The Scalability of a Multi-Agent System in Security Services.

Korba, Larry; Song, Ronggong

https://doi.org/10.4224/5765692

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC-CNRC

*The Scalability of A Multi-agent System in Security Services**

Song, R. and Korba, L.
August 2002

* published in NRC/ERB-1098 August 2002, 23 Pages. NRC 44952.

Canada

National Research Council Canada
Conseil national de recherches Canada
Institute for Information Technology
Institut de Technologie de l'information

ERB-1098

NRC·CNRC

The Scalability of A Multi-agent System in Security Services

Ronggong Song and Larry Korba
August 2002

Canada

NRC 44952

National Research Council Canada
Institute for Information Technology

Conseil national de recherches Canada
Institut de Technologie de l'information

The Scalability of A Multi-agent System in Security Services

Ronggong Song and Larry Korba
August 2002

The Scalability of A Multi-agent System in Security Services*

Ronggong Song Larry Korba
Institute for Information Technology,
National Research Council of Canada
E-mail: {Ronggong.Song, Larry.Korba}@nrc.ca

Abstract

Security services become important when developing practical multi-agent e-commerce systems. We first provide an overview of the multi-agent system scalability and model a multi-agent application system -- a labor market case based on the JADE multi-agent platform. We then simulate the scalability of the multi-agent application system in security services, and analyze their effect. Finally, we offer additional research on the adaptation and self-organization of the multi-agent system to make the system better able to cope with large numbers of agents and to be more efficient.

1. Introduction

Multi-agent systems have been identified as a programming paradigm that allows flexible structuring of distributed computation over the Internet. This new paradigm proposes solutions to highly distributed problems in dynamic, open computational domains. It is expected that they will take an important role in the future information society and especially in e-commerce applications. Unfortunately, most multi-agent systems that have been built so far involve interactions between relatively small numbers of agents. When multi-agent systems are employed in larger applications, what issues of scaling need to be considered? In an e-commerce or e-government application employing negotiating agents, is it possible to determine how many agents can be supported without significant delays to the user? These are the scalability questions. Scalability has become an important issue in the successful deployment of multi-agent software, since performance requirements can change over time. Therefore, it is often expected that the multi-agent software can be scaled up or down to ensure the desired performance at minimal costs.

On the other hand, security services become important when developing practical multi-agent e-commerce systems. However, security and privacy protection for multi-agents is still a young discipline. In this paper, we provide an overview of the multi-agent system scalability and model a multi-agent application system -- a labor market case based on the JADE multi-agent platform. We then simulate the scalability of the multi-agent application system in security services and analyze their effect on the multi-agent system scalability. Finally, we offer additional research on the adaptation and self-organization of the multi-agent system to make the system better able to cope with large numbers of agents and to be more efficient.

The rest of the paper is organized as follows. The background about multi-agent system scalability is briefly reviewed, and a labor market multi-agent application system is proposed in the next section. In Section 3, the scalability in the security services is simulated and analyzed. In Section 4, the scalability of the multi-agent system is simulated for the different organization forms. In Section 5, some concluding remarks and directions are presented for further research.

* NRC Paper Number 44952

2. Overview of Multi-agent System Scalability

In this Section, we first introduce the concept and questions of scalability described in [26], and then analyze the factors that impact multi-agent system scalability and discuss the technologies and methods measuring scalability and enhancing scalability.

2.1 Definition of Scalability

The problem of scalability is one with which the distributed computing community is still wrestling. Scalability problems generally manifest themselves as performance problems. According to O.Rana's description [26], scalability in its most general form is defined as: "the ability of a solution to a problem to work when the size of the problem increases". Actually, scaling can be performed up or down. A program is considered able to scale up, if the addition of certain resources (e.g. memory, processor) will lead to an increase in performance. A program can be considered able to scale down if it is possible to remove/reduce the access of certain resources (e.g. less memory, slower processor). For example, with the rise of small and mobile computing devices the problem often arises if and how software can be scaled down to run in a more resource-constrained environment. While it is important to realize that systems can also scale down, scaling up is the by far most often discussed approach. In this report, the scalability we discuss is focused on scaling up.

According to multi-agent communications, multi-agent systems will need to scale in a number of different dimensions as follows.

- The total number of agents involved increasing on a given platform.
- The total number of agents involved increasing across multiple systems or platforms.
- The size of the data upon which the agents are operating is increasing.
- Increasing the diversity of agents.

In the first two scenarios, the total agent density and the resulting effect on system performance can be determined in terms of metrics associated with a particular platform or an operating environment. These metrics include memory usage, scheduling/swapping overheads (for active agents), cloning an agent, or dispatching an agent to a remote site (with mobile agents) and are often the only metrics reported for agent performance and scalability [15].

If agent density is increased to obtain better performance, we can measure the relative speed-up, giving us the net gain in benefit by increasing the number of agents, as often quoted in parallel computing literature [35]. According to Amdahl's law and O.Rana's description, the slower agents will limit the gained speed-up in the system or agents, which are forced to operate sequentially. Thus, the achievable speedup is nearly linear when the problem size is increased as more agents are added.

Increases in agent diversity also have a corresponding effect on agent density, and in this case scalability management is related to methodologies for agent analysis and design. Software engineering approaches for developing agent communications, which extend beyond existing approaches based on object-oriented modeling techniques, or knowledge engineering approaches are needed.

Scalability can also be stated in terms of co-ordination policies in agent communities, in terms of the total number of message exchanges necessary to converge on a solution. Either agents can be grouped to limit message exchanges, or a global utility function may be specified. Various co-ordination policies are discussed in [34], with the co-ordination problem described as composing some objects (tasks, goals, decisions and plans) with respect to some overall direction (goal, functionality), with several actors involved in the co-ordination process. The necessity to converge to a global optimum as emphasized by game theoretic and economic models also has an impact on performance, and correspondingly scalability.
Another set of approaches for managing scalability is related to modeling agent systems to predict performance. Various approaches exist, one of which is based on the concept of a messenger paradigm. A survey can be found in [13].

Actually, many scalability problems in large-scale, agent-based systems, are related to limited scalability of searching and matching facilities. Unfortunately, some of these problems are inherently non-scalable and can be tackled only by considering the application for which agents are developed, for example, some specific services in multi-agent systems such as naming services, location services and directory services, etc.

In addition, the term "scalability" is not always used to refer to architecture, service and performance. In some cases it is used to refer to scalable functionality. For example, the SAIRE approach [21] claims to be scalable because it supports heterogeneous agents. Shopbot [38] claims to be scalable because its agents can adapt to understand new websites. In these cases, we think the term extensible functionality would seem to be more appropriate than the term scalability.

2.2 Factors Affecting Multi-agent System Scalability

Two main factors can be identified, that impact the scalability of a multi-agent system: load and complexity. Here the load mainly includes memory load, CPU load and communication load. Memory load means the amount of memory occupied by the system and agents. CPU load means the CPU usage including processes, threads, special computations for security and privacy, intra-container communications, etc. Communication load means message routing load because of message passing as agents' major form of communication. Mostly, these loads affect each other and affect the response time to the request. Even a medium-sized multi-agent system with only a few hundred agents is already a significant workload, which would require the use of several high end PC or a multi-processor WS.

When dealing with the amount of agents in the memory and the CPU usage it is important to distinguish between reactive and proactive agents. A reactive agent will only be executed if it receives a message. Consequently a reactive agent will only consume processor time if it computes a response to an incoming message. Therefore increasing the numbers of reactive agents is predominantly a memory (agent storage) problem. Large multi-agent systems consisting of reactive agents tend to focus on techniques to minimize the amount of agents in the memory. Idle agents are paged out (serialized to file) and agents that receive a message are paged in (de-serialized from file). G.Yamamoto [14] demonstrates in an impressive way that this approach can enable the hosting of several hundred thousand agents. Unlike reactive agents, proactive agents don't need messages to start actions. By themselves, they can initiate complex tasks like the discovery of new services. Each proactive agent is an object with one or more threads of control. The hosting of proactive agents requires significant memory and processor resources. Increasing the number of proactive agents leads to an increase in concurrent threads that sooner or later exceeds the possibilities of a single machine. The

distribution of processor load is a central issue in the development of multi-agent systems using the proactive agents. Each agent must have at least a Java thread in order to be able to proactively start new conversations. Thus, only if the number of concurrent threads can be distributed over different physical machines, it is possible to scale up the number of agents in a multi-agent system without risking a decreased performance of individual agents.

Agents rely on message passing as their major form of communication. Fast and reliable message delivery is hence of the utmost importance in order to avoid potentially chaotic behavior. In order to keep runtime overheads low, some multi-agent systems (e.g. JADE) uses multiple communication means for ACL message transport.

