Modeling and Simulating the Scalability of a Multi-Agent Application System.

Song, Ronggong; Korba, Larry

*Modeling and Simulating the Scalability of A Multi-agent Application System **

Song, R. and Korba, L.
August 2002

* published in NRC/ERB-1097 August 2002, 27 Pages. NRC 44951.

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de Technologie de l'information

ERB-1097

NRC·CNRC

Modeling and Simulating the Scalability of A Multi-agent Application System

Larry Korba and Ronggong Song

August 2002

Canada

NRC 44951

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de Technologie de l'information

Modeling and Simulating the Scalability of A Multi-agent Application System

Larry Korba and Ronggong Song
August 2002

Modeling and Simulating the Scalability of A Multi-agent Application System[*]

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

Abstract

Scalability is a vital issue in the development of practical multi-agent systems. In this paper, we first provide an overview of approaches that may be applied to the multi-agent system scalability. We then model and simulate the scalability of a multi-agent application: a labor market case based on the JADE multi-agent platform. We analyze the effect of individual performance metrics on labor market system scalability.

1. Introduction

The growth in networked information resources requires the distribution and interoperation of information systems. Such systems cannot be easily realized with traditional software technologies because of the limits of these technologies in coping with distribution and interoperability. In the past few years Multi-Agent Systems (MAS) have emerged, combining research from the field of Distributed Artificial Intelligence (DAI) with new approaches to software engineering. This new paradigm proposes solutions to highly distributed problems in dynamic, open computational domains. These multi-agent systems have been studied as systems in which large numbers of agents process complicated tasks by breaking them down into smaller parts and interacting with each other. Different application platforms have been developed to support agent system development. For example, JADE [3, 9, 18] is one such software framework. It provides an environment for developing agent applications in compliance with the FIPA specifications [11] for interoperable intelligent multi-agent systems.

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 we consider in this document. 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.

In this paper, we provide an overview of approaches that may be applied to assess multi-agent system scalability, and describe scaling techniques and methods for enhancing scalability for agent systems. We then model and simulate the scalability of a multi-agent application system -- a labor market case in the JADE

[*] NRC Paper Number 44951

multi-agent platform, and analyze the effect of the performance metrics on the labor market system scalability.

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

2. Overview of Multi-agent System Scalability

We first introduce the concept and questions of scalability described in [26], and then analyze the factors that impact multi-agent system scalability. We then 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 discussion 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 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, including metrics like the total number of message exchanges necessary to converge on a solution, for instance. 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 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 one Java thread in order to be able to start new conversations proactively. Thus, only if the number of concurrent threads can be distributed over different physical machines, is it 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 for 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 computational complexity. Computational complexity mainly depends on the algorithms used in the agent system. In addition, it is also important to note, that the scalability problem of multi-agent systems is not only a question of computational resources but 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 propose 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 dependent upon 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 Multi-agent Scalability

2.4.1 A Multi-agent Application Overview

To give an overview of a multi-agent system, we introduce a Labor Market Case (LMC) based on the JADE multi-agent platform. The LMC is a labor market service system providing job search and match function. This system hosts job agents on a server. Users access this server via their Web browsers and find jobs. They can obtain information from several job agents in a single search action.

In LMC, users are represented as user agents. A user instantiates his own agent on the multi-agent server and inputs search requirements, such as job title, profession, location, salary range, etc. His agent then questions all job agents that can provide jobs. Each job agent queries its job database to obtain the best-matched jobs and the recommended jobs, reporting the results to the user agent.

Job agents live while the server runs. User agents live for six months to one year and are removed by the system when they expire. A user can access his user agent many times while it is alive. Even if he/she switches off his/her computer, his/her agent will still be alive, performing its functions and awaiting for the next time the owner wishes to connect.

The LMC provides users with an asynchronous job search function. That is, the user can access again to browse the search results after he/she inputs a set of search requirements.

In a LMC platform, 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.

