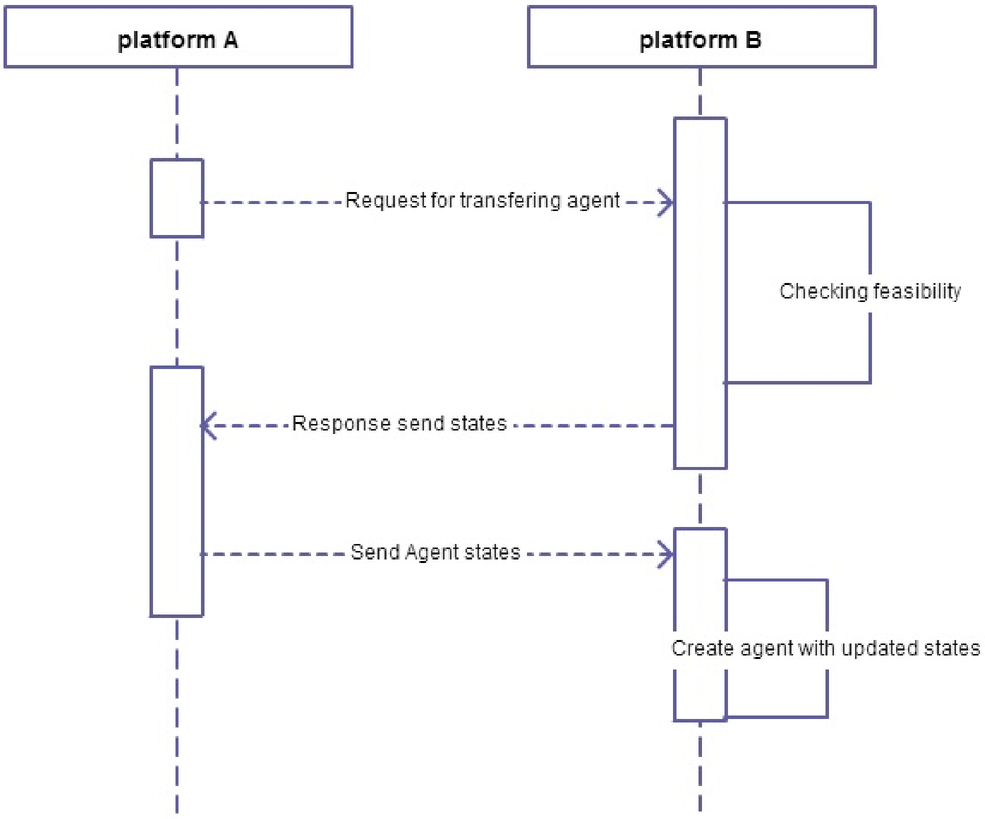


Figure 3.3: Sequence diagram for mobility process in proposed approach

Newly created agent get AID of its clone from synchronized states and send alive message. The agent gets necessary environmental states and ready to contribute in multi-agent systems. Hence this approach can re-create crash agents and its clone with synchronized states when any of them crashes.

3.4 Conclusion

Analysis of literature review help in extracting major problem in previous approaches that are being used to solve this problem. Unfortunately these approaches were not suitable for our problem. In multi-agent systems, agents produce and use information. Some information need to share with other agent in multi-agent and some information should be private to respective agent. In simple word there is need to classify the states of agents in multi-agent systems. Hence in proposed approach states are divided in local states (private to agent) and environmental states (share with all agent in MAS).Other challenging matter was to save data, in older way of storing data such as databases, serialization, Jade persistency framework are not able to store data which changes such frequently as states of agent changes. In order to cope this issue, proposed approach create clone agent who consistently communicate with genuine agent to share latest states. Clone agent is reflection of original agent but it saves the internal states of agent which are later use to recreate agent when it crashes.

The approach put forward in this thesis is based on communication of agents because communication especially text based messages does not produce load on system. There are two platforms, one is running on local machine while second is running on cloud. Agents reside on local platform and their clones are on remote platform, we call them clone agents. Both agent and its clone communicate with each other to maintain latest states of agent. The states of agent are enclosed in serialized object. The states among clone and agent are shared via ACL message having object which contains states. When any one dies either agent or its clone, it sends ACL message to concern monitoring agent to create agent or clone agent with latest states. Concern monitoring agent is further responsible for creating agent and initiate communication among agent and its clone. Even if the several agent and clones dies at same time the system will probably recreate them and start functioning properly. In next chapter these prototypes will be test and evaluated. The research questions will answer on basis of obtained results and compare outcomes with hypothesis.

Chapter 4

IMPLEMENTATION

4.1 Introduction

This chapter presents the implementation of the proposed approach for persisting agent states. It covers the detailed implementation of each module and resources that are used to develop the prototypes. There were two prototypes which were developed in this thesis; first was developed using the proposed approach whilst the other was developed using existing state of art technique. There are several techniques which were previously used to persist agent states. In this thesis a Database is used because it is most popular technique among all other techniques for persisting agent states. Although, this technique has some drawbacks but so far it is considered to be a reliable. Both prototypes are evaluated in chapter 5 and their results will be compared for conclusion.

This chapter begins with an introduction to the functionality of proposed approach. The tools and techniques were used in development of prototypes are discussed the next section. The concrete implementation and working of prototype for proposed approach is considered first which is followed by the implementation details of database technique. The chapter conclude by presenting the overall challenges which were faced when implementing these prototypes.

4.2 Related Tools and Systems

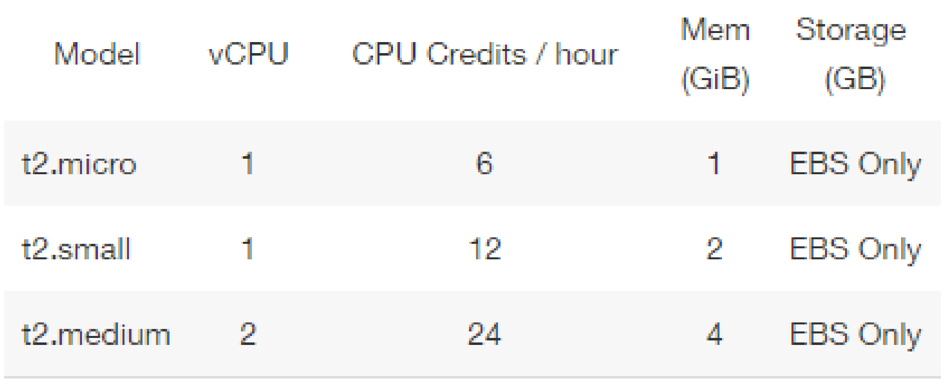
The prototypes were implemented using diffident technologies. These prototypes could be implemented on single machine, over network and over internet as well. The scenario is selected after a detailed study of the design patterns used to develop Multi-agent systems. The detailed architecture is presented in chapter 3. The tools and systems used for implementation are below.

