International Review of Management and Marketing

ISSN: 2146-4405

available at http: www.econjournals.com

International Review of Management and Marketing, 2016, 6(S8) 27-32.

Special Issue for "International Conference on Applied Science and Technology (ICAST) 2016, Malaysia"

A Large Data Exchange Method for Multi-agent in Java Agent Development Framework

Wathiq Laftah Al-Yaseen[1]*, Zulaiha Ali Othman[2], Mohd Zakree Ahmad Nazri[3]

[1]Al-Furat Al-Awsat Technical University, Data Mining and Optimization Research Group, Centre for Artificial Intelligence Technology, Iraq, [2]School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 Bandar Baru Bangi, Malaysia, [3]School of Computer Science, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, 43600 Bandar Baru Bangi, Malaysia. *Email: wathiqpro@gmail.com

ABSTRACT

One of the business intelligent solutions that are currently in use is the multi-agent system (MAS). Communication is one of the most important elements in MAS, especially for exchanging large low level data between distributed agents (physically). The agent communication language in Java Agent Development framework has been offered as a secure method for sending data, whereby the data is defined as an object. However, the object cannot be used to send data to another agent in a different machine. Therefore, the aim of this paper was to propose a method for the exchange of large low level data as an object by creating a proxy agent known as a delivery agent, which temporarily imitates the receiver agent. The results showed that the proposed method is able to send large-sized data. The experiments were conducted using 16 datasets ranging from 100,000 to 7 million instances. However, for the proposed method, the RAM and the CPU machine had to be slightly increased for the receiver agent, but the latency time was not significantly different compared to the use of the java socket method (non-agent and less secure). With such results, it was concluded that the proposed method can be used to securely send large data between agents.

Keywords: Data Exchange, Multi-agent, Agent Communication, Low Level Data, Java Agent Development Framework
JEL Classifications: C61, C63, C82, C88

1. INTRODUCTION

The efficient management of information exchange has become a challenging issue in most areas of computer science research. A multi-agent system (MAS) is one of the areas that suffer from this problem (Noroozi, 2009). The MAS is comprised of a number of independent software modules named agents. These agents are used to construct a flexible and extensible framework for large heterogeneous and complex distributed software applications. One of the significant characteristics possessed by the MAS is social ability. Social ability means that each agent can co-operate and communicate with other agents, while supporting data exchange, information exchange and negotiation (McArthur and Davidson, 2004). A common language is required in order to achieve efficient communication between agents. Much work has gone into developing an agent communication language (ACL) that is declarative, syntactically simple and readable by people.

The Java Agent DEvelopment framework (JADE) framework is one of the environments used to implement agents in runtime. JADE uses FIPA-ACL language for communication between agents. This language supports the exchange of high level data (messages and small data) between agents located in different machines (logical environment) by using a serialized object as well as the exchange of low level data between agents located in the same machine (physical environment) by using an object. However, the FIPA-ACL does not have a method for the exchange of large low level data between agents located in different machines. Therefore, this paper proposed a method for a MAS communication in JADE with the ability to exchange large low level data between agents distributed in different machines. This method creates a proxy agent, temporarily known as a delivery agent, to transfer data as an object to a receiver agent. The experimental results proved that the proposed method can result in the exchange of large data

between agents located in different machines with a secure connection.

The remainder of this paper is organized as follows. Section 2 provides a brief review of the related work on the communication approaches between agents. The proposed method is described in section 3, while section 4 presents experimental results, in order to demonstrate the proposed method performance. The concluding remarks are given in Section 5.

2. RELATED WORKS

2.1. ACL

There are several software environments concerned to develop the MASs, for example AgentBuilder, FIPA-OS, Retsina, Zeus (Lee et al., 1998) and JADE.