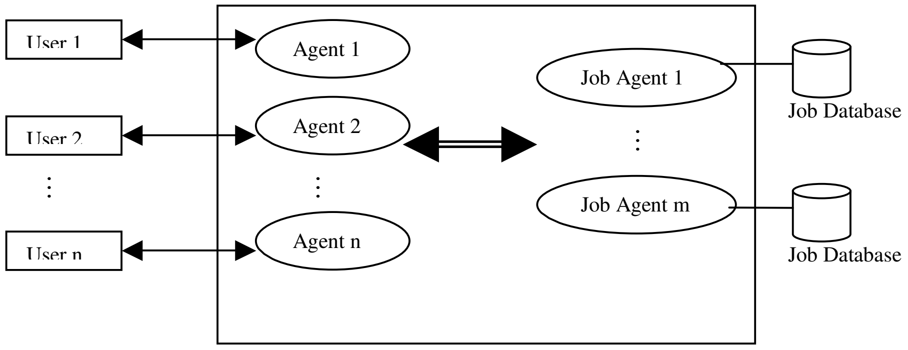

Figure 2.1. Architectural overview of a Labor Market Case.

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.

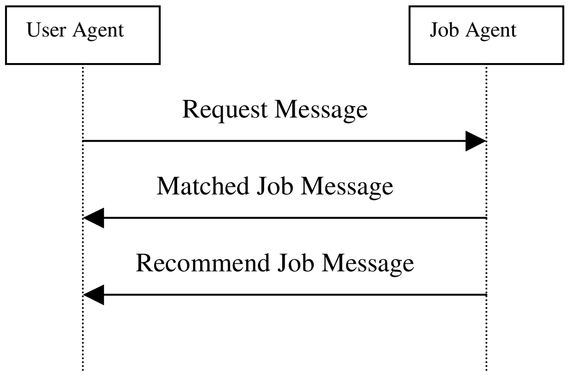

Figure 2.2. Interaction diagram between user agent and job agent.

In LMC system, since each user has its own agent on a server and each agent lives for half a year to 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.

An 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 construct models of multi-agent communities hierarchically.

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 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 and match 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.

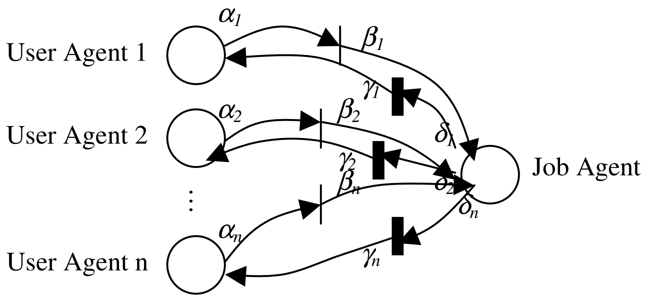

Figure 2.3. Petri Net: n user agents and a job agent.

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.

2.5 Scalability Enhancing Techniques

Currently the main approaches for significantly enhancing scalability are replicating the framework, component distribution, component replication and agent scheduling. Since the replication of the framework [29, 24] tends to waste resources and hence does not appear to be a promising solution for scaling up, it will be ignored in this report.

2.5.1 Component Distribution

An easy way to ensure that the combined resources of several physical machines can be used is to manually distribute the main components of the multi-agent system over several machines. When hosting the components in different processes, an inter-process communication (IPC) is required. This can be done by using only simple socket communication, via language specific approaches like Java's RMI or by using language and platform independent approaches like CORBA. JADE is a distributed multi-agent system. It can host agents on different agent containers (machines) and uses RMI for inter-container communication and IIOP for inter-platform communication.

While providing an easy way to distribute, this approach has two major drawbacks, the first of which is the manual distribution. It is up to a human programmer to decide where components have to reside. This significantly complicates adjustments to the often-varying load situation of machines. The second and more serious disadvantage concerns the linkage of the components to the resources of only one physical machine. This means, that it is impossible to scale individual components of the multi-agent system beyond the limits of a single machine. While this might not be a concern for a registry component, message routers and agent hosts can easily outgrow the capabilities of single machines.

As a result, the system is limited in its scalability since individual components cannot be distributed over more than one physical machine.