Another factor that affects system scalability is computation complexity. The computation complexity mainly relies on the algorithms used in the agent system, especially security algorithms such as authentication algorithms, cryptography algorithms and signature algorithms, etc. In addition, it is also important to note, that the scalability problem of multi-agent systems is not only a question of computational resource it is also one of software complexity.

2.3 Determining Scalability: Performance Metrics

In order to assess whether a given multi-agent system scales successfully, we need to identify metrics for measuring scalability. A scaling strategy can be evaluated from a number of perspectives, such as the performance of an individual element, the workload of a particular element, communication costs between elements, and persistence support for elements.

It is clear that we cannot provide a unique performance metric that combines all possible performance criteria, since the weighting constants are very application-dependent. Therefore, we only identify the important performance metrics. Various metrics for scalability exist [25], but in this report we only proposed two categories of metrics: (1) those related to system parameters, and (2) those related to coordination mechanisms. System metrics are generally associated with agent management on a particular host, the operations performed by an agent, and the transfer of agents or messages between hosts. The main system metrics we discuss here are the performance characteristics of the agent server. They may be as follows.

(1) CPU usage and memory requirement as agents density is increased;
(2) Number of searches per second as agent density is increased;
(3) Interaction time as agent density is increased;
(4) Response time to the request as the number of simultaneous messages sent is increased.

The number of agents kept in memory is determined by the memory size, which greatly affects performance. The number of processes per second measures the factors affecting performance: the total memory size and individual agent size. The number of searches per second (throughput) measures the system scalability in relation to the number of user agents and job agents. The response time to request measures the system scalability in relation to the number of simultaneous messages sent.

On the other hand, choice of a suitable coordination mechanism is essential to support scalability in agent systems. Coordination can be pre-defined, by a "Conversation Policy" [23, 16]. The objective being to

predetermine the number of message transfers between agents, or to minimize conflict between agents, so that an agent community can converge to a given outcome faster. Both scenarios will facilitate an increase in agent density, with better reinforcement algorithms contributing towards improved performance [33]. Metrics associated with coordination policies may be as follows.

(1) Total number of messages transferred between agents;
(2) Total number of agents involved;
(3) Maximal distance between agents involved;

Metrics may also be related to conversation and privacy policies, such as:

- The total number of simultaneous conversations supported,
- The response time between conversations,
- The algorithms that will be used for anonymous protection, etc.

2.4 Modeling A Multi-agent System Scalability

2.4.1 A Labor Market Application Overview

In coming years it is expected that the matching of supply and demand in the labor market will increasingly be performed through such (Internet-based) intermediaries. Thus, electronic labor market is a natural domain for testing security and privacy protection and system scalability. It involves large number of end-users and online labor market. Our particular scenario involves agents that encapsulate the basic requirements and some personal data of the end-user and job match agents.

End-users' goals for their job search are presented to the user agents. The end-user sends his/her searching requirements to the user agent via web browsers. The user agent then sends the requirements and some personal data of the user to the job match agents according to the user's privacy policy. When matching the user's personal information with the available functions, a match must be made between the user profile and the demands of the party requiring personnel. In this process, data will be exchanged in a security and anonymous way.

Job match agents compare the user's data with the job database. If some jobs match the user's requirements, the job agent will send the job information to the user agent. Otherwise the job agent will send some recommended jobs or message such as no job available currently to the user agent after searching its job database. Sometimes the job match agents may need to negotiate the user agent to get more personal information of the user.

In the labor market case (LMC), the system hosts at least one job agent on the server, but may host many user agents. Each job agent wraps a job database in order to obtain job data from the database. User agents can also move among different agent containers and platforms to obtain job information from all the job agents in the community. Figure 2.1 depicts one of the possible models for the architecture of LMC.

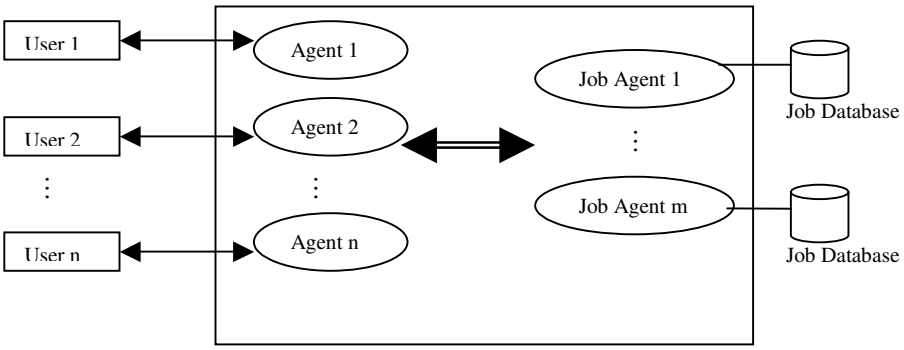

Figure 2.1. Architectural overview of a labor market case.

2.4.2 Agents Interaction and Control

An agent processes jobs by interacting with other agents. To interact with other agents, an interaction protocol is required. The interaction protocol defines the message format and the attributes of jobs. The format of a message consists of the name of the message, the name of the parameters and the types of the parameters.

The order of messages is also important, because agents need to exchange several messages in order to interact with each other. For example, in LMC, in an exchange between a user agent and a job agent, the user agent first sends a request message to the job agent, and the job agent then returns a matching result message followed by a recommend result message. We call this kind of message sequence a session. An interaction diagram is illustrated in Figure 2.2.

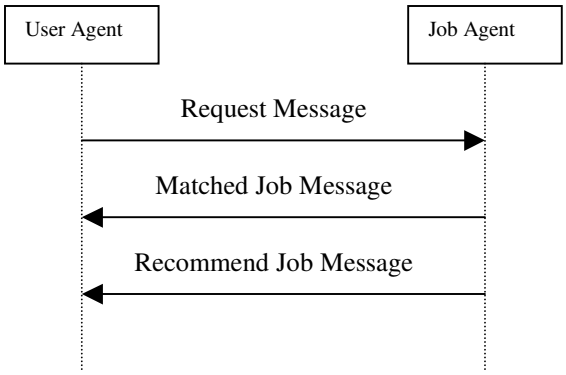

Figure 2.2. Interaction diagram between user agent and job agent.

In LMC system, since each user has own agent on a server and each agent lives for half or one year, there may be tens of thousands of agents running on a LMC server. On the other hand, the size of a user agent is not small, so cumulatively, these agents occupy considerable memory. Moreover, each agent has a thread for processing jobs. If too many threads are running concurrently a system overload may occur. Therefore, a control mechanism for memory and threads occupied by agents is one of the key issues for servers that host thousands of agents.

Agent scheduler controls the amount of memory occupied by agent by keeping agents in secondary storage. It also controls the number of threads running concurrently by scheduling the activities of agents.

On the other hand, since the user agents in LMC system may be mobile, it is possible that many of them may come to a LMC server in a short period, causing a system overload. To avoid such problems, we also need a middleware and mechanism for controlling the flow of agent movements.

2.4.3 Modeling Scalability

We identify a framework for modeling agent communication, which may extend agent design and analysis methodologies, such as [15]. Our modeling scheme provides separate models for system and coordination parameters, and can be used to hierarchically construct models of multi-agent communities.

For system parameters, we model CPU usage (performance), memory requirement and throughput of searches as user and job agent density is increased on a host. We also need to model the response time to the request.

In addition, in order to obtain the complexity cost for interaction between user agents and job agent, we uses Petri nets (PN) approach and a notation based on PN, because they provide a robust tool for modeling the data flow mechanisms present within distributed systems. Also theoretical results concerning PN are plentiful, and numerous tools are available for a quantitative analysis of such discrete event-based systems.

We model the labor market case with a Petri Net in Figure 2.3. A place-transition pair represents each user agent, with outgoing messages from the user agents modeled by an immediate transition and messages from Job agents modeled by timed transitions. The parameters $\alpha_1, \alpha_2, \ldots, \alpha_n$ represent computing delays associated with request-message, security and privacy processing. The parameters $\beta_1, \beta_2, \ldots, \beta_n$ represent messaging delays from user agents to the job agent. The parameters $\delta_1, \delta_2, \ldots, \delta_n$ represent searching delays associated with search, match, security and privacy processing. The parameters $\gamma_1, \gamma_2, \ldots, \gamma_n$ represent messaging delays associated with timed transitions, indicating message delays from the job agent to user agents.

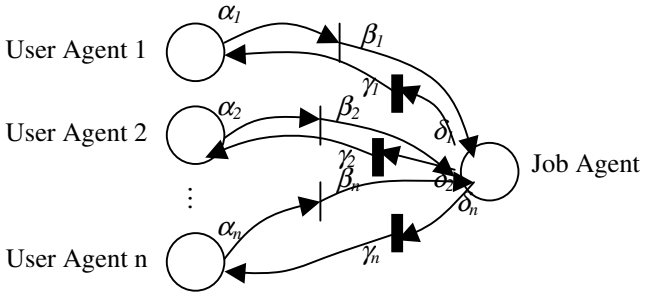

Figure 2.3. Petri Net: n user agents and a job agent.