4.2.1 Amazon EC2

Amazon Elastic Cloud EC2 is an Amazon cloud service which has been providing IT infrastructure over the internet since 2006. Amazon offers global compute, storage, database, analytics, application, and deployment services to organizations which are trusted by hundreds of enterprise companies and start-ups. These services can significantly improve the organization progress and also reducing IT infrastructure cost. Organizations do not need to deploy and maintain their own IT infrastructure because service providers include Amazon offers robust and stable infrastructure in very reasonable price. The companies start preferring using computing services over the internet to reduce their workload of managing human resources, workload for deploying and maintaining computers and networks and cost of hardware.[11]

Amazon web service EC2 able use to access instance using web interface. A web interface allows the user to obtain and configure instances. An Instance is a virtual machine created by the web service which can boot an Amazon machine image (AMI). This contains a pre-configured image of the virtual machine that is used to create virtual machines. The user has complete control of the virtual machine and it allows them to access computing resources across Amazons availability zone network. Amazon EC2 is scalable which means instances can be scale up and down with respect to computing requirements. On an increase in computing load, EC2 allows users to launch new instances and similarly close instances on reducing load. These elastic scaling results in efficient use of resources and makes economical for user to pay only for only computing they used. Amazon EC2 offer different types of instance which are discussed below

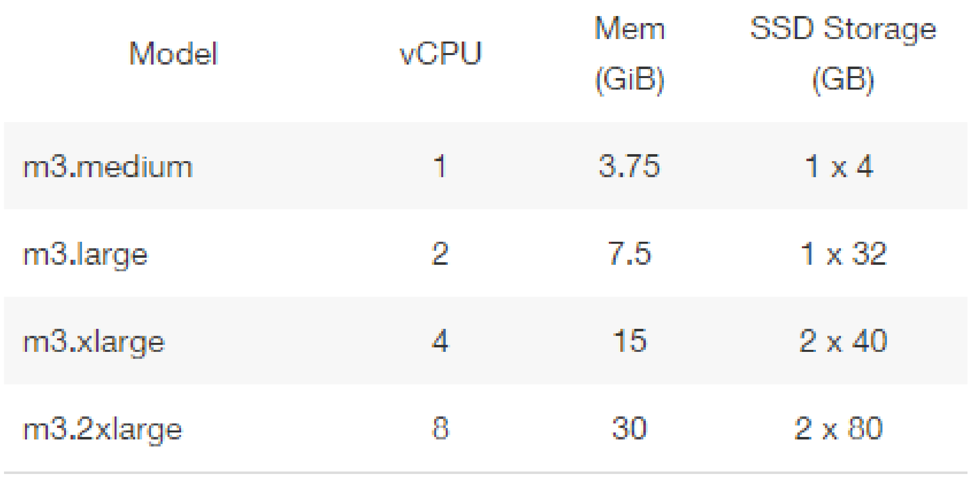
T2 is general purpose instance with baseline level of CPU performance but it can rise above baseline depending upon the CPU credits. T2 instances are Burst able Performance Instances which means instance would not have consistence CPU performance. The performance is governed by credits and credits of instances increase when they are idle and consumed when they are using CPU. These instances has Intel Xeon Processors operating at 2.5GHz and could rise to 3.3Ghz with memory, and network resources. The different type of T2 instances and their specification are listed below in table 4.2



Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage (GB)
t2.micro	1	6	1	EBS Only
t2.small	1	12	2	EBS Only
t2.medium	2	24	4	EBS Only

Figure 4.1: T2 instances specifications [11]

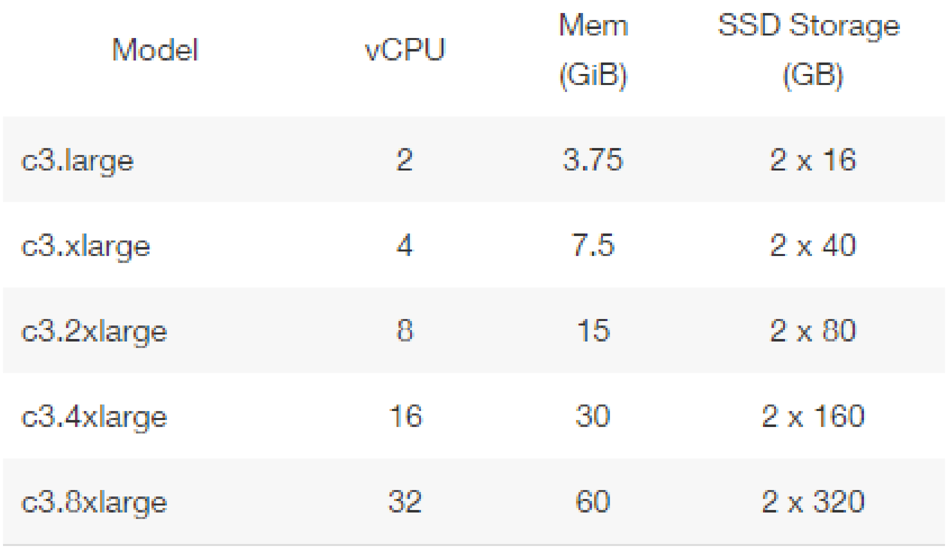
M3 is another general purpose instances empowered with Intel Xeon E5-2670 v2 (Ivy Bridge) Processors which have consistant CPU performance. It has SSD storage for faster performance of I/O devices. It is therefore appropriate choice for many medium scale applications. The figure 4.2 shows the detail specification of M3 instances.



Model	vCPU	Mem (GiB)	SSD Storage (GB)
m3.medium	1	3.75	1 x 4
m3.large	2	7.5	1 x 32
m3.xlarge	4	15	2 x 40
m3.2xlarge	8	30	2 x 80

Figure 4.2: M3 instances specifications [11]

C3 instances are compute optimized and allow users to access high performance CPU performance. Figure 4.3 shows the specifications of different type of C3 instances



Model	vCPU	Mem (GiB)	SSD Storage (GB)
c3.large	2	3.75	2 x 16
c3.xlarge	4	7.5	2 x 40
c3.2xlarge	8	15	2 x 80
c3.4xlarge	16	30	2 x 160
c3.8xlarge	32	60	2 x 320

Figure 4.3: C3 instances specifications [11]

I2 is storage optimized family of instances whose storage capabilities with high speed read and write operation and high IOPS. The I2.x large instance has the lowest storage capability among the instances of this family i.e 30.5 GB and I2.8x large has maximum storage of 244GB.

4.2.2 Inter-platform Communication

Two JADE platforms were used in the proposed approach and the agents in these platforms cannot communicate with each other. The approach used for inter-platform communication is MTP. It is message transfer protocol in JADE for communication between agents in different JADE platform. The marshalling, un-marshalling and transmiting of ACL message in this protocol is according to FIPA specifications. JADE 3.2 and above versions support HTTP and IIOP message transfer protocols. However in prototype implemented using proposed approach HTTP-MTP was used. HTTP-MTP is installed in main container by default. The explicit installation is required to use it in peripheral containers. The working of MTP is shown in figure 4.4.