JADE is an open source java software aims to development of distributed multi-agent applications based on a peer-to-peer communication architecture in compliance with the FIPA specifications (Bellifemine, 2005). It is providing a set of services and graphical tools for debugging and testing of agents. A JADE comprises of agent containers that can be distributed over the network and implemented within a java virtual machine (JVM). Each agent lives in a container, and the collection of containers makes up a platform. There is a special container known as main-container that represents the bootstrap of platform as well as all the other containers of platform must be joined with the main-container by registering with it (Bellifemine et al., 2005; Kumar and Kumar, 2014). Moreover, the security policy of JADE provides connection authentication for exchange data between agents, in addition it has a high security through encryption all the interoperation between agents like use remote procedure call for message encryption (Nguyen et al., 2002). JADE supports exchange data between agents with high restrictions on this exchange. One of these restrictions, it can send only small data with the ACL message as Java serialization object to an agent located on other machine. Moreover, the method of sending a serialization object inside ACL message is very slow. Another restriction, JADE provides send object-data, but it needs to proxy (agent controller) of receiver agent on the other machine so can send an object, this is impossible. These restrictions make JADE difficult and poor to exchange a large low level data between agents.

2.2. JADE Environment

A few researchers in the field of MASs, such as (Noroozi, 2009; Garro et al., 2002; Jang et al., 2004; Berna-Koes et al., 2004), have tried to improve the communication and exchange of large data between agents. Noroozi (2009) and Berna-Koes et al. (2004) used extra channels (backchannels) to exchange low level data between agents. The use of backchannels is necessary to establish a TCP connection by the initiator that is included in the server agent (sender). The server opens a passive line on a port and sends a permission request, in the form of an ACL message, with a reference number (unique number of the particular specified channel) to the client agent (receiver) in order to connect with this specified port. If accepted, the server agent will send the low level data over this channel to the client agent. Furthermore, the content of the message should be fixed and agreed to between the agents. This method was designed for a MAS that uses KQML as the language of communication.

Garro and Palopoli (2003) proposed an XML MAS for e-learning and skills management systems. E-learning and skills management systems are concerned with enterprise knowledge management. These systems are needed for the exchange, reuse and sharing of meta-knowledge between members of the scientific community operating in the field of e-learning. Consequently, the authors used XML for representing and exchanging knowledge via the internet as it has the representation capabilities of HTML and the data management features of the DBMS. They exploited these capabilities of XML, and made them suitable for use by agents for the exchange of information and knowledge in e-learning and skills management systems.

Jang et al. (2004) presented an idea to address the problem of message passing between agents located in different platforms of actor architecture (AA), which used an object-based message to exchange data. They extended the behavior of the message manager that is responsible for handling the passing of messages between agents in the AA by adding a new service known as forwarding. The scenario of a forwarding service is: The message manager of the sender agent delivers the message to the current AA platform of the mobile agent (receiver agent). Then, the AA platform delivers the message to the message manager of this platform where the mobile agent currently resides, and finally, that message manager delivers the message to the mobile agent. Moreover, this architecture needs other components, such as the transport manager, transport sender, transport receiver and delayed message manager, in order to achieve the process of message passing between the agents. Thus, all these components will increase the consumption of system resources.

3. PROPOSED A LARGE DATA EXCHANGE METHOD FOR MAS

There are three methods in the JADE framework for the exchange of data between agents. The first is to exchange data as a serialized object inside the ACL message. To send the serialized object, the setContentObject is used to embed the serialized data inside the ACL message, while the getContentObject is used to receive the serialized object sent from the other agent. In this method, the size of the data should be small as well since the data has to be saved as a serialized object. The drawback of this method is that it is very slow, and thus, it will take a long time to send the data. The second method is to send the data by creating a new agent and then embedding the data inside that new agent. The instruction to send data is createNewAgent, while the instruction to receive the data is getArguments. The third method is to send the data as objects. This method needs a proxy (agent controller) of the receiver agent in order to be able to send the data. There are two cases in this method: If the sender and receiver agents belong to the same platform (intra-platform), then the process of sending the data is very simple

because the proxy of the receiver agent can easily get it by using getAgent. The data object can be sent by using putO2AObject and received by the receiver agent by using getO2AObject. In the case where the sender and receiver agents are in different platforms (inter-platform), then it would be impossible to connect the sender agent to the proxy of the receiver agent. Therefore, this method is not appropriate for the exchange of data between agents in different machines.