The objective of building the Petri net is to obtain the complexity cost for interaction between the user agents and the job agent, and this we define as:

$$Complexity = \sum_{k=1}^{n} (a_k \alpha_k + b_k \beta_k + c_k \delta_k + d_k \gamma_k)$$

Where $a_k, b_k, c_k, d_k = 0$ *or 1*. If all *a, b, c, d* equal 1, the *complexity* represents the total complexity cost for all agent actions.

3. The Scalability in Security Services

In this Section, we first introduce our simulation tools, environment and simulation parameters, and give the simulation results based on the above models for the labor market application in the JADE multi-agent platform. We then analyze and compare their scalability from their performance results.

3.1 Simulation Tools and Environment

We use JAVA as the programming language, the JADE 2.5 platform as the multi-agent platform, and the Java™ 2 Platform, Standard Edition (J2SE™) version 1.3.1_01 and 1.4.0 as the essential Java tools and APIs for developing the simulation applications.

The tests here are done on three computers. One computer is Intel Pentium 3 (named NC12950, Laptop), its CPU is 700 MHz with system memory of 256 MB, and the operation system is Windows 2000. The second computer is Intel Pentium 4 (named NC84), its CPU is 1.50GHz and Memory is 256 MB, and the operation system is Windows 2000. The communication between the two computers is 100Mbps local Ethernet.

The tests are based on the idea that the users' agents send the query-request messages to the job agents, and the job agents then send the replay messages to these users' agents. Since we do not have a sample job database, and since we believe that the speed of the database search depends on the database platform and the search mechanism, in this work we don't deal with the processing time of the database, i.e., we use a time of zero for the database processing time. We only measure the processing time of the agents for request and reply.

3.2 Simulation Parameters

In the labor market application system, the important scalability parameters should be user size, job size, job search and match speed, response time for a query, computing complexity for privacy and security processing, CPU usage and memory requirement, etc. Since we don't consider the processing time of the job database here, the simulation parameters don't include job size, and search and match speed in this simulation.

Thus, in our simulation, the simulation parameters include user size, average response time for a query in the user agent side, average process time for a query in the job agent side, CPU and memory usage, computing complex cost for privacy and security processing. The parameters are defined as follows.

- User size: the number of the users (senders).
- T_Time: total processing time for all request and reply messages in both users' agents and job agent side.
- CPU usage: the percentage of CPU used during the agents work.

- Memory usage: the percentage of Memory used during the agents work.
- Computing complexity cost for privacy and security processing: the processing time to support the privacy and security function.

3.3 Simulation Results

The basic algorithms supporting privacy and security protection include public key cryptography, symmetric key cryptography, hash function, signature and blind signature algorithm, (among others). Although there are many factors which may effect on the system scalability when we use the privacy and security protection functions, e.g. protocol complexity, cryptographic or other management, algorithmic complexity, etc., we believe the key factor is the computational complexity of cryptography algorithms. Public key cryptography and signature algorithms have much marked effect on the system scalability than other cryptographic algorithms. On the other hand, since we do not have the detailed information concerning the mechanisms and algorithms supporting privacy and security protection for LMC, our simulation focuses on two examples: the effect of the public key algorithm and symmetric key algorithm. In the first situation, we test the effect of the RSA public key algorithm on the system scalability when it is used to PET services (e.g. signature or blind signature). In the second situation, we test the effect of the IP ESP+AH tunnel on the system scalability when it is used to data integrity and encryption using 3DES and MD5, and address integrity using SHA1.

3.3.1 Effect of the RSA public key algorithm

The testing was performed on the PC: NC12950. The users' agents and job agent were run in the same main container. In order to test the effect of the RSA public key algorithm on the system scalability, we made a comparison of the testing with and without RSA and without RSA.

In this test, one user only has one user agent, and each user agent only sends 50 request messages to the job agent. In the simulation with RSA algorithm, each message is signed using RSA signature algorithm with a 1024 bit mod by the user agent and verified by the job agent. We test that one time RSA signature needs to spend about 50 milliseconds in the PC: NC12950. We simulate the results for one user agent, two user agents, …, 128 user agents, respectively. The test results are as follows. Table 3.1 depicts the total processing time for the request and reply messages with RSA signature.

Table 3.1: The total processing time for request and reply messages with RSA algorithm.

User agents	1	2	4	8	16	32	64	128
Messages	50	100	200	400	800	1600	3200	6400
T_Time (ms)	2484	4887	9664	19288	38696	76840	153271	307682

Figure 3.1 depicts the CPU and Memory usage for the request and reply messages with the RSA algorithm. Considering CPU usage history, the first pulse is the simulation result for one user agent sending 50 request and reply messages, the next pulse is for two user agents sending 100 request and reply messages and each user agent sending 50 messages, ..., the last pulse is for 128 user agents sending 6400 request and reply messages and each user agent sending 50 messages. In addition, each pulse has two peaks in Figure 3.1, and the first peak is the CPU usage for the JADE platform start-up and the second peak is the CPU usage for sending the messages with RSA algorithm. Since the user agent size still is small, the increase of the agents only has a little effect on the Memory usage.

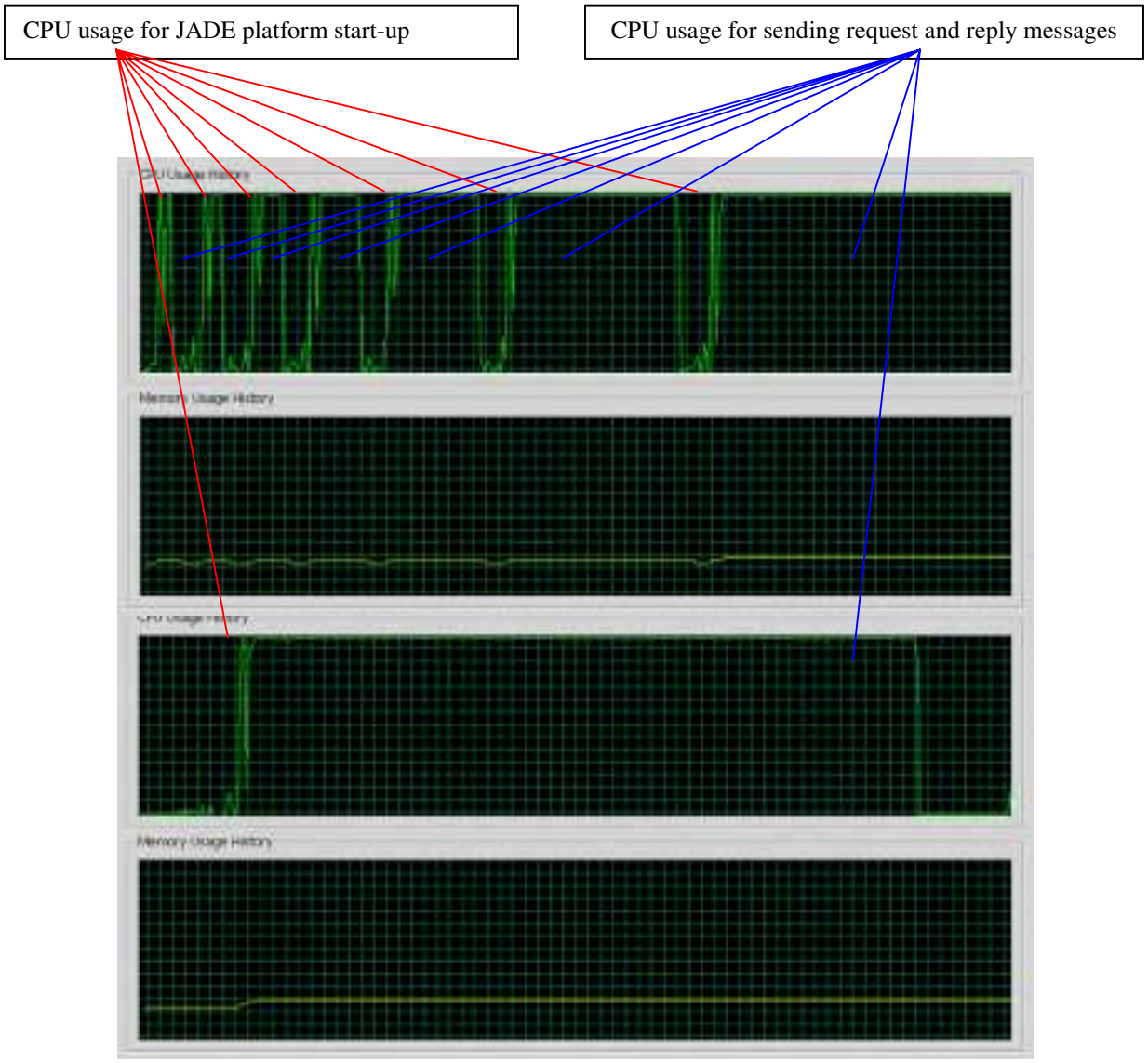

Figure 3.1. CPU and Memory usage for the request and reply messages with RSA.