2.5.2 Component Replication

Instead of only distributing the components over different physical machines, it is also possible to replicate them. If the message router is expected to become a bottleneck, it is possible to have two or more message routers, each hosted on a different machine. Assuming that some load balancing is in place, this solution helps to significantly boost the performance of the multi-agent system. Typically, the agent hosts and message services (e.g. router) are replicated since they are the most likely to be performance bottlenecks. Like component distribution, the problem of inter-process communication must be solved by either implementing proprietary protocols via sockets or by using standardized ones like CORBA. It is important to note that with duplication of components, the amount of necessary inter-process communication is likely to increase. It is, therefore, important to design the system in a way that the load of IPC is distributed e.g. by using IIOP.

Component inflation is a very common strategy to ensure scalability within a multi-agent system. The replication of hosts and message routers is a common technique and by using mobile agents that can migrate between the agent hosts, a decentralized load balancing can be achieved. But it is important to realize, that the replication of components has two major disadvantages, resource consumption and increased complexity. By adding complete components like agent hosts, resources are wasted since all the host specific services are also replicated. In addition, the system becomes more complex, since more components have to be managed. Load balancing is now an additional problem!

While having multiple message routers might not be problematic, an inflation of hosts can quickly become a maintenance challenge, especially if large numbers of agents are floating between them. It is therefore no surprise, that there are literally no reports about successful deployed systems using this technique.

2.5.3 Agent Scheduling

To increase the capacity of individual agent hosts, special agent/thread schedulers are often used. Agent scheduling tries to optimize the distribution of the sparse resources among a large numbers of agents. Agents that are not performing tasks are deactivated thus preserving resources for the others. Agent scheduling typically requires that agents be split into a large group of deactivated agents and a small group of active agents.

Besides memory, the deactivated agents consume no other resources, while the active agents have access to all resources. Agent scheduling requires a “good” scheduling policy to determine which agents are to be moved from deactivated to active state and the ability to control the individual agent’s access to system resources. Various scheduling policies are possible that use ranking of importance, heuristics and statistics.

The most common form of scheduling is the event or message-oriented scheduling where an event or the arrival of messages priories agents. In such systems that typically consist of reactive agents, only agents that received messages are moved from the deactivated queue into the group of active agents (for the duration of the message processing). A variation of this approach can be seen in [14] where (reactive) agents are moved depending upon events (user logs) from the deactivated to the active group or vice versa. Agent scheduling is the only technique that has a proven track record of enabling the execution of large numbers of (reactive) agents. When using scheduling for proactive agents the CPU time slice for each agent is reciprocal to the number of agents, which renders this approach useless for large numbers of proactive agents. The fact that scheduling itself is also a computationally expensive operation might even worsen an already existing CPU shortage. Consequently this technique is not very useful for systems with predominately proactive agents. Since we have chosen JADE as our LMC system platform and JADE uses proactive techniques, and since JADE itself chooses very good scheduling techniques, this approach is not very useful for our LMC system.

3. Modeling the Labor Market System

In coming years it is expected that the matching of supply and demand in the labor market will increasingly be performed through Internet-based intermediaries such as software agents. Thus, electronic labor market is a natural domain for testing multi-agent system scalability. It involves a 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 privacy policy and searching requirements to the user agent via web browsers. The user agent then sends the requirements and some personal data 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 an anonymous way. Thus, the user agent may require some PET technologies to protect the user’s personal data or it may need to negotiate with some special job agents regarding the user’s privacy policy.

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 this particular scenario, user size and job size become more important for system scalability. Since real-time response to the request is not necessary, an appropriate response delay is acceptable.

Based on agent's function (e.g. one agent with one user or one agent with multi-users) in the Labor Market System (LMS), we propose three models as follows. They have different functions and scalabilities. We will simulate and analyze them in Section 4.

3.1 Model 1

Each container has only one user agent and one job match agent (see Figure 3.1). All users' CV, privacy policy, profile and search conditions are saved in the user database. All users share one user agent that sends user's request data and some personal data to the job match agent according to the user's privacy policy. All job data are saved in job database. The job agent matches the user's request with job data stored in the job database. All user agents and job agents are proactive agents. We call this the centralized model.