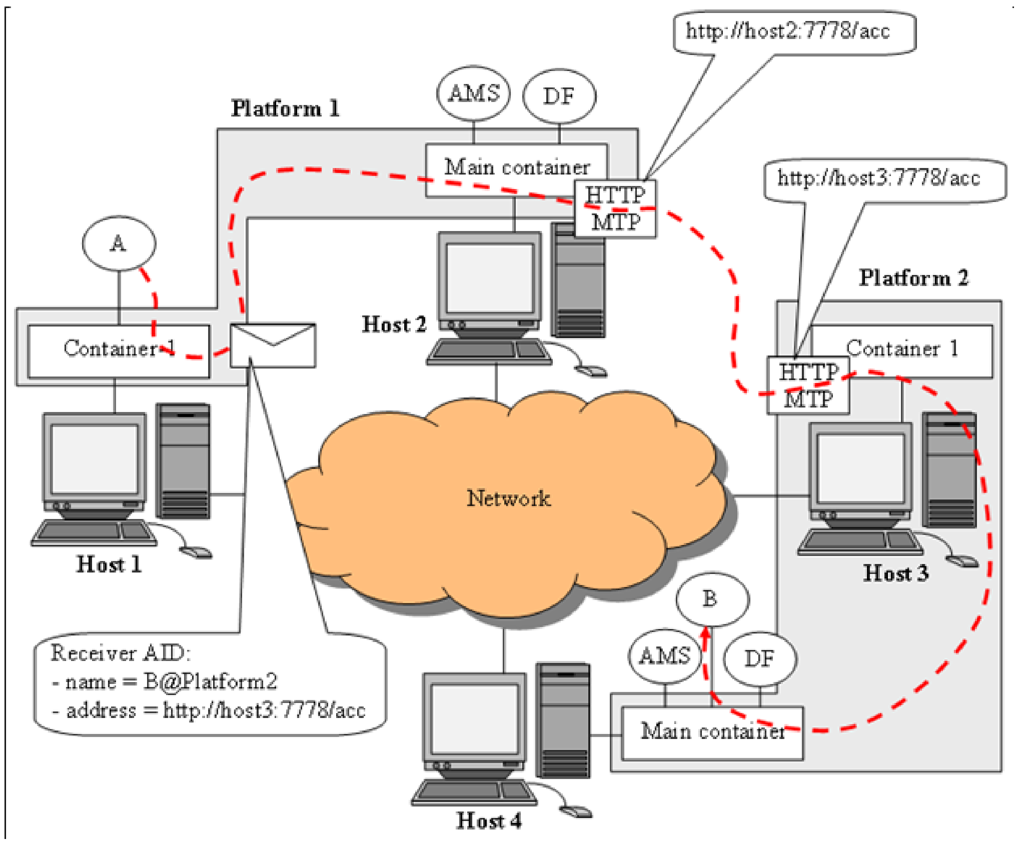


Figure 4.4: HTTP-MTP working [12]

4.3 Implementation of Prototypes

Chapter three of this thesis contains the architecture of the proposed approach along with some concrete details. Similarly it is possible to develop multi-agent systems using a database oriented persistency framework. Two Agent-based systems were designed and developed in order to evaluate the proposed approach. The developed systems simulated an auction system with each having the same functionalities. One system used previously proposed approach and other is developed using proposed approach. Both prototypes were developed using same resources, technologies and functionality for unbiased evaluation. Using different hardware for each prototype will not give fair results for comparison, because prototype running on better hardware will definitely have good results. Therefore agent development platform used for both systems was JADE and deployed on Amazon EC2 and Local Machine.

These prototypes simulated auction system because it is real time. The working agents were customized bidders who bid like humans. A bidding algorithm was designed and implemented in order to make an agent sufficient intelligent so that it can bid. Algorithm abled agents to collect information from environment and analysed it to make an appropriate bid, therefore the agents needed strong communication. Frequently changing agent states were real challenge to persist and they cannot be saved using previously proposed approaches. The working agents which were customized bidders in system were frequently changing their internal states.

From previous discussion we can conclude that the proposed approach is feasible for persisting agents states and recreates them when they die in multi-agent systems. This approach can be implemented using available tools and techniques. The prototype is implemented using two platforms which are recommended way of using proposed approach. In the prototype, one platform was deployed on Amazon EC2 whilst second platform was was deployed on local computer. The platform which was running in local computer contains working agent who actually bidders. The clones were resided on platform running on cloud. The agents in both platforms communicated with each other using the internet.

The second prototype was developed using JADE Persistency Framework which uses database to save agent states. In this prototype, a database which was a persistency unit and it was placed on Amazon EC2 instance. However the Multi-agent system simulated auction was deployed on local machine. Its been make sure to send necessary requests to database for fair evaluation. The architecture of this prototype will discuss later in this chapter for more understanding.

4.3.1 Implementation of Proposed Approach

As auction system was simulated, therefore all the possible states could involve in auction system were gathered. These states were classified in environmental and local states as discussed in chapter three. For example an agent name, agent AID, agent amount (Total amount agent has to make bid) were Local states of an agent whilst the states such as; highest bid so far ,agent addresses who made last bid, Addresses of remote monitoring agents were the environmental states. The local and environmental states were encapsulated in two objects. The object contains environmental states were accessible to all other agents in system. Each working agent maintained its local states which were not assessable to any other agent.

JADE framework was used to develop the prototype having two JADE platforms. A platform deployed on local machine was called the local platform while a platform which was deployed on Amazon Ec2 instance was called Remote Monitoring platform. Each platform had only monitoring agent known as local monitoring agent and remote monitoring agent. The working agents were placed in local platform and clone agents were on remote platform. These two platforms were running independent of each other and they communicated using ACL messages. Each working agent in system created and maintained its Local states object however the monitoring agents were managing environmental states.

The local and remote platforms started independent of each other but the local monitoring agent took name and MTP URL of Remote monitoring agent using input form as shown in figure 4.5. MTP URL was used to access remote platform and agent name helped to deliver a message to particular agent. This information was saved in environmental states object so that working agent can access it. The working agents got remote monitoring agent address from environmental states of local monitoring agent and send ACL message to create its clone. It created clone and provided AID of original agent which is actually working agent who requested for clone. Then clone agent knew about its original agent so that it sent an acknowledgement ACL message to an original agent. The original agent got source AID of clone from ACL message to get AID of its clone. Here both agent and its clone had AIDs of each other. Then after each making bid, working agent sent its latest local states to its clone agent. This way clones stored the local states of their respective working agents.

Considering when working agent dies, the agent will generate ACL message for its clone agent. The message will notify clone agent that working agent has been dead. The clone agent will send message to local monitoring agent for creating new agent. This message would have synchronized local state of dead working agent. The local monitoring agent would create new working agent with synchronized states. Similarly if clone dies, it would send ACL message to its working agent. The working agent would have to send request to remote monitoring agent for creating its new clone with most recent states.

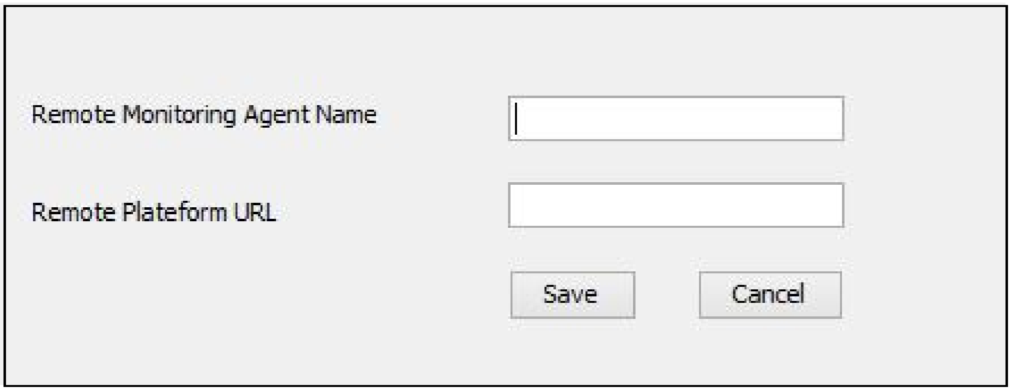


Figure 4.5: Local Platform Input form

4.3.2 Implementation of JADE Persistence Framework

This prototype is also simulating auction system such as prototype implemented for proposed approach. This prototype had one Jade platform which was deployed on local machine. The database is deployed on Amazon EC2 micro instance. The working agents running on local machine would perform the database operations. All the conditions and resources were kept identical in both prototype which consequence in fair results.