In order to evaluate and compare the effect of the RSA algorithm on the system scalability, we also tested the above situation without RSA algorithm. Table 3.2 depicts the total processing time for the request and reply messages without RSA signature.

Table 3.2: The total processing time for request and reply messages without RSA algorithm.

User agents	1	2	4	8	16	32	64	128
Messages	50	100	200	400	800	1600	3200	6400
T_Time (ms)	70	80	171	321	651	981	1842	2724

Figure 3.2 depicts the CPU and Memory usage for the request and reply messages without RSA algorithm. When considering CPU usage history, the first two pulses are the result of one user agent sending 50 request

and reply messages, the next two pulses are for two user agents sending 100 request and reply messages and each user agent just sending 50 messages, ..., the last two pulses are for 128 user agents sending 6400 request and reply messages and each user agent sending 50 messages. In addition, each pulse has two peaks in Figure 3.2, and the first peak is the CPU usage for the JADE platform start-up and the second peak is the CPU usage for sending the messages without RSA. Since the user agent size is small, the increase of the agents only has a little effect on the Memory usage.

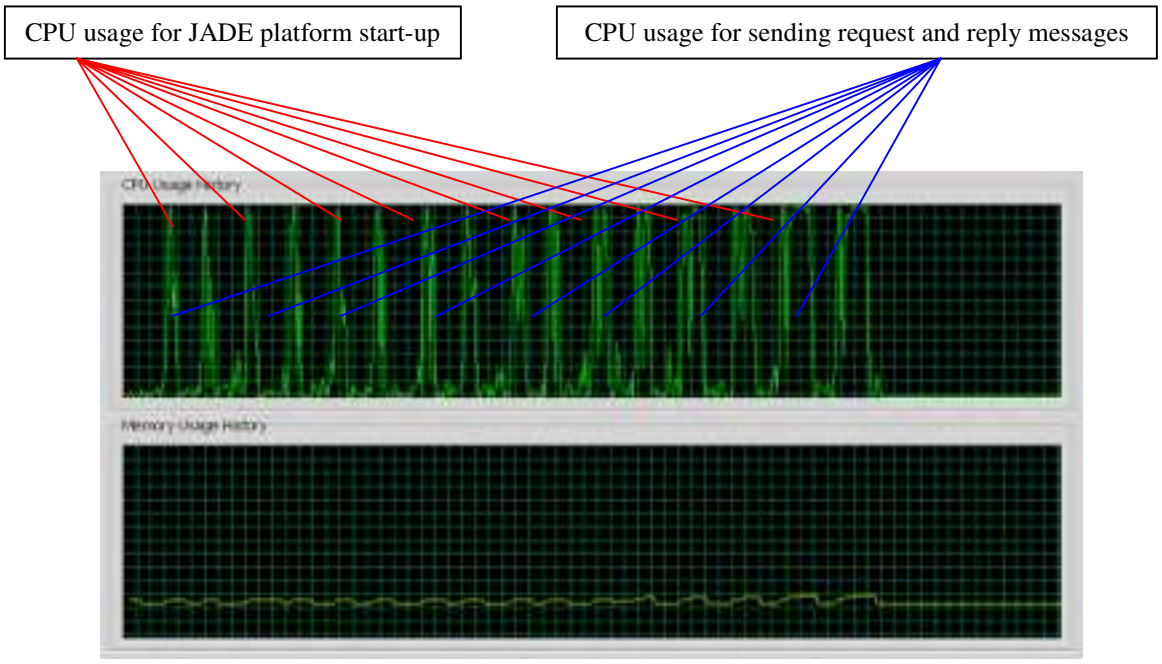

Figure 3.2. CPU and Memory usage for the request and reply messages without RSA.

Figure 3.3 depicts the comparison of the total processing time for sending request and reply messages without RSA and with RSA.

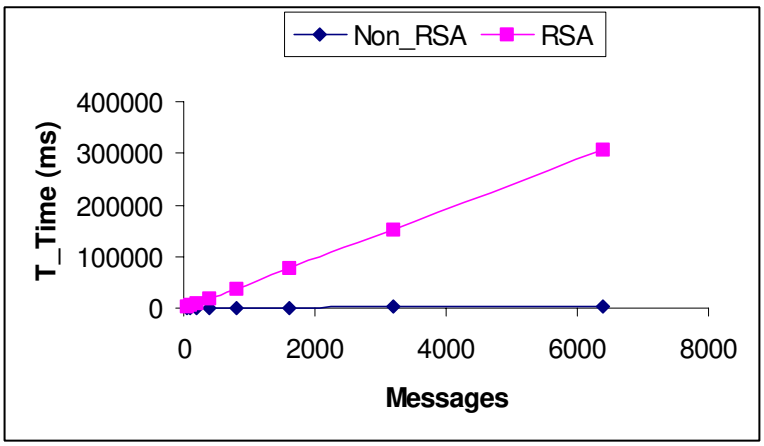

Figure 3.3. The comparison of the T Time for the above situations.

Our conclusions for the above tests are:

- Public key cryptography would have a severe effect on the system scalability if it were used to the confidentiality or authentication of the data transmission.
- The T_Time increases linearly with the increase of the number of messages that are processed using public key system.

3.3.2 Effect of the IP ESP+AH tunnel

The testing was performed on PC: NC12950 and NC84. The users's agents and job agent are run within the same platform. The users' agents are run in the non-main container of the NC12950, and the job agent is run in the main container of the NC84. In order to test the effect of the IP ESP+AH tunnel on the system scalability, we provide a comparison of the testing both with and without IPSec Tunnel.

In this simulation, one user only has one user agent, and each user agent only sends 50 request messages to the job agent. In the simulation with IP ESP+AH tunnel, the integrity of the address and data is protected using the hash function SHA1 for each packet, and the data of each packet is encrypted using 3DES and hashed using MD5. We produce results for one user agent, two user agents, …, 128 user agents, respectively. The simulation results are as follows. Table 3.3 depicts the total processing time for the request and reply messages with IP ESP+AH Tunnel in the PC: NC12950.

Table 3.3: The total processing time for request and reply messages with IP ESP+AH Tunnel in the user agent side.

User agents	1	2	4	8	16	32	64	128
Messages	50	100	200	400	800	1600	3200	6400
T_Time (ms)	1171	1472	2544	4557	8542	16454	31786	61058

Figure 3.4 depicts the CPU and Memory usage for the request and reply messages with IP ESP+AH tunnel in the user agent side. The first pulse in the CPU usage history is the simulation result for one user agent sending 50 request and reply messages, the next pulse is for two user agents sending 100 request and reply messages and each user agent sending 50 messages, ..., the last pulse are for 128 user agents sending 6400 request and reply messages and each user agent sending 50 messages. Since the user agent size still is small, the increase of the agents only has a little effect on the Memory usage.

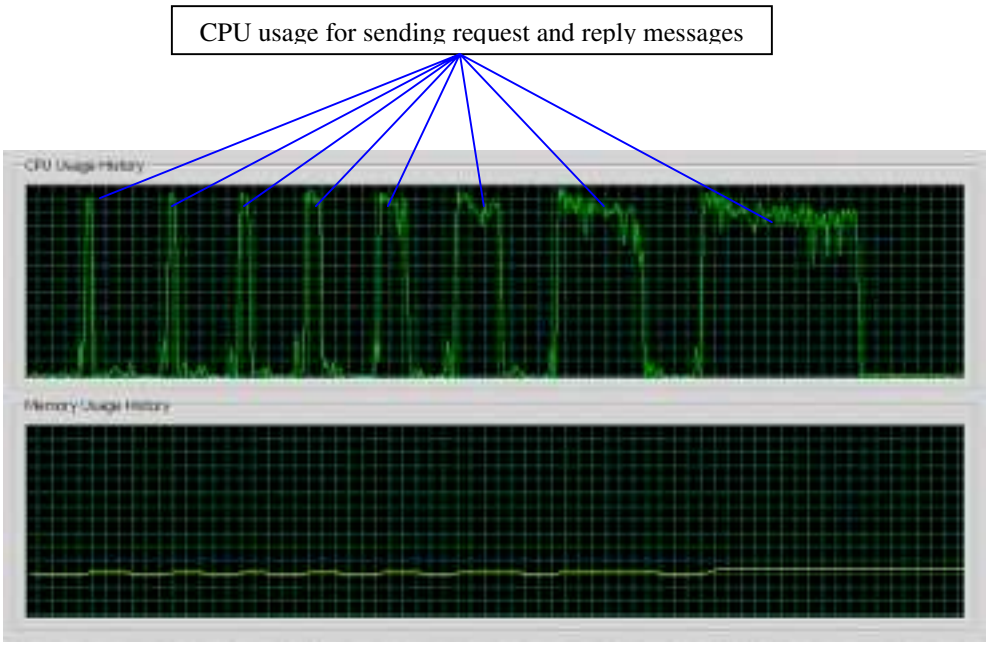

Figure 3.4. CPU and Memory usage for the request and reply messages with IPSec in the user agent side.