The proposed communication method was aimed at improving the data exchange process between agents and making the communication more efficient in MAS. The proposed method uses three types of agents, namely sender agent, receiver agent and delivery agent. The scenario of the proposed communication method for MAS in JADE between two machines (A and B) is as follows:

1. Create a sender agent in the main container of the platform in machine A
2. Create a receiver agent in the main container of the platform in machine B
3. Prepare the required data for sending by the sender agent
4. Create a new delivery agent by the sender agent in a new container (Container-1) in the platform of machine B with the data embedded as an object
5. Send a request message from the delivery agent to the receiver agent about the initial receipt of the data
6. Move the receiver agent to container-1 of the delivery agent
7. Send a message from the receiver agent to the delivery agent informing that the former is ready to receive the data
8. Data is sent by the delivery agent and is received by the receiver agent
9. Move the receiver agent to the original main container of the platform in machine B
10. Delete the delivery agent and container-1.

3.1. Sender Agent

This agent is a static agent. It reads and puts the required data into an object in preparation for sending it to another agent. After that, it delivers the object directly to the receiver agent when the latter is found in the same machine. In contrast, when the receiver agent is located in another machine, then the sender agent creates a new agent (delivery agent) in the target machine and embeds the required object inside this new agent in order to send it with the delivery agent to the receiver agent. The java pseudo code of the sender agent is shown in Figure 1.

3.2. Delivery Agent

The delivery agent is responsible for transferring data from the sender agent to the receiver agent. This agent is created by the sender agent at the target machine where the data should be received. This agent can be deleted after completing the data transfer, or may remain for more transfers in the future between the sender agent and the receiver agent. Since JADE is operating in JVM, the delivery agent is created logically on the target machine, but physically the delivery agent is created in the same machine as the mother agent (sender agent). Subsequently, the delivery agent will use the resources of the sender agent machine like the CPU, memory, etc. The java pseudo code of it is shown in Figure 2.

3.3. Receiver Agent

The receiver agent is responsible for receiving the data from the sender agent. This agent resides in the target machine and is ready to receive the data. This type of agent is reactive in that when it receives a message from the delivery agent, it moves to the same container of the delivery agent in order to get the data, and then returns to the previous container. The pseudo code is shown in Figure 3.

The AUML (sequence diagram) of proposed communication method for MAS is shown in Figure 4. This diagram shows the process of transfer data from sender agent (machine A) to receiver agent (machine B) by using the mediator entity that is delivery agent.

From the AUML, we saw delivery agent had been created in a new container on the target machine, so that we can ask question; why sender agent did not create delivery agent in the main container of target machine directly? Because the JADE framework deal with platforms on all machines as virtual platforms contain on virtual containers. Consequently, JADE cannot create a new agent from machine inside container already found in platform of another machine. For example, the sender agent in machine (A) cannot create delivery agent in the main-container of machine (B), where when anybody try to implement the above process then the JADE will create delivery agent in new container (e.g., Container-1) in platform of machine (B).

4. EXPERIMENTAL RESULTS

To evaluate the proposed communication method for the MAS, two approaches were considered for comparison: In the "java

Figure 1: Pseudo code of sender agent

```
Load data;//multimedia, control, etc.
Create profile P with parameters "Machine B", "Container-1";
Create a new container of profile p;
Create a new agent ("Delivery Agent", data) in Container-1 of
platform in machine B.
```

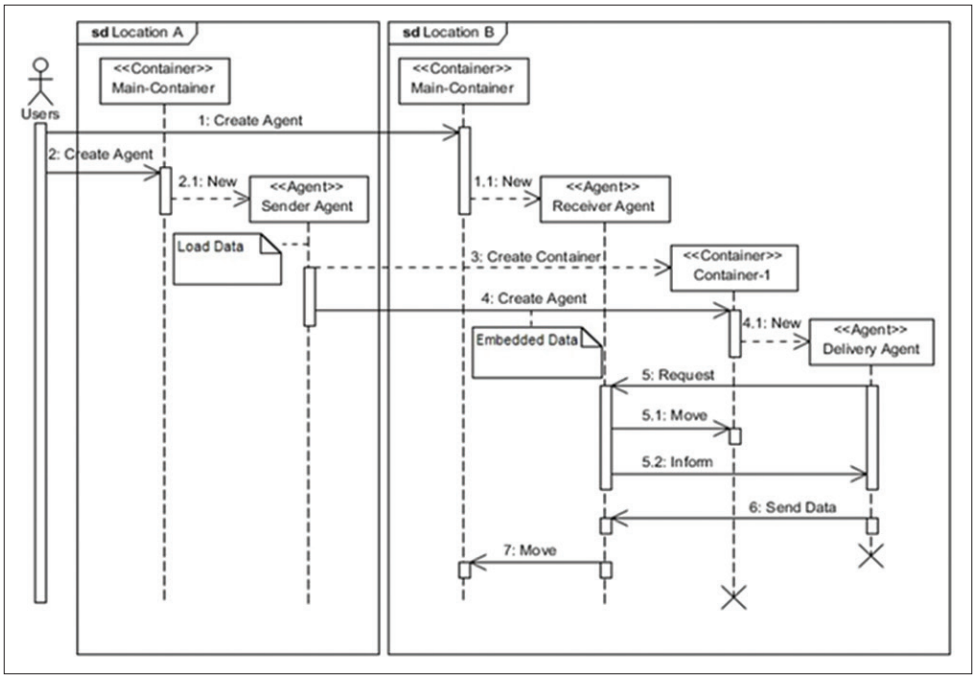
Figure 2: Pseudo code of delivery agent

```
Data=getArguments();
Send request message to the Receiver Agent in order to initialize the
receiving of data;
Add cyclic behaviour until the Receiver Agent ends with readiness
to receive data;
Proxy=getAgent (Receiver Agent);
Proxy.putO2AObject (data);
Send message to Receiver Agent to get data.
```