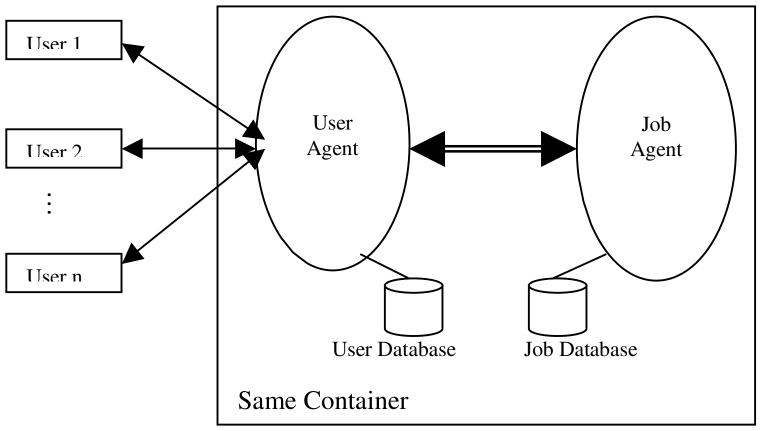

Figure 3.1. The Labor Market Application Model 1.

The advantages of this model are as follows: the scalability of the user and job size is very good with a single host, and it suits centralized control and management, by reducing the complexity and cost for privacy and security management. On the negative side, the user agent has more power, becoming the trusted third party. The quantitative analysis of the scalability is simulated in Section 4.

In this model, one possible configuration allows job agents in different containers but the same platform to share their job database. Figure 3.2 depicts this situation. If so, it is not necessary for user agents to communicate with other container's job agents. The advantage in this situation is a reduction of communication cost between different containers. The disadvantage is that the shared job database may be very large, which may increase search and match cost.

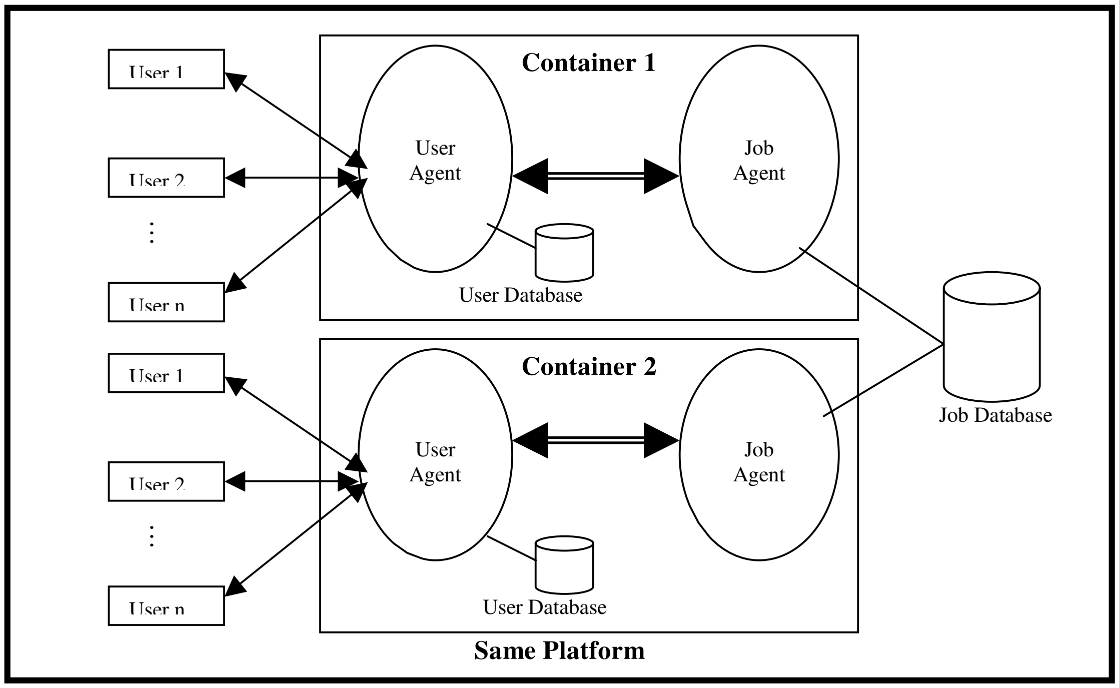

Figure 3.2. Model 1with the shared job database in the same platform.