In order to evaluate and compare the effect of the IP ESP+AH tunnel on the system scalability, we also test the above situation without IP ESP+AH tunnel. Table 3.4 depicts the total processing time for the request and reply messages without IP ESP+AH Tunnel.

Table 3.4: The total processing time for request and reply messages without IP ESP+AH Tunnel in the user agent side.

User agents	1	2	4	8	16	32	64	128
Messages	50	100	200	400	800	1600	3200	6400
T_Time (ms)	971	1232	2133	3425	7200	13599	26999	52936

Figure 3.5 depicts the CPU and Memory usage for the request and reply messages without IPSec in the user agent side, where for CPU usage history, the first pulse is the simulation results for one user agent sending 50 request and reply messages, the next pulse is for two user agents sending 100 request and reply messages and each user agent just sending 50 messages, ..., the last pulse is for 128 user agents sending 6400 request and reply messages and each user agent sending 50 messages. Since the user agent size is small, the increase in the number of agents only has little effect on the Memory usage.

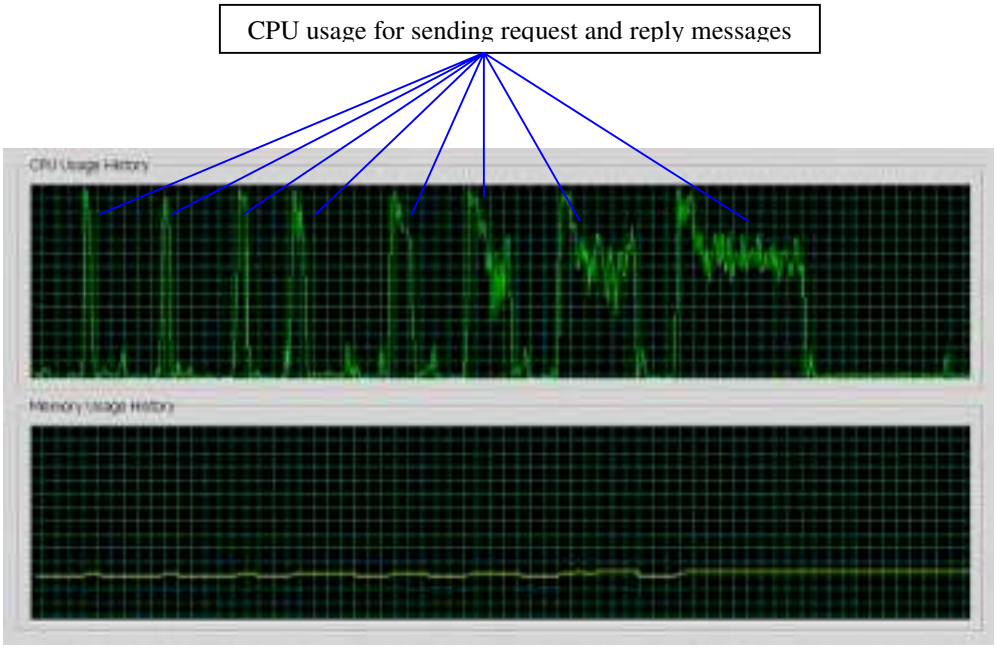

Figure 3.5. CPU and Memory usage for the request and reply messages without IPSec.

Figure 3.6 depicts a comparison of the total processing time for sending request and reply messages with and without IPSec in the user agent side.

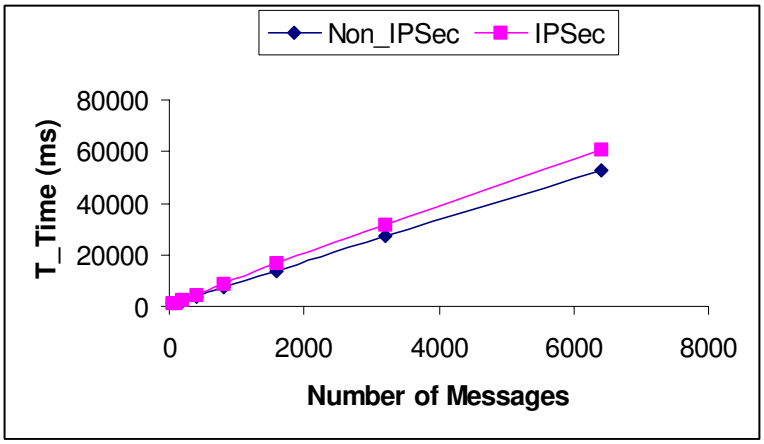

Figure 3.6. The comparison of the T Time for the above situations.

From the above tests we derive the following conclusion.

- The symmetric key cryptography and integrity function only have little effect on the system scalability if they were used to the confidentiality and integrity of the data transmission.

4. Improving the Scalability of the Multi-agent System

In this Section, we offer additional research on the adaptation and self-organization of the multi-agent system to make the system better able to cope with large numbers of agents and to be more efficient. To cope with this, the best and more generalized approach is to allow agents to build and maintain their own organizational structures and enable them to dynamically change between them. We believe a MAS is more scalable if it can both operate with different population sizes and deal with dynamic changes to population during operation.

Turner and Jennings proposed techniques for increasing scalability through changes in the agent system environment in [28]. According to this approach, agents in such systems need to be self-building and adaptive.

- Self-building: That is, they must be able to determine the most appropriate organizational structure for the system by themselves during run-time.
- Adaptive: That is, they must be able to change this structure as their environment changes.

In practice, this means enabling the following two kinds of features.

- Allow two or more agents to compose and decompose when organizational load is heavy or light respectively.
- Enable a group of agents to create or destroy an extra agent playing the role of intermediary to undertake collective tasks. The alternative to having an intermediary for these tasks is to select one of their numbers to take on the collective tasks.

4.1 Testing Environment

In this testing, we assume each user has several available jobs that are owned by one of the job agents, and each job agent has different job database with other job agents. The tested user agents are from 200 to 1000 and job agents from 1 to 10, and they are run in the same machine (Laptop: NC12950) and the same container. In addition, since the searching time is related to the database software and the database size, we assume the searching time is 100ms for searching and matching one job database (based on high-performance software design, which is instructed by Professor C. M. Woodside, one operation of database server takes 85ms). The specific metrics used herein to compare scalability are as follows (see Figure 4.1).

(1) W_Time: The average waiting time when the jobs are matched;
(2) Number of user agents: The total user agents that are run in the system;
(3) Number of job agents: The total job agents that are run in the system.

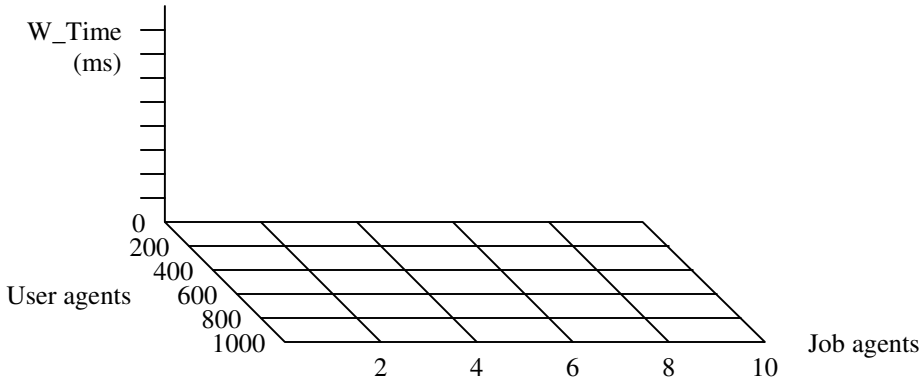

Figure 4.1. Testing metrics when the jobs are matched.

4.2 Organizational Forms and Agents Interaction

We first present several organizational forms of the labor market system (LMS) for our labor market case. These forms are different from the forms in [28] according to our LMS. In LMS, the possibility that the user agents form groups and share job information is small because of the competition between users. However, this approach would be useful for users when an intermediary agent undertakes collective tasks. These forms can be distinguished by the constraints within which the agents interact with each other. For example, whether they can share information, form co-operative groups or take action to combat inefficiency. The purpose of examining these various forms are firstly to show that different organizational forms place different resource requirements on agents, and secondly, to determine what the relationship between the resource requirements of a given form and the number of constituent agents is.