Figure 3: Pseudo code of receiver agent

```
Add cyclic behaviour until a message is received from the Delivery
Agent to initialize;
Move to container of Delivery Agent;
Add cyclic behaviour until a message is received from the Delivery
Agent to get data;
Data=getO2AObjcet();
Move to the previous container.
```

Figure 4: AUML sequence diagram of proposed communication method for multi-agent system



socket" approach to the exchange of data between two machines, a java socket class was used to determine one machine as the server and the other as a client; while in the "agent" approach, three agents, as proposed in section 3, were used for the exchange of data between two machines. This approach was implemented by using the JADE 4.3.3 environment. The specifications of the two machines used in experiments were: The sender machine had a Core i5 2.60 GHz CPU and a 12 GB RAM, while the receiver machine had a Core i7 3.40 GHz CPU and 4 GB RAM. In addition, the Windows 8.1 Single Language was used with each of the machines. The benchmark KDD Cup 1999 was used to generate 16 datasets having a different number of instances (size), as shown in Table 1. These datasets were used to evaluate the performance of the proposed communication method and to compare it with the java socket approach.

The utilization of the system resources, as well as the time, were used as evaluation measurements for comparing the performances, where the Sigar class in java was used to compute the load on the system resources like the CPU and RAM. In addition, the currentTimeMillis java function was used to obtain the time interval (latency) in milliseconds before and after the sending of the whole dataset between machines. Table 2 presents the averages of exchange each dataset 10 times between machines. At each time interval, the peak value of the utilization of the CPU and RAM was taken for one second only. Accordingly, the mean and standard deviation were computed for each outcome. Table 2 shows the utilizations of CPU were close between approaches on the sender machine. Moreover, Table 3 shows the mean and standard deviation with the best and worst cases of latency time for 10 times of exchange each dataset. Figure 5 compares the utilization of the CPU between the Java socket approach and the agent approach. It shows that the utilizations of the CPU were close between the two approaches.

Table 1: Characteristics of testing datasets

Dataset	#Instance	Size (byte)
DS1	100,000	15,373,260
DS2	200,000	30,746,520
DS3	300,000	46,119,780
DS4	400,000	61,494,903
DS5	500,000	76,866,300
DS6	600,000	92,239,560
DS7	700000	107,612,820
DS8	800,000	122,986,080
DS9	900,000	138,359,340
DS10	1,000,000	153,732,600
DS11	2,000,000	307,465,200
DS12	3,000,000	461,197,800
DS13	4,000,000	614,930,400
DS14	5,000,000	768,663,000
DS15	6,000,000	922,395,600
DS16	7,000,000	1,076,128,200

However, the java socket approach with the sender machine was better than the proposed agent approach in terms of memory usage because the agent approach used the memory of the sender machine twice to store the same dataset at the same time, once when opening the dataset by the sender agent to prepare it for sending, and once when sending the dataset to the delivery agent that used the same memory of the sender machine. Figure 6 shows the comparison of the approaches for memory usage.