The platform ran on local machine has monitoring agent known as Main agent. This agent kept track of all working agent and environmental states. It manages the auction system by maintain data structure. It had list of all working agent and notify agents to bid in queue. The working agents register themselves with main agent by sending their states. The system is capable of persisting states of agent and recreating agent when dies. Whenever any agent would die, main agent will fetch it states from database and recreate it with synchronized states. The detailed flow of prototype is shown in figure 4.6.

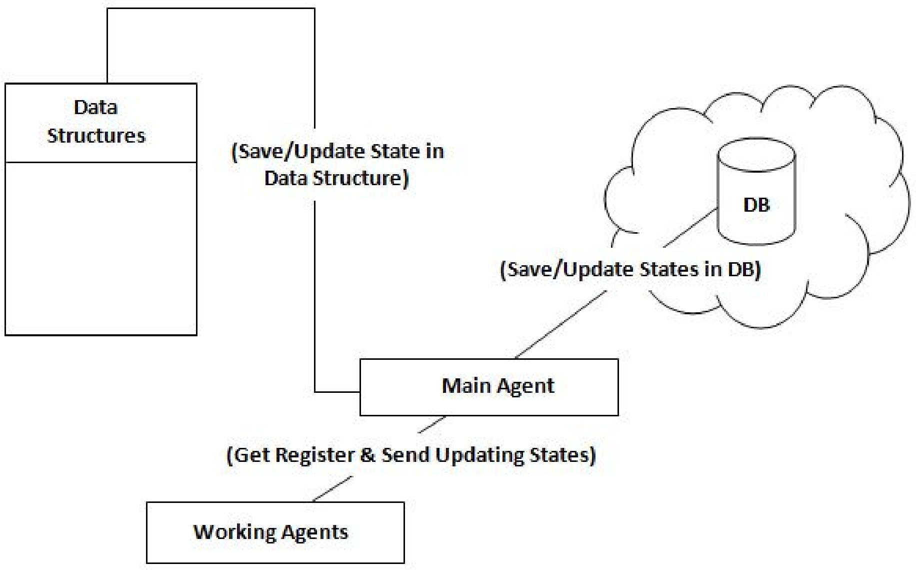

Data Structures
(Save/Update State in Data Structure)
DB
(Save/Update States in DB)
Main Agent
(Get Register & Send Updating States)
Working Agents

Figure 4.6: Architecture diagram of JADE persistence framework

4.4 Conclusion

This chapter considered the implementation of the two prototypes. The first prototypes was implemented using proposed approach while second is developed with JADE persistence framework. These both prototypes simulate auction system, they will be evaluated and their results compared. The prototypes were implemented using same technologies, bidding algorithm for fair evaluation. The basic functionalities of prototypes were tested such as persisting agent stats and recreation of agent. The prototypes passed the initial testing and detailed performance evaluation is discussed in chapter five.

Chapter 5

EVALUATION AND RESULTS

5.1 Introduction

In order to evaluate the performance of the proposed approach, it was necessary to compare it with previous approaches to persistency. There are several persistence frameworks for multi-agent systems. Each framework has different advantages and limitations. The Jade persistence framework was selected for comparison after analysing popular frameworks. This framework was found to be the latest approach and considered to be the most stable and efficient. Also it is available as plugin in JADE platform with good support which made implementation less challenging.

The purpose of evaluation is to gather evidence which then allows assessing hypothesis of the thesis. The results should validate the performance of proposed approach. Most of research questions are answered in this chapter on basis of results. The conclusion of this chapter will provide analysis and reasoning on results. The research question will be answered in the context of the results and their analysis. The evaluation and answer of research questions will lead to either proving the hypothesis true or false.

There were two prototypes developed for comparing performance analysis according to certain parameters which are discuss later in this chapter. These prototypes were implemented with same functionality and techniques to ensure a fair comparison. The complete implementation of both prototypes was discussed in chapter four. These prototypes are evaluated using different matrices such as memory usage, CPU usage, response time and Agent recreation time. This chapter is organized as follows; section 5.1 presents a roadmap for the evaluation process. The section 5.2 briefly discusses experiments perform to get results. The section 5.3 and 5.4 presents the qualitative and quantitative results of both prototypes respectively.

5.2 Experiment Setup

The thesis includes the comparative study of two prototypes simulating auction system. The detailed discussion on reasons for selecting an auction system for implementation is discussed in chapter four. The prototypes were deployed on resources having the same specifications, implemented using same frameworks and techniques to ensure fair results. The auction modules of prototypes were deployed on local machine whilst persisting modules were placed on an Amazon EC2 micro instance. The persisting module in proposed technique was a Multi-agent system. This system was developed using the JADE framework. On the other hand the SQL database was used to save agents state in system was developed with the JADE persistency framework.

Before quantitative evaluation, the prototypes were deployed and tested qualitatively. This testing included the functionality testing of different modules and features by running several test cases. The test cases performed for functionality test are listed in later in this chapter. This testing ensured that prototypes were ready for evaluation process. In order to evaluate the performance of the prototype certain parameters have been identified. These parameters include memory usage, CPU usage and response time of the system. The analysis and discussion on these parameters are added later in this chapter. The detailed hardware specifications are listed below.

The local machine used for deploying and evaluating prototypes was HP 630 laptop. The technical specification of laptop is listed in table below.

Processor	Intel Core i5-2430M
RAM	2GB DDR3
Operating System	Window 8, 32bit

Table 5.1: Specification of Local Machine

The persistence units of both prototypes were deployed on T2 micro instance. This is a general purpose instance, which is suitable for workloads such as web server or IDE. The specifications of the T2 micro instance, are listed in the table below

Processor	Intel Xeon Processors
Processor speed	2.5GHz to 3.3 GHz
CPU Credit/ hour	6
RAM	1 GB
Operating System	Linux
Storage	EBS

Table 5.2: Specifications of Amazon Instance

T2 instances are Burstable Performance Instances. The can burst their CPU performance above the base line performance depending on CPU credits. The CPU credits govern the performance burst. Instances earn CPU credit when they are idle and consume them when they are active. A full core performance of CPU is given for one minute on charging single CPU credit.

5.3 Qualitative Results

Qualitative results mainly concern with functionality and quality of the prototypes. This testing was performed to test working of core modules of both persistency framework. Although the prototypes are simulating auction system but using different approaches. It was important to make sure that key features of prototypes were working. This was initially useful to analyse if these prototypes were suitable for evaluation process. The qualitative results were gotten to ensure that both prototypes are performing same functionalities. The prototypes can fairly compare if and only if they are doing same functionalities. The list of key functionalities and their testing results of proposed approach and JADE persistence framework are presented in next two sub section respectively.