4.2.1 Case 1: Independent Agent Platform

In this case, each user agent can reason about and communicate with each job agent. However, user agents are unaware of other user agents and job agents are unaware of other job agents. Consequently, agents cannot form groups, share information, or undertake co-operative behavior. Figure 4.2 depicts this case.

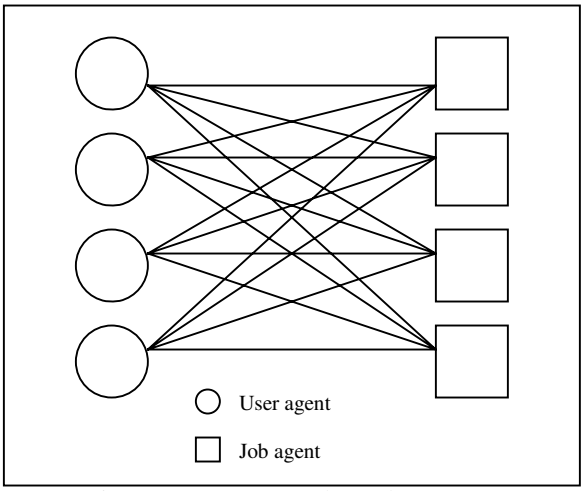

Figure 4.2. Case 1: Independent agents.

The simulation is done on the laptop: NC12950. During testing, we assume all user agents send the request messages to the job agents in the same time, and the user agents do not know which job agents are better for them. Thus, the probability that the jobs are matched is same for each job agent. Table 4.1 and Figure 4.3 depict the testing results.

Table 4.1. Average waiting time (ms) for different user and job agents in Case 1.

User Agents	200	400	600	800	1000
W_Time (ms) for 2 Job Agents	17765.29	33253.11	50685.6	63580.68	82959.39
W_Time (ms) for 4 Job Agents	35218.12	67459.99	101198.2	132049.5	168596.1
W_Time (ms) for 6 Job Agents	52097.12	101404.3	151981.8	201440.8	254541.3
W_Time (ms) for 8 Job Agents	69079.75	135657.6	203536.5	270143	340512
W_Time (ms) for 10 Job Agents	86203.1	170924.6	255000.1	339121.3	426733.1

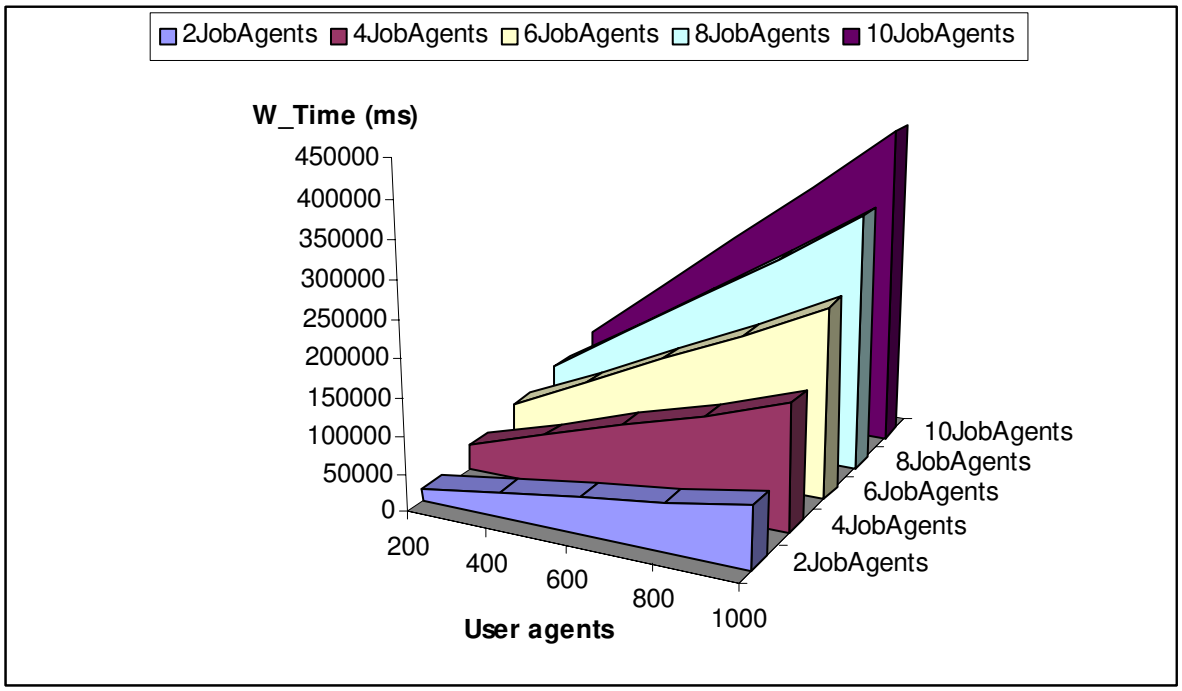

Figure 4.3. Average waiting time for different user and job agents in Case 1.

From the above simulation results, we know that the increase of the number of the user and job agents has a great effect on the average waiting time in this case. The average waiting time increases linearly with the increase of the number of the user and job agents. But this kind of organization is simple and does not require the complex inference ability.

4.2.2 Case 2: Co-operating Job Agent Platform

This case additionally allows job agents to be aware of other job agents. Therefore, job agents are able to form groups with other relevant job agents, allowing them to co-operate by sharing tasks that are commonly undertaken. For example, sharing information about job positions and categories. Thus, one job agent can give some recommended job agents to the user agent. Figure 4.4 depicts this case.

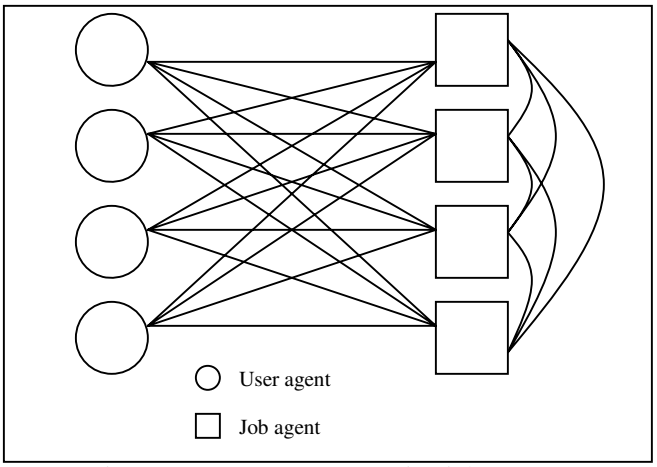

Figure 4.4. Case 2: Co-operating job agents.

The simulation is done on the laptop: NC12950. During testing, we assume all user agents send the request messages to the job agents in the same time. In this case, the user agents also do not know which job agents are better for them, but the job agents know each other. Thus they can give the user agents a good suggestion if they don't have the jobs to satisfy the user agents' requests, and we assume each job agent only give three recommended job agents in this testing. In addition, the communications among the job agents also require some resource. Table 4.2 and Figure 4.5 depict the testing results.

Table 4.2. Average waiting time (ms) for different user and job agents in Case 2.

User Agents	200	400	600	800	1000
W_Time (ms) for 2 Job Agents	17807.7	33296.45	50729.13	63628.43	83004.01
W_Time (ms) for 4 Job Agents	26735.8	50956.86	75680.33	97875.69	125454
W_Time (ms) for 6 Job Agents	26947.99	51194.77	75927.53	98135.94	125705.6
W_Time (ms) for 8 Job Agents	27201.81	51505.27	76260.64	98492.53	126053.9
W_Time (ms) for 10 Job Agents	27465.05	51861.57	76658.36	98927.4	126484.2

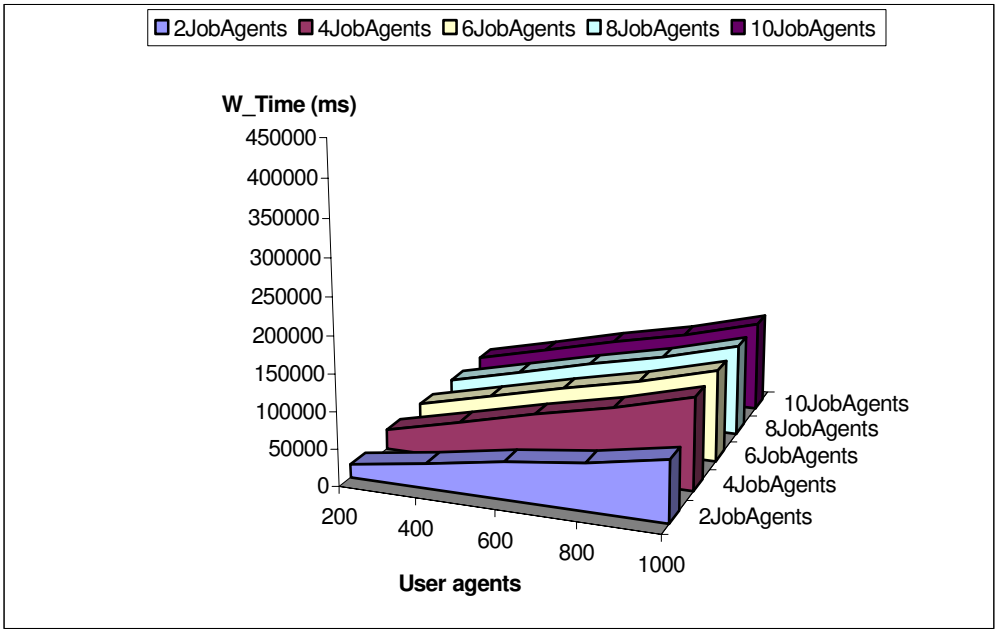

Figure 4.5. Average waiting time for different user and job agents in Case 2.