On the other hand, the utilization of the CPU of the receiving machine in the java socket approach was much better than that of the agent approach because the CPU of the receiving machine was used to move the receiver agent to another container in order to collect the dataset. The observed differences between both approaches are shown in Figure 7.

Moreover, the memory usage of the receiver machine by the java socket approach was also much better than that of

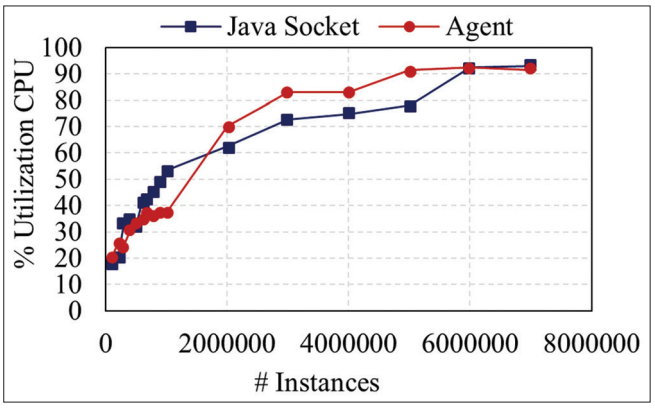
Table 2: Comparison of the consumption system resources between proposed method and java socket

Dataset	Sender				Receiver			
	CPU		RAM		CPU		RAM	
	Socket	Agent	Socket	Agent	Socket	Agent	Socket	Agent
DS1	18±4.08	20.3±2.58	0.3±0.48	1±0	8.9±1.79	6.1±0.88	1.5±0.71	2.9±0.32
DS2	20±6.62	25.7±8.12	0.9±0.32	2±0	10±2.98	10.4±1.26	2.8±0.63	10±0
DS3	33±7.83	25±3.97	1.7±0.48	3±0	11.3±0.82	19.5±5.36	3.9±0.57	12.8±1.32
DS4	34.6±1.9	30.4±6.11	1.9±0.32	3.7±0.48	18.4±28.33	20.4±0.7	5.2±0.42	13.4±0.52
DS5	32.1±6.01	33.7±2.26	1.2±0.42	7±0	14.9±1.73	23±4	7.6±0.52	16±0
DS6	41.3±5.87	35.1±4.75	2.8±0.42	7±0	14.6±4.72	25.5±3.6	8.4±0.52	17±0
DS7	42.3±5.33	36.8±3.55	3±0	6.1±0.32	14.1±1.29	23.6±4.22	8.8±1.03	18.8±0.63
DS8	44.7±8.78	36.6±5.25	3.8±0.42	7±0	15.1±2.69	25±5.85	11±1.33	20.6±1.17
DS9	49.6±9.05	37.3±2.41	3.9±0.32	7±0	17.4±2.12	32.5±5.3	12.6±0.84	20.3±0.95
DS10	53.4±3.17	37.3±3.77	4±0	10±0	18±4.59	30.8±5.01	13.1±0.88	20.8±0.63
DS11	62.6±3.27	70.2±6.88	7.9±0.32	19.6±0.52	27.5±3.06	53.9±7.2	21.2±5.31	31±0
DS12	72.6±8.6	83±4.9	13.9±0.32	20.4±2.67	39.6±6.24	52.9±19.02	40.4±0.7	42.1±14.81
DS13	74.6±10.18	83.1±5.82	13.4±0.52	33±0	47.5±6.98	63.2±1.69	47±7.32	62.9±1.29
DS14	77.9±12.73	91.4±3.17	15.3±0.67	35.2±3.29	46.8±10.45	65.8±9.2	48.3±0.48	64.4±2.95
DS15	92.4±2.76	92.5±2.55	24.9±3.84	39.4±4.06	50.7±8.59	66.1±1.91	54.1±0.74	65.8±0.63
DS16	93.1±2.38	91.6±2.27	26±0	47.5±2.76	60	70.4±9.4	54±0.67	66±0