5.3.1 Proposed Approach Qualitative Results

There were several features of prototypes and most of them are similar. Because auction in both prototypes implemented using same logic but steps involved in persistency frameworks were different. To ensure the fair results, it was essential that the prototypes were working and there would no failure on increasing number of agents. The purpose of prototype developing using proposed approach is to store agent states and recreate them when they crash with latest states. The testing of key functions of proposed approach in prototype is listed in table below.

Functionality	**Expected Results**	**Results**	**Remark**
Launching Both platform and sync monitoring agents	Both platform monitoring agents should have AIDs of each other.	The platforms had AIDs of each other.	PASS
Adding bidders on user input	The system should simulate with given number of bidder.	The prototyped added agents according to user input and simulated them.	PASS
Inter-Platform Communication of working Agents	Agents on any platform should able to send or received ACL message from agent on other platform.	The ACL messages were being received and send.	PASS
Clone Creation	Agent should be created in remote platform.	The clone was created with provided states of agent on remote platform	PASS
Working Agents states synchronization with its clone agent.	The clone agent should acquire the synchronized states of working agent.	The working agent sent its synchronized states when state changes	PASS

Working Agents recreation with synchronized states when single or several agents crash	The agent should recreate with synchronized states.	The prototype recreated several cashed agents.	PASS
Clone agents recreation when single or several clone agents crash	The agent should recreate and request for synchronized states.	The prototype recreated several cashed agents.	PASS
Recreating monitoring agents	The monitoring agent should be recreated by prototype and start surveillance of other agents.	The prototype recreated monitoring agent which start monitoring from synchronized states.	PASS

Table 5.3: Test Cases for proposed approach

Each test case is executed several times for accuracy. In case of any unexpected results or failure in any test case, it was debugged and removed particular reason of this failure was identified and removed.

5.3.2 Qualitative Results of JADE Persistence Framework

The qualitative results of prototype which is developed using Jade persistence framework are presented in this section. The core objectives of both prototypes are similar but uses different approaches which consequences in different test cases. The number of test cases could be different for each prototype, because procedure of persisting agent states and recreating crashed agent are different. The table below evaluates the key functionality of this prototype.

The proposed approach has eight test cases and JADE persistence framework has five test cases because proposed approach involved more steps in persisting and recreating processes.

Functionality	**Expected Results**	**Results**	**Remark**
Launching JADE platform with working agents locally.	The JADE platform will run locally and create working agents.	The JADE container launched successfully.	PASS
Adding bidders on user input	The system should simulate with given number of bidder.	The prototyped added agents according to user input and simulated them.	PASS
Communication with MainAgent	MainAgent should have AIDs of all other agent and it should able to send them ACL message.	MainAgent had AIDs of all agents and could sent message to other agents.	PASS
Agents connectivity with Database	The database should contain the states of all agents.	The database had states of all agents in JADE platform.	PASS
Working Agents recreation with synchronized states when single or several agents crash	The agent should recreate with synchronized states.	The prototype recreated several cashed agents.	PASS

Table 5.4: Test Cases for JADE persistence Framework

5.4 Quantitative results

The quantitative evaluations of both prototypes considers their performance. The detailed quantitative evaluation could abled us to answer the research questions. First result metrics were created for evaluation of both prototypes. An experimental setup was made for each metric. The analysis of these results would help to understand the usability of reliability of approach for Multi-agent systems.

The quantities results need to be very clear and comprehensive. Both prototypes were evaluated using same result metrics. Result matrices were created on the analysis of metrics used to evaluate similar approaches. The results were collected using these result matrices and performing experiment discussed in section two of this chapter. The results were fetched by gradually increasing the number of agents in systems. The result matrices used for evaluating prototypes are listed below.

5.4.1 Response Time

The response time of systems includes the time period, an agent took to; register itself with monitoring agent, persisting its state and take part in auction system. Considering the proposed approach, the response time of agent will include; agent registration with monitoring agent, requesting remote platform for its clone and working agent synchronization with its clone agent. Similarly response time in prototype with Jade persistence framework is time it took to register with main agent, store its states in Database and ready to take part in auction system.

On comparing both prototypes the system developed using JADE persistence framework had a lower response time as shown in graph below in figure 5.1. The reason of this big difference was agent communication. In the proposed approach the agents share their states with clone agent via communication and networks which are usually faster. On the other hand Databases are slower in reading and writing which consequences in slower response time. The numbers of agents were gradually increased in the prototype. The system powered by the JADE persistence framework put a significant delay and increase the response time. In contrast the system using proposed approach slightly increased the overall response time. The working agents were directly communicating with its clone on cloud using ACL messages containing serialized states object. This communication reduced the response time where each transaction in database has a slower response time in the reading and writing process..

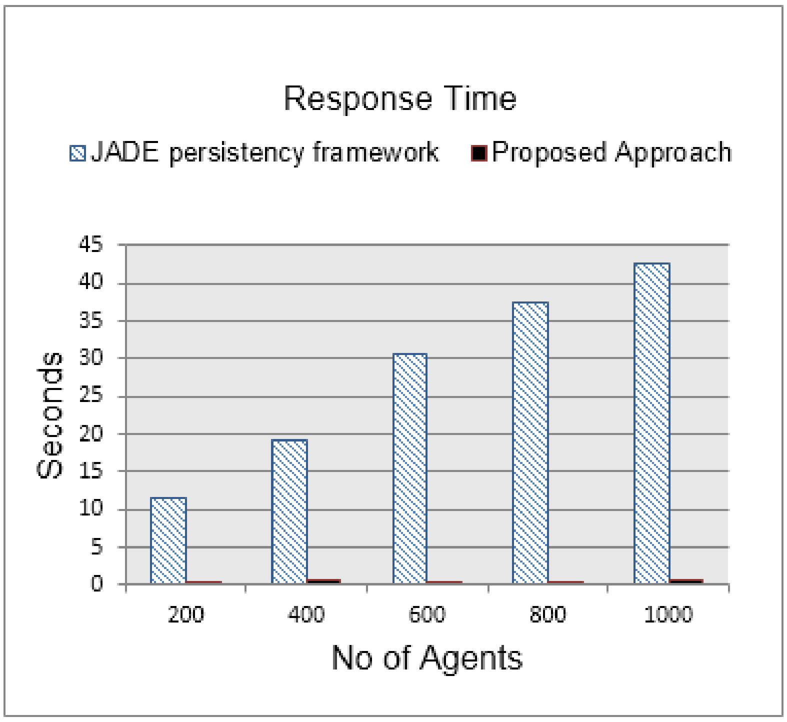


Figure 5.1: Response Time

The response time of the proposed approach remains below 0.6 second for a thousand users. In contrast the response time of the JADE persistence framework increased to forty five seconds. The growth rate of response time of the proposed approach is shown in figure 5.2 to clearly show the patterns in response time. The increase in response time is linear but remain a very slight increase. This slight increase in response time shoes the improvement in the performance of the proposed approach. The approach is therefore be considered as a good choice for a systems where agents' states frequently change.