The second situation is where the job agent in a different container has for itself, its own job database. Figure 3.3 depicts this situation. In this situation, it is necessary for the user agent to communicate with other container's job agent. This may reduce the local search and match cost, but if the user agent doesn't get the result from local job database it may need to go to other containers to search, which may increase communication cost between different containers.

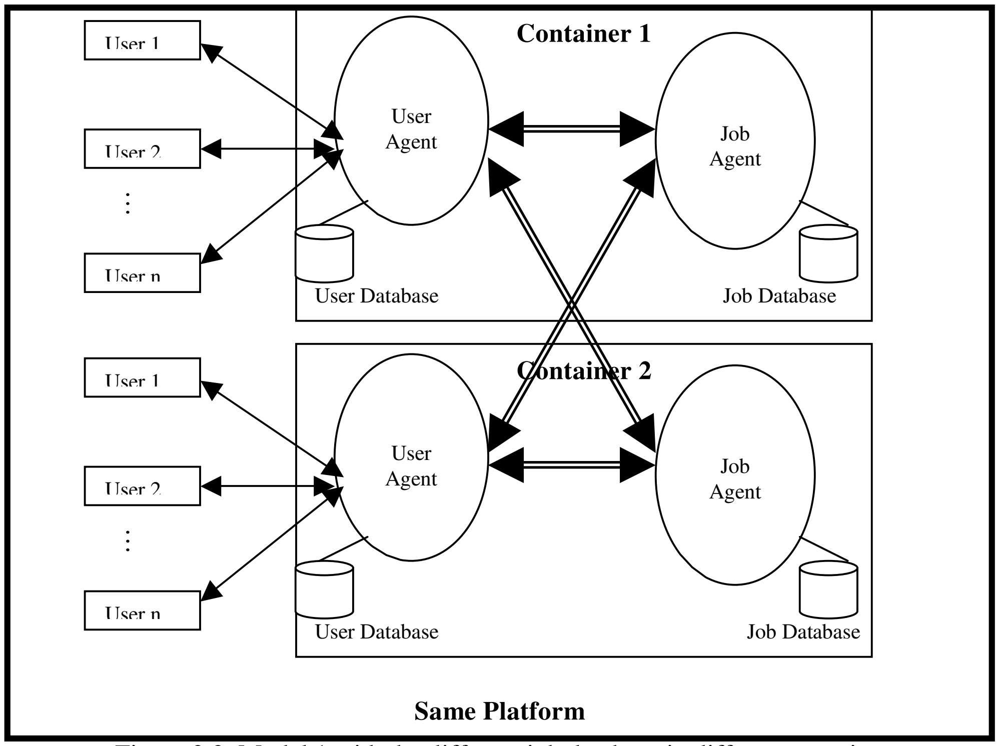

Figure 3.3. Model 1 with the different job database in different containers.

Finally, it is necessary for the user agent to communicate with other job agent on different platforms. Figure 3.4 depicts this situation. If the user agent doesn't get results from its own platform, it needs to go to another platform for job searching, which would increase the communication cost between different platforms.