Table 3: Comparison of latency time (s) between proposed method and java socket

Dataset	Latency (±σ)		Best case		Worst case	
	Socket	Agent	Socket	Agent	Socket	Agent
DS1	1.398±0.011	2.114±0.244	1.383	1.718	1.420	2.317
DS2	2.79±0.033	3.386±0.198	2.738	3.224	2.841	3.709
DS3	4.13±0.02	5.015±0.191	4.104	4.715	4.157	5.250
DS4	5.44±0.025	6.448±0.205	5.396	6.164	5.475	6.691
DS5	6.898±0.02	8.045±0.238	6.858	7.901	6.924	8.536
DS6	8.296±0.052	9.568±0.297	8.218	9.375	8.402	9.929
DS7	9.614±0.076	10.838±0.447	9.576	10.809	9.829	11.387
DS8	11.154±0.033	12.489±0.329	11.093	12.028	11.196	13.009
DS9	12.533±0.121	13.978±0.366	12.315	13.813	12.754	14.380
DS10	13.905±0.129	15.376±0.281	13.780	15.310	14.155	15.920
DS11	28.922±1.658	32.322±0.212	28.132	31.962	33.610	32.525
DS12	41.066±0.081	47.007±0.649	40.966	46.630	41.250	48.237
DS13	54.437±0.171	66.14±0.64	54.268	65.426	54.750	65.997
DS14	67.68±0.081	145.883±19.706	67.588	119.207	67.872	160.665
DS15	135.129±20.079	163.452±5.106	106.830	151.917	196.293	170.878
DS16	173.437±38.951	247.578±17.546	122.602	224.103	271.752	271.994
P value	0.027366138		0.030463128		0.162722326	

Figure 5: Utilization of CPU of sender machine



the agent approach. Figure 8 compares the memory usage between the java socket approach and the agent approach in the receiver machine. With regard to the latency of sending the dataset between the machines, it was observed that the best and worst time intervals for the exchange of data between the two approaches were very close, and this was emphasized by the t-tests that were conducted, where the t-test of the best case was 0.030463128, while the t-test of the worst case was 0.162722326. Figure 9 compares the best and worst latency times between the approaches.

It was observed that the best cases of latency for datasets with sizes of <4,000,000 instances were very close, while the latency times started to differ after that. In the worst case, the latency time of the agent approach was sometimes better than that of the java socket approach. In general, the differences in latency between both approaches were not significant, where the t-test for them was 0.027366138.

Based on the above results, the proposed communication method was regarded as having given a good performance in the exchange of data between JADE agents according to the important feature provided by JADE with regard to high security that is not provided by java socket.