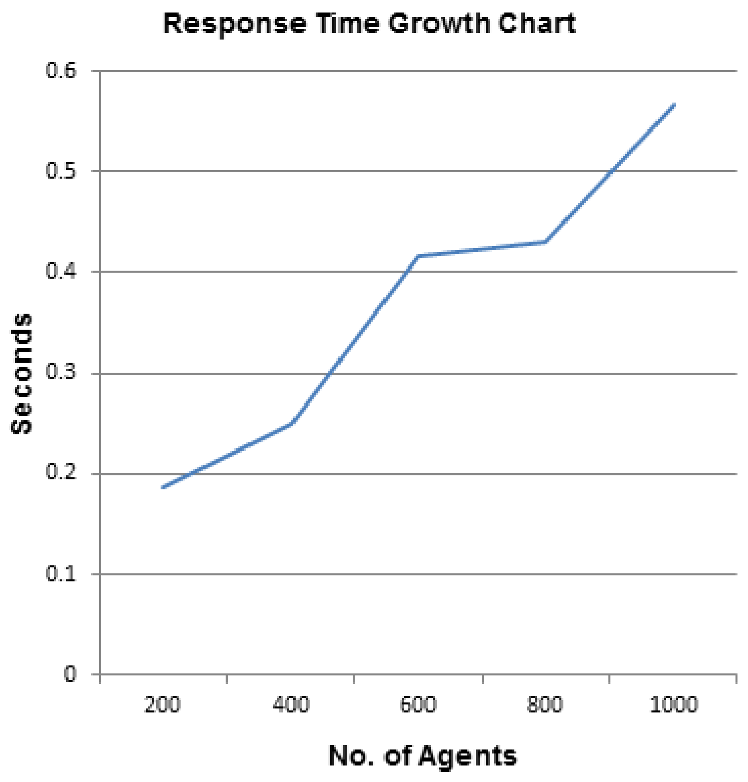


Figure 5.2: Growth Rate of Response time for Proposed Approach

5.4.2 CPU and Memory Usage

The overall CPU and memory analysis showed that proposed approach considerably efficient in CPU performance. Their usage patterns are used to compare the performance analysis of both prototypes. Their analysis will help to understand the performance of both prototypes. The prototype is only considered to be useful when it can execute and deploy on reasonable computational resources. Should prototypes demand for more computational and hardware resources, this would reduce the utilitarian aspects of the system. The new persistence approach should be resource effective as well. The Multi-agent systems can vary in number of agents; they can contain few agents or one enormous. The Multi-agent systems containing fewer agents would prefer to use less hardware resources. Similarly multi-agent systems having enormous number of agents need resource effective approach to run smoothly.

As far as memory is concern the prototype with proposed approaches use more memory than database driven framework as shown in figure 5.3.

This usage of memory is because clone of local monitoring agents resides on the local machine. The number of agent increases as monitoring agents are not exists in database driven approach. The working agent in local machine consumes memory because they hold some additional information such as Local and monitoring agents AIDs, remote platform information. Although the proposed approach consumes more memory than the JADE persistence framework, this is not really that limiting in terms of scalability. Also it consumes memory to provide additional information and more stability. The systems especially cloud instances often have large amounts of memory available. In contrast memory usage is small even thousand agents are running. These thousand agents consumes 32MB of memory which is not such high. These day smart devices and hand held devices have memory in Giga bytes.

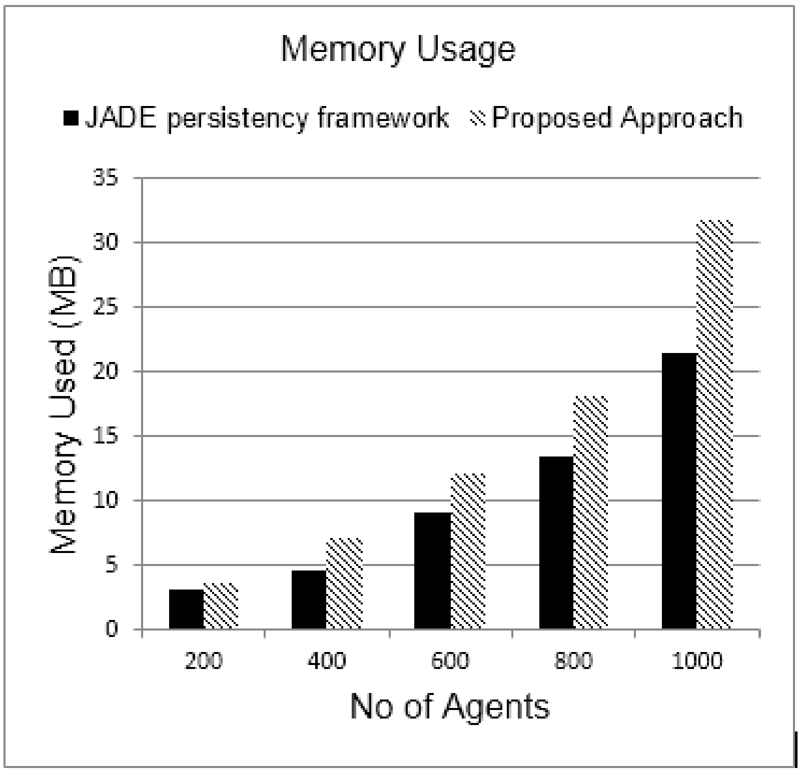


Figure 5.3: Comparison of Memory Usage

5.4.3 System Launch Time

The launch time of systems is duration system took to start actual auction. At this state both systems had complete their initial setup. The JADE persistence framework setup included database initialization, starting main agent and working agents registration. The average launch time of JADE persistence framework was four minutes. On the increasing number of agents, the system crashes quickly because of database response times. The response time graph in figure 5.1 clearly shows that on increasing the number of agents, decreases the response time.

The launch time of prototype uses proposed approach contains time it takes to synchronizes both local and remote JADE platforms. The synchronization includes; launching of both platforms, creating working agents and their clones, synchronizing working agents with their clones. The launch time of this system was two and half minutes for every test case. This launch time however can be reduced by increasing number of monitoring agents. This confirms that the proposed approach is able to manage internal state of agents when they die. Also this time is sufficiently less than JADE persistence framework.

5.4.4 Agent Recreation Time

Agent recreation time is one of the most vital features of prototype. The agents can crash while performing their tasks. Both prototypes are able to recover them with synchronized states. It is important to analyses the performance of both systems. To compare the agent recreation time of the prototypes, agents were crashed during their execution. Different numbers of agents were killed to analyse the performance of prototypes and net effect on the end results. For example the bidder wins the auction should remain same either agents were killed or not. The killed agents should recreate with synchronized states so that it can adjust in environment.

The time both prototypes took to recreate crashed agent is agent recreation time. The recreation time comparison is show in graph in figure 5.4. The graph shows proposed approach is much quicker in recreating a crashed agent. It is very important to recreate crashed agents quickly because any delay in recreating agent could cause instability in the system. Considering agents crash that change local states frequently, delay in recreating these agents result in loss of significant information. This information includes local states which should recover as soon as possible. The result showed proposed system is quick in recreating crash agents. The reason is proposed approach is basically communication oriented. The networks are very fast which improved the performance of system.