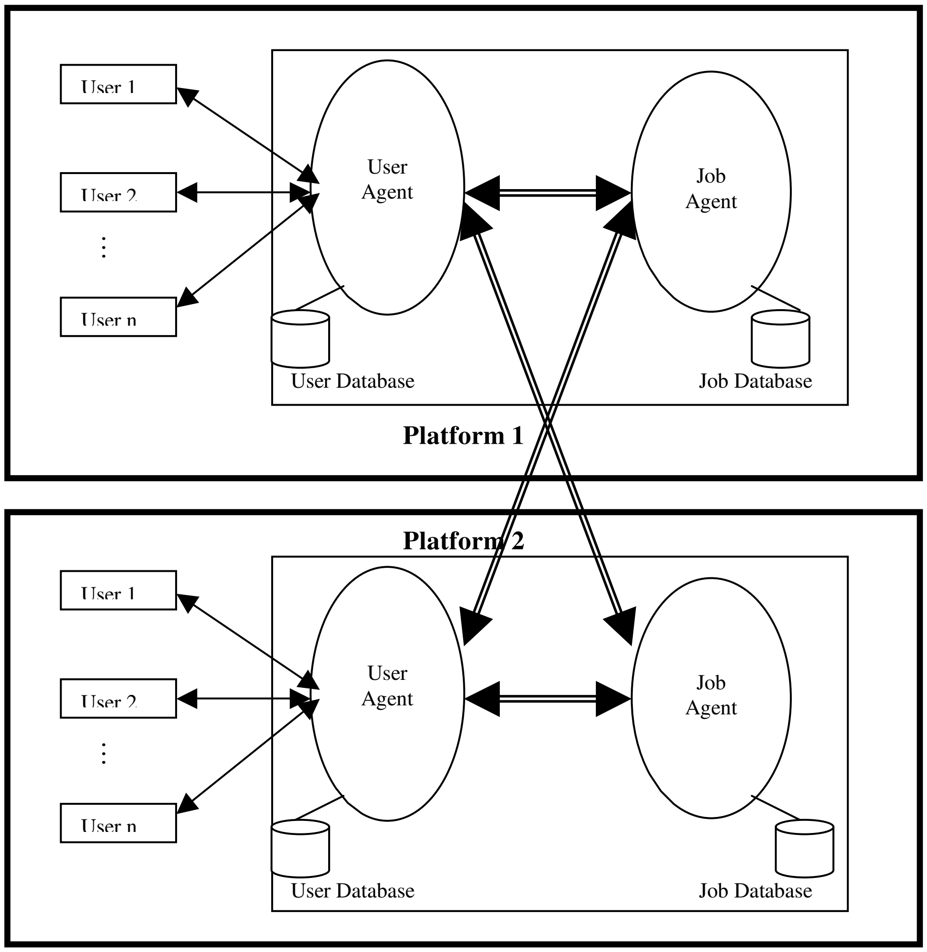

Figure 3.4. Model 1: Communication between different platforms.

3.2 Model 2

Each user has his/her own personal agent and each container only has one job match agent (see Figure 3.5). Each user's agent sends his/her request data and some personal data to the job match agent according to the user's privacy policy. All job data is saved in the job database. The Job agent matches the user's request with job data stored in the job database. We call this a semi-centralized model.

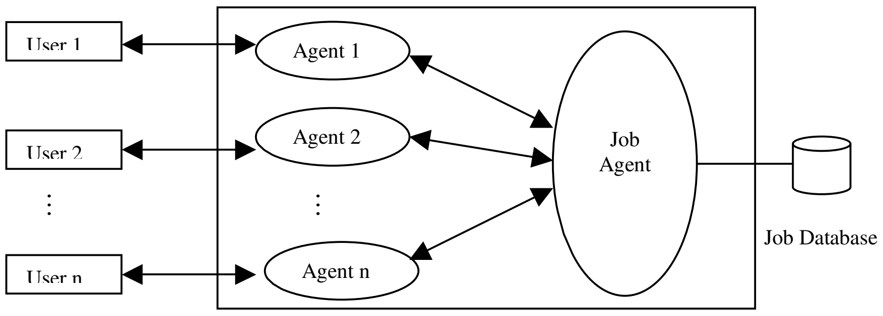

Figure 3.5. The Labor Market Application Model 2.

The advantages of this model are that the scalability of the job size is very good for a single host, and the user has more power to control his privacy management. But since each user has his/her own agent and the labor market system uses JADE as agent platform (i.e. using proactive agent technique), this may limit the user agent size within a single host. Thus we must distribute the user agents among different hosts (containers) for scalability. Another disadvantage is that the complexity cost of privacy and security management may increase because every user controls his/her own privacy and security policy. The quantitative analysis of this model scalability will be simulated in Section 4.

In this model, there are also three situations. The first situation is similar to the first situation of the model 1. Figure 3.6 depicts this situation. In this situation, every user agent brings the user's search conditions and personal data to the job agent.