Figure 6: Memory usage of sender machine

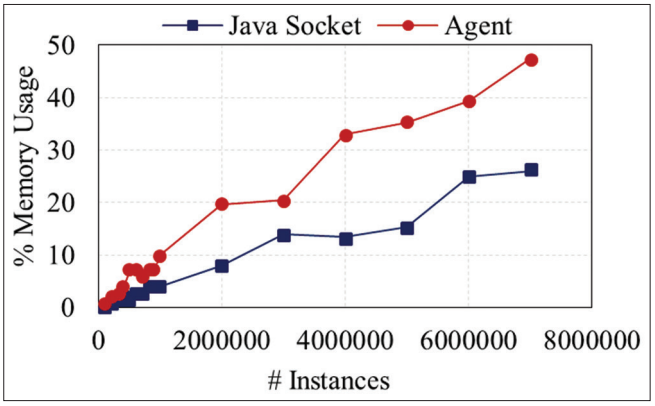


Figure 7: Utilization of CPU of receiver machine

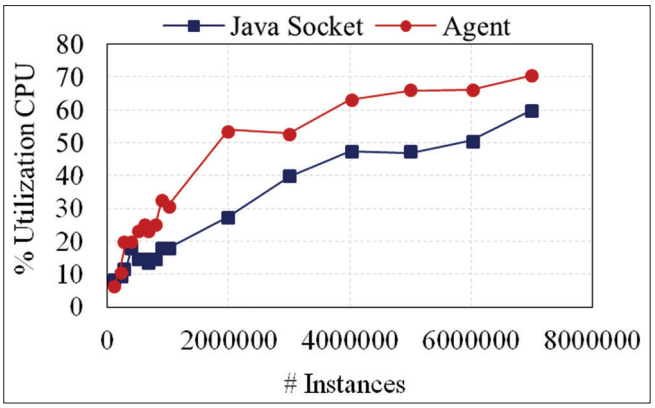


Figure 8: Memory usage of receiver machine

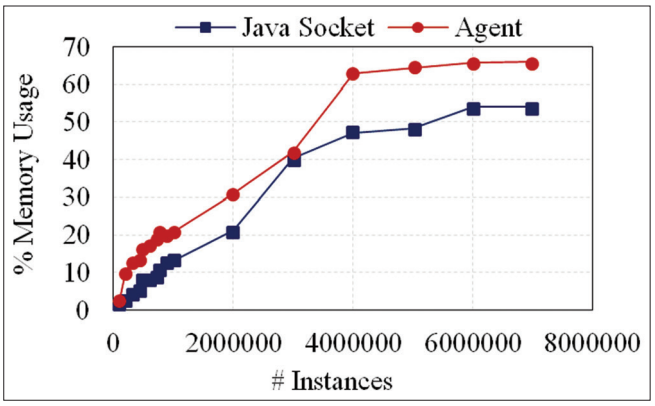
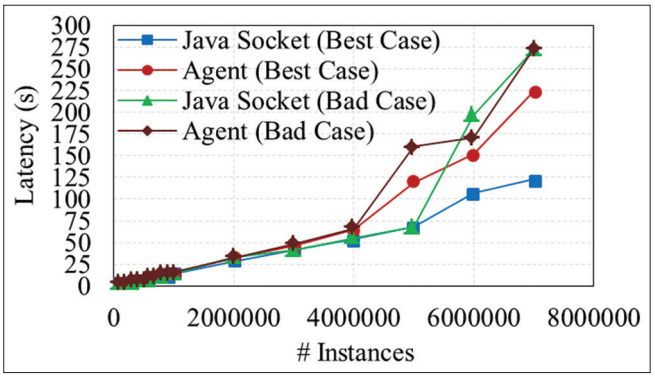


Figure 9: Latency time to send a dataset between machines



5. CONCLUSION AND FUTURE WORK

The majority of the communication languages based on FIPA are able to send high level data between agents, but there are no formal methods or scenarios for the sending of low level data (e.g., videos, audios, etc.) between agents in MASs. This paper proposed an efficient communication method based on JADE for the exchange of large low level data between agents in different environments. The results proved that the proposed method has good performance with high security for the exchange of data in MASs compared to a popular approach for the exchange of data in java that uses a socket class (non-agent and less secure).

In the future work, we will attempt to improve the communication between agents to become more faster by suggesting a java secure method to exchange data between agents with high security.

REFERENCES

AgentBuilder, An integrated software toolkit to develop intelligent software agents and agent-based applications. Available from: http://www.agentbuilder.com/.

Bellifemine, F., Bergenti, F., Caire, G. (2005), JADE - A Java agent development framework. Multi-Agent Programming. US: Springer. p125-147.

Berna-Koes, M., Illah, N., Sycara, K. (2004), Communication efficiency in multi-agent systems. Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on. Vol. 3. IEEE.

FIPA-OS, A component-based toolkit enabling rapid development of FIPA compliant agents. Available from: http://www.fipa-os.sourceforge.net/index.htm.

Garro, A., Palopoli, L. (2002), An xml multi-agent system for e-learning and skill management. Net. Object Days: International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World. Berlin, Heidelberg: Springer.

JADE. An open source platform for peer-to-peer agent based applications. Available from: http://www.jade.tilab.com/.

Jang, M.W., Amr, A., Gul, A. (2004), Efficient agent communication in multi-agent systems. International Workshop on Software Engineering for Large-Scale Multi-Agent Systems. Berlin, Heidelberg: Springer.

Kumar, S., Kumar, U. (2014), Java agent development framework. International Journal of Research, 1(9), 1022-1025.

McArthur, S.D.J., Davidson, E.M. (2004), Multi-agent systems for diagnostic and condition monitoring applications. Power Engineering Society General Meeting, 2004. IEEE.

Nguyen, G., Dang, T.T., Hluchy, L., Laclavik, M., Balogh, Z., Budinska, I. (2002), Agent Platform Evaluation and Comparison. Rapport Technique. Bratislava, Slovakia: Institute of Informatics.

Noroozi, A. (2009), A Novel Model for Multi-agent Systems to Improve Communication Efficiency. Computer Engineering and Technology, 2009. ICCET'09. International Conference on. Vol. 2. IEEE.

Retsina, R. Available from: https://www.cs.cmu.edu/~softagents/retsina_agent_arch.html.