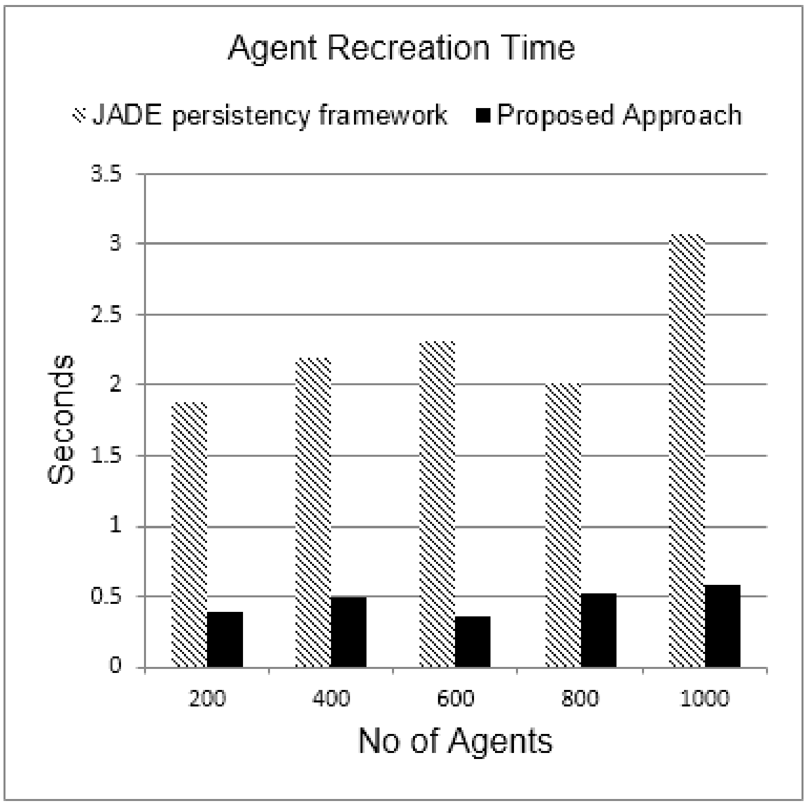


Figure 5.4: Comparison of Agent recreation time

5.5 Conclusion

The overall performance of the proposed approach outperformed the previous persistence frameworks. Meanwhile approach gives in memory solution, someone may consider it less robust or reliable than the database driven persistence frameworks. The database holds the structural data where agent states update frequently and agent can change its internal structure. The scalability, runtime error handling and delays were challenges in recent approaches. These challenges make in-memory cloud based approach for persistence more appealing. Also using cloud enhances the scalability capabilities of proposed approach.

The results showed that the proposed approach improved the CPU utilization, Agent recreation time, response time and system launch time. The recreation time, response time and launch time in any experimental setup slightly differs because communication is faster than requesting database.

These results are significantly improved and quite efficiently working for thousands agents. On other hand old approach for persisting agent states crashes while simulating auction for thousand agents. This improvement in persisting agent states will significantly enhance the performance of Multi- agent systems.

The memory usage in proposed system is increased because proposed approach put forward in-memory solution. The states are saved in memory as encapsulated object. The size of this object can vary but these memory usages will not much high. The current technologies have memory in tetra bytes especially in case of cloud. Also small devices and smart phones have memory in Gigabytes. Certainly this memory use of proposed approach is quite reliable and does not put much burden on the hardware. The detailed conclusion and answer to research questions are stated in chapter six.

Chapter 6

CONCLUSION AND FUTURE WORK

The research in thesis presents a new type of persistency framework for Multi-agent systems. The proposed approach provides in-memory storage of internal states of agents. These states are stored in mirror agent which resides in cloud. The internal states of agent are being synchronized with mirror agent by communication between them. Should an agent or its clone crash, the remaining agent is responsible for re-creating the crashed one. Section 6.1 presents the final assessment and research questions are answered in order to answer research hypothesis. Section 6.2 includes future work which is followed by section 6.3 which presents conclusion of this research.

6.1 Final Assessment

In first chapter, the research questions were state which were constrains on the performance and implementation of the approach. They have been evaluated from the result driven from chapter five. These research questions have to discuss to clearly understand the performance and the scalability of the approach. The reply to research question Q-2 that is

How can we propose a flexible persistence framework for capturing and persisting states of agent at the application level?

The evaluation results showed that a widely applicable persistence framework can be developed for capturing and persisting states of agent on application level. The proposed approach has no concern with task or execution of agent. It saves and keeps synchronized states of agents without knowing the semantics. The behaviour of agent and its responsibilities does not demand any change persistence framework. This means proposed approach is generic and can use in multi-agent systems belong to any domain.

Discussion on Research Question RQ-3

What should be the behaviour of multi-agent systems when several agents or severa clonel agents die in system or agent states change frequently?

In proposed approach whenever an agent dies, it sends a request to the remote monitoring agent to create new agent with synchronized states. In chapter 2 there is complete discussion, how previously proposed approaches recreate agents and their efficiency and accuracy. If one agent or several agents die it would not make any difference because clone will take their place and will send request to Local remote agent to produce new agent. The re-created agent would have synchronized local states which clone had provided and environmental states which would get from Local monitoring agent. Similarly in case several or more than half of clones crash, their respective agent will send request to remote monitoring agent to create clone and agent will also provide synchronized states for creating their clone with latest states. The scenario is discussed in section 5.4 where more than half of agents were killed to test the performance of prototype for agent recreation. The results showed that agents were recreated in very small friction of time with their synchronized states.

The proposed approach is communicative and networks these days are very fast. These agents can share states with their respective clones in friction of second. Also the ACL message has very reasonable size and creates less traffic load. Considering the case clone agent dies, the clone agent will get new synchronized state from its working agent. The complete procedure is presented in chapter three.

Discussion on Research Question RQ-4

What would be performance and scalability of the proposed approach?

The proposed approach was tested using a thousand agents. The results discussion in previous sections clearly showed that approach is satisfactory in term of performance. In order to check scalability proposed approach is tested for thousand agents for analysing performance matrix. The large multi-agent systems usually have less than hundred agents. Therefore this approach is useful for persisting agent states and recover crashed agents. The results and answer to research questions demonstrate that the hypotheses is true. The qualitative results showed that the agents states in Multi-agent systems can be saved using proposed approach. The agent crashed while execution can be recreated. Similarly the quantitated results strengthen the validity of hypothesis. These results proved proposed approach quite better than previous approaches.

6.2 Future work

The Multi-agent systems that are using previously proposed persistency frameworks which make them slower. The comparative results showed that approach proposed in this thesis is considerably good in performance. There should be some method to equip such Multi-agent systems (using old persistency frameworks) with proposed approach. This transformation process should not slow the performance of systems. It should work independently and parallel so that they should no hurdle for system to perform its regular tasks.

Currently Multi-agent systems mostly contain less than a hundred agents. The use of smart devices are increasing by peoples to manage their daily routines and solving their problems. Agents can do this management tasks in behalf of humans. In this context each user can map with personalized agent. In this scenario, there will be immense number of agents who will put significantly high burden on agent framework. So far this evaluation is enough but there would need to evaluate for such numbers of agents. There is also possibility that there would need to perform some improvements too.