In this case, the increase of the number of the user agents also has a great effect on the average waiting time, but the effect of the job agents would reduce greatly when the number of the job agents is more than a constant number (general, it is the number of the recommended job agents). Of course, this case has better scalability than Case 1. But when the number of the job agents is too large, it still has great effect on the average waiting time since the communication among the job agents needs to take a lot of resource. Final, this kind of organization requires the job agents have a certain of inference ability.

4.2.3 Case 3: Agent Platform with An Intermediary Agent

This case is identical to case 2, with the exception that there is an intermediary agent that undertakes collective tasks. In this situation, the intermediary agent should be fair for all users. Figure 4.6 depicts this form.

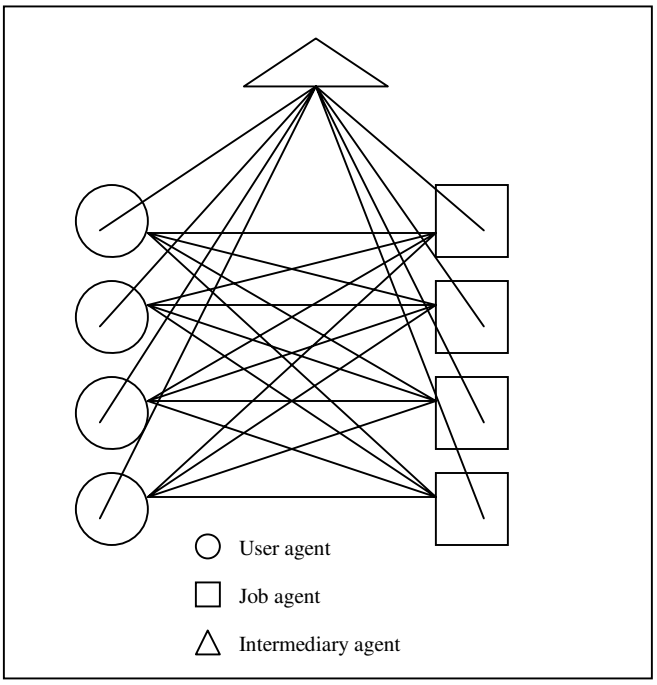

Figure 6.6. Case 3: Co-operating job agents with intermediary agent.

The testing was performed on the laptop: NC12950. During testing, we assume all user agents send the request messages to the job agents at the same time. In this case, the user agents also do not know which job agents are better for them, but the intermediate job agent know all job agents. Thus it can give the user agents a best suggestion. In addition, the communications between the intermediate agent and the job agents require some resources. Table 4.3 and Figure 4.8 depict the testing results.

Table 4.3. Average waiting time (ms) for different user and job agents in Case 3.

User Agents	200	400	600	800	1000
W_Time (ms) for 2 Job Agents	26639.23	50663.11	75997.21	94473.96	124803.4
W_Time (ms) for 4 Job Agents	26722.38	50918.03	76424.01	95129.47	125590.8
W_Time (ms) for 6 Job Agents	26803.92	51083.86	76673.45	95494.05	126016.4
W_Time (ms) for 8 Job Agents	26883.89	51246.49	76918.1	95851.62	126433.8
W_Time (ms) for 10 Job Agents	26962.34	51406.03	77158.09	96202.37	126843.2

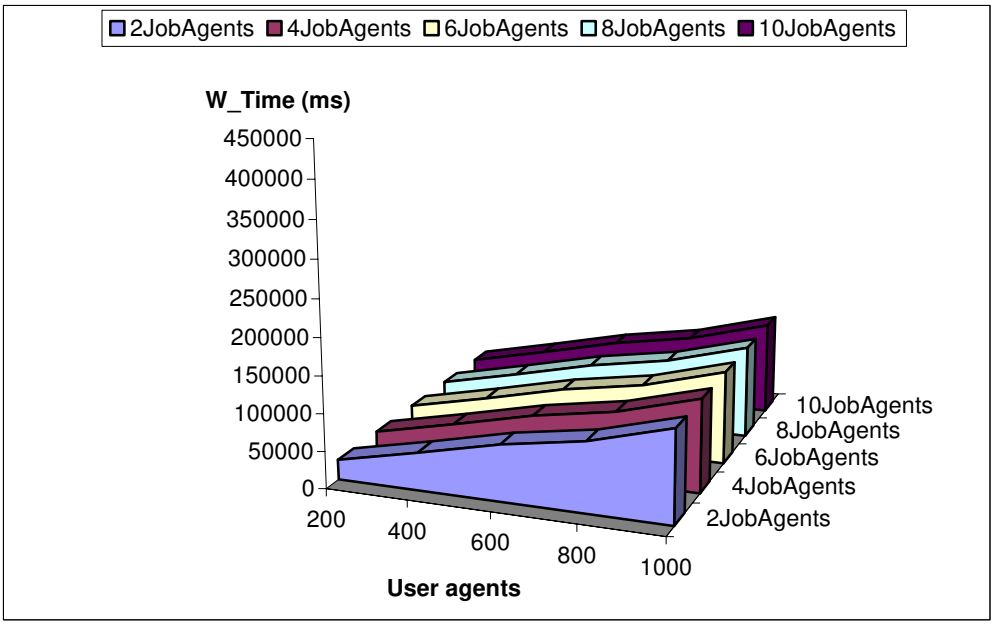

Figure 4.8. Average waiting time for different user and job agents in Case 3.

In this case, increasing of the number of the user agents still has great effect on the average waiting time, but the effect of the number of the job agents would be greatly reduced, and the increase of the number of the job agents would only have a little effect on the average waiting time. This case has better scalability than Case 1 and 2 when the number of the job agents is more than a constant number (general, it is the number of the job agents recommended by the intermediate agent). But this kind of organization requires that the intermediate agent have good inference ability.

4.2.4 Case 4: Adaptable Organization

There are other agent organizations. An important one of them is an adaptable organization. It means the platform allows the agents to dynamically change between organizational forms. For example, it may appropriate to select form 1 or 2 initially, and then change to form 3 with the increase of the number of the job agents.

In order to complete these functions, the agents must be augmented with several extra abilities as follows.

(1) The ability to remove or add knowledge of the existence of particular acquaintances to global or specific task data structures;
(2) The ability to create intermediaries or destroy them;
(3) The ability to transfer model information and delegate tasks to other agents.

In addition, since changing organization form has repercussions for the collective, the agents need a distributed method for trigging re-organization. In this situation, the actions that the agents can take are as follows.

(1) De-centralize a task: Create a commitment to share common information. This option moves the agents into form 2;

(2) Centralize a task: This option moves the agents into form 3;
(3) End a commitment to share common information: This option moves the agents into form 1.

There are many issues in this area. For example, how does the platform decide which parameters of the agent operation are appropriate for trigging re-organization? How does the re-organization cope with fluctuations in user's demands remains unclear? How about the effects of the trigger levels for re-organization? Answering the questions raised by these issues constitutes our further research.

5. Conclusion

There are many issues related to large-scale distributed system. In this paper, our research relates to the upward scalability of the labor market multi-agent systems in security services and organization forms. We have tested the system scalability in security services. Based on the simulation results, we get the following conclusions.

- The public key system would have severe impact on the system scalability if it were used in the multi-agent systems. But the symmetric key cryptography and integrity function only have little effect on the system scalability.

We have also proposed some improving cases for the labor market application, and compared their system scalabilities. Based on the tests, we have the following results.

- The platform based on the independent agent form scales up poorly, but its implementation is simple. It is more suitable for the small-scale systems.
- The platform based on the intermediate agent form has scales better but it requires the platform to have a better inference ability. It is more suitable for large-scale systems.
- The platform based on the adaptable organization form, is more flexible and suitable for variable agent size.

The dynamics of multi-agent system are hard to predict. The number of agents in large-scale distributed applications can vary considerably over time. The systems need to be able to scale almost immediately without a noticeable loss of performance, or a considerable increase in administrative complexity. Scalability is an important, yet under-researched, aspect of agent platforms. While scalability can refer to many different aspects like agent complexity or message volume, this document focuses only on the number of agents and messages, the resource consumption and performance effect of agents for the multi-agent system. Other area requiring more research is the multi-agent system scalability based on the adaptable organization form.

References

[1] A.Back, I.Goldberg and A.Shostack. Freedom 2.1 Security Issues and Analysis. May 2001. Available at http://www.freedom.net/info/whitepapers/Freedom_Security2-1.pdf.

[2] A.Lysyanskaya, R.Rivest and A.Sahai. Pseudonym Systems. Selected Areas in Cryptography: 6th Annual International Workshop, SAC'99, Volume 1758 of Lecture Notes in Computer Science, pages 184-200, Springer-Verlag, 1999.

[3] A.Poggi and G.Rimassa. An agent model platform for realizing efficient and reusable agent software. *Proceedings of the fourth international conference on Autonomous agents (AGENTS'00)*, pp.64-69, Barcelona Spain, ACM Press, June 2000.

[4] D.Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. Journal of Cryptology, vol.1, no.1, pages 65-75, 1988.
[5] D.Chaum and J.Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In Advances in Cryptology—CRYPTO '86, pages 118-167, Springer-Verlag, 1986.
[6] D.Chaum. Untraceable Electronic Mail, Return Address, and Digital Pseudonyms. Communications of the ACM, vol.24 no.2, pages 84-88, 1981.
[7] D.Goldschlag, M.Reed and P.Syverson. Hiding Routing Information. In R.Anderson, editor, Information Hiding: First International Workshop, Volume 1174 of Lecture Notes in Computer Science, pages 137-150, Springer-Verlag, 1996.
[8] D.R.Simon. Anonymous Communication and Anonymous Cash. In Advances in Cryptology – CRYPTO '96, Volume 1109 of Lecture Notes in Computer Science, pages 61-73, Springer-Verlag, 1996.
[9] F.Bellifemine, A.Poggi and G.Rimassa. JADE. *Proceedings of the fifth international conference on Autonomous agents (AGENTS'01)*, pp.216-217, Montreal Quebec Canada, ACM Press, May 2001.
[10] F.Brazier, M.V.Steen and N.Wijngaards. On MAS Scalability. Proceedings of Workshop "Infrastructure for Scalable Multi-Agent Systems", 6 pages, Montreal Canada, May 2001.
[11] Foundation for Intelligent Physical Agents. Specifications. 1997. Available at http://www.fipa.org.
[12] T.Ozsu and P.Valduriez. Principles of Distributed Database Systems. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 1999.
[13] G. Di Marzo, M.Muhugusa, C.Tschudin and J.Harms. Survey of Theories for Mobile Agents. Technical Report No.106, TeleInformatics, University of Geneva, 1996.
[14] G.Yamamoto and H.Tai. Architecture of an Agent Sever Capable of Hosting Tens of Thousands of Agents. Proceedings of fourth annual conference on Autonomous Agents, pages 70-71, 2000.
[15] G.Yamamoto and Y.Nakamura. Architecture and Performance Evaluation of a Massive Multi-Agent System. Proceedings of third annual conference on Autonomous Agents, pages 319-325, May 1999.
[16] I.A.Smith, P.R.Coen, J.M.Bradshaw, M.Greaves and H.Holmback. Designing conversation policies using joint intention theory. Proceedings of International Joint Conference on Multi-Agent Systems, 1998.
[17] I.Goldberg, D.Wagner and E.Brewer. Privacy-Enhancing Technologies for the Internet. In *Proceedings of IEEE COMPCON '97*, pages 103-109, 1997.
[18] JADE Project Home Page. Available at http://jade.cselt.it/.
[19] J.Borking. Proposal for Building a Privacy Guardian for the Electronic Age. In H.Federrath, editor, Anonymity 2000, Volume 2009 of Lecture Notes in Computer Science, pages 130-140, Springer-Verlag, 2000.
[20] J.L.Gustafson. Re-evaluating Amdahl's Law. Communications of the ACM, 31:532-533, 1988.
[21] J.B.Odubiyi, D.J.Kocur, S.M.Weinstein, N.Wakim, S.Srivastava, C.Gokey and J.Graham. SAIRE - a scalable agent-based information retrieval engine. Proceedings of the first international conference on Autonomous agents, pages 292-299, Marina del Rey, CA USA, Feb 1997.
[22] L.Chen. Access with pseudonyms. In Ed Dawson and Jovan Golic, editors, Cryptography: Policy and Algorithms, Volume 1029 of Lecture Notes in Computer Science, pages 232-243, Springer-Verlag, 1995.
[23] M.Greaves, H.Holback and J.Bradshaw. What is a Conversation Policy? Proceedings of workshop on Specifying and Implementing Conversation Policies, at third annual conference on Autonomous Agents, May 1999.
[24] M.Matskin. Agora: An Infrastructure for Cooperative Work Support in Multi-Agent Systems. In proceedings of Workshop "Infrastructure for Scalable Multi-Agent Systems" at Agents 2000, Barcelona, 2000.
[25] M.Woodside. Scalability Metrics and Analysis of Mobile Agent Systems. In Proceedings of Workshop "Infrastructure for Scalable Multi-Agent Systems" at Agents 2000, Barcelona, 2000.
[26] O.Rana and K.Stout. What is Scalability in Multi-Agent Systems. Proceedings Autonomous Agents 2000, Barcelona, pages 56-63, 2000.
[27] P.Boucher, A.Shostack and I.Goldberg. Freedom Systems 2.0 Archtecture. December 2000. Available at http://www.freedom.net/info/whitepapers/Freedom_System_2_Architecture.pdf.
[28] P.J.Turner and N.R.Jennings. Improving the Scalability of Multi-agent Systems. Proceedings of the 1st International Workshop on Infrastructure for Scalable Multi-Agent Systems, 2000.
[29] P.Maes. Hive:Distributed Agents for Networking Things. In IEEE Concurrency, pages 24-33, April-June 2000.
[30] R.Deters. Scalability & Multi-Agent Systems. Proceedings of Workshop "Infrastructure for Scalable Multi-Agent Systems", 7 pages, Montreal Canada, May 2001.
[31] R.Hes and J.Borking. Privacy-Enhancing Technologies: The Path to Anonymity. Revised Edition. A&V-11. Den Haag: Registratiekamer, 1998.
[32] R.Samuels and E.Hawco. Untraceable Nym Creation on the Freedom 2.0 Network. Zero-Knowledge Systems, Inc. white paper, 2000. Available at http://www.freedom.net/info/whitepapers/Freedom-NymCreation.pdf.
[33] R.Sun and T.Peterson. Multi-agent Reinforcement Learning Among Heterogeneous Agents in the Internet. Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999.
[34] S.Ossowski. Co-ordination in Artificial Agent Societies. Apringer-Verlag, 1999.
[35] V.Kumar and A.Gupta. Analysis of Scalability of Parallel Algorithms and Architectures: A Survey. SuperComputing91, pages 396-405, 1991.
[36] W.Dai. Pipenet 1.1. 2000. Available at http://www.eskimo.com/~weidai/pipenet.txt.
[37] Private Credentials. Zero-Knowledge Systems, Inc. white paper, 2000. Available at http://www.freedom.net/info/whitepapers/credsnew.pdf.

[38] R.B.Doorenbos, O.Etzioni, and D.S.Weld. A Scalable Comparison-Shopping Agent for the World-Wide Web. In W.L.Johnson and B.Hayes-Roth, (eds.), Proc. Proceedings of the First International Conference on Autonomous Agents (Agents'97), pp. 39-48, Marina del Rey, CA, USA, 1997. ACM Press.

Biography

Ronggong Song received his B.Sc degree in mathematics in 1992, M.Eng degree in computer science in 1996, Ph.D. in network security from Beijing University of Posts and Telecommunications in 1999. He had employed as Network Planning Engineer at Telecommunication Planning Research Institute of MII, and Postdoctoral Fellow at University of Ottawa. Now, he is working at NRC of Canada. His research interests are privacy protection, network security, e-commerce, IP mobility.

Larry Korba is the group leader of the Network Computing Group of the National Research Council of Canada in the Institute for Information Technology. He is the leader of the Canadian contribution to the Privacy Incorporated Software Agent (MULTI-AGENT) project. His research interests include privacy protection, network security, and computer supported collaborative work.