6.3 Conclusion

The use of the internet of things is widely increasing which demands for new paradigm of developing application and managing them. These devices can huge in a number and distributed over the networks for solving any problem. The Multi-agent systems can be one of possible option to handle such problems. Putting agents in IOT results in massive number of agents because of increase in use of embedded and wearable devices. Consider each devices as a customised agent, each devices will has different hardware specification and configuration which will increase the probability of an agent to crash. Therefore a persistency framework was introduces to improve the performance and the reliability of the Multi-agent systems.

The architecture of persistency framework proposed in this thesis is presented in chapter three. In order to evaluate the framework a prototype was implemented using proposed approach which was compared with JADE persistency services. The comparison results are formulated and presented in chapter five. The discussion on these results and assessment showed performance benefits of proposed approach. In final assessment section, the research questions were answered. They were consider to be constrains on proposed approach. These answers proved research hypothesis true. These results showed that proposed approach scale Multi-agents systems better as compare to JADE persistency services.

Bibliography

[1] Quan Liu, Lu Gao, Ping Lou, *Resource Management Based on Multi-Agent Technology for Cloud Manufacturing*, School of Information Engineering, Wuhan University of Technology, Wuhan, China.

[2] Ralph Deters, *Scalability And Multi-Agent Systems*, University of Saskatchewan.

[3] Rocco Aversa, Beniamino Di Martino, Massimiliano Rak, Salvatore Venticinque, *Cloud Agency: A Mobile Agent Based Cloud System*, Department of Information Engineering Second University of Naples Aversa (CE), Italy.

[4] Octavio Gutierrez-Garcia and Kwang-Mong Sim, *Self-Organizing Agents for Service Composition in Cloud Computing* , 2nd IEEE International Conference on Cloud Computing Technology and Science.

[5] Giovanni Rimassa, *Runtime Support for Distributed Multi-Agent Systems*, Ph. D. Thesis, University of Parma, January 2003.

[6] Joan Ametller, Sergi Robles, and Joan Borrell, *Agent Migration over FIPA ACL Messages* , Computer Science Dept. Universitat Autnoma de Barcelona.

[7] E.Cortese, F.Quarta and G.Vitaglione, *Scalability and Performance of JADE Message Transport System*, Centro Direzionale isola F7

[8] Chih-Tien Fan, Wei-Jen Wang and Yue-Shan Chang, *Agent-based Service Migration Framework in Hybrid Cloud* , 2011 IEEE International Conference on High Performance Computing and Communications

[9] Giovanni Caire, Giovanni Rimassa and Fabio Bellifemine *JADE: a versatile run-time for distributed applications on mobile terminals and networks* , 2004 IEEE International Conference on Systems, Man and Cybernetics

[10] Bracciali, Giancarlo Fortino , Francesco Rango, *An application-level technique based on recursive hierarchical state machines for agent execution state capture*, DEIS University of Calabria, Via P. Bucci cubo 41c, 87036 Rende (CS), Italy

[11] http://aws.amazon.com/ec2/

[12] http://jade.tilab.com/

[21] Zhang, H., Jiang, G., Yoshihira, K., Chen, H., Saxena, A.: *Intelligent Workload Factoring for a Hybrid Cloud Computing Model* In California : International Workshop on Cloud Services, Los Angeles, July, 2009.

[14] G. Fortino, A. Garro, S. Mascillaro, W. Russo, *Using event-driven lightweight DSC-based agents for MAS modeling*, International Journal on Agent Oriented Software Engineering 4 (2) (2010).

[15] M. Berger, S. Rusitschka, M. Schlichte, D. Toropw, M. Watzke. *Porting Agents to Small Mobile Devices In Workshop on Ubiquitous Agents on Embedded, Wearable and Mobile Devices*, AAMAS 2002, July 15-16, 2002, Bologna, Italy.

[16] The Hibernate Project Home Page. Available at http://www.hibernate.org.

[17] Gutierrez-Garcia, J.O., Sim, K.M , *Self-Organizing Agents for Service Composition in Cloud Computing.* In USA: 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010.

[22] Ramaswamy, A., Balasubramanian, A., VijayKumar, *A Mobile Agent Based Approach of Ensuring Trustworthiness in the Cloud.* In Chennai: IEEE-International Conference on Recent Trends in Information Technology, 2011 June 3-5 pp. 678-682.

[19] http://www.codingeek.com/

[20] David Wong; Noemi Pacictek, Ton Walsh, Joe DiCelie, Mike Yotmg, *Concordia: An infrastructure for Collaborating Mobile Agents* ,Bill Peet Mitsubishi Eleclric ITAHorizon Systems1432 Main Street Walaham, MA02154, USA

[21] Zhang, H., Jiang, G., Yoshihira, K., Chen, H., Saxena, *A Intelligent Workload Factoring for a Hybrid Cloud Computing Model.*, In California : International Workshop on Cloud Services, Los Angeles, July, 2009.

[22] Ramaswamy, A., Balasubramanian, A., VijayKumar, *A Mobile Agent Based Approach of Ensuring Trustworthiness in the Cloud.*. In Chennai: IEEE-International Conference on Recent Trends in Information Technology, 2011 June 3-5 pp. 678-682.

[23] Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood, *Developing multi-agent systems with JADE.*, Vol. 7. John Wiley And Sons, 2007.

[24] Kumar, Sanjeev, Philip R. Cohen, and Hector J. Levesque. *The adaptive agent architecture: Achieving fault-tolerance using persistent broker teams.* MultiAgent Systems, 2000. Proceedings. Fourth International Conference on. IEEE, 2000.

[25] Song, Cheng, et al. *Optimal control for multi-agent persistent monitoring.* Automatica 50.6 (2014): 1663-1668.

[26] Chowdhury, Nilanjan Roy, and Srikant Sukumar. *Persistence based analysis of consensus protocols for dynamic graph networks.* arXiv preprint arXiv:1404.1168 (2014).

[27] Lunze, J.,Lunze, J., Allgwer, F., Brger, M., Demir, O., Helmke, U., von Heusinger, A. And Schuh,*Multi-agent Systems: Control Theory of Digitally Networked Dynamic Systems.* Springer International Publishing, 2014. 263-324.

[28] Akhtar, Nadeem, Aisha Shafique Ghori, and Nadeem Salamat. *Requirement analysis, Architectural design and Formal verification of a multi-agent based University Information Management System.* arXiv preprint arXiv:1501.01273 (2015).

[29] Sturm, Arnon, and Onn Shehory. *The Evolution of MAS Tools. Agent-Oriented Software Engineering.* Springer Berlin Heidelberg, 2014.

[30] Kabysh, Anton, and Vladimir Golovko. *General model for organizing interactions in multi-agent systems.* International Journal of Computing 11.3 (2014): 224-233.

[31] Caire, Giovanni, Danilo Gotta, and Massimo Banzi. *WADE: a software platform to develop mission critical applications exploiting agents and workflows.* Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[32] Lees, Michael, et al. *Simulating Agent-Based Systems with HLA The Case of SIM AGENT-Part II (03E-SIW-076).* Proceedings of the 2003 European Simulation Interoperability Workshop. Stockholm, Sweden. 2003.

[33] Helsinger, Aaron, Michael Thome, and Todd Wright. "Cougaar: a scalable, distributed multi-agent architecture." Systems, Man and Cybernetics, 2004 IEEE International Conference on. Vol. 2. IEEE, 2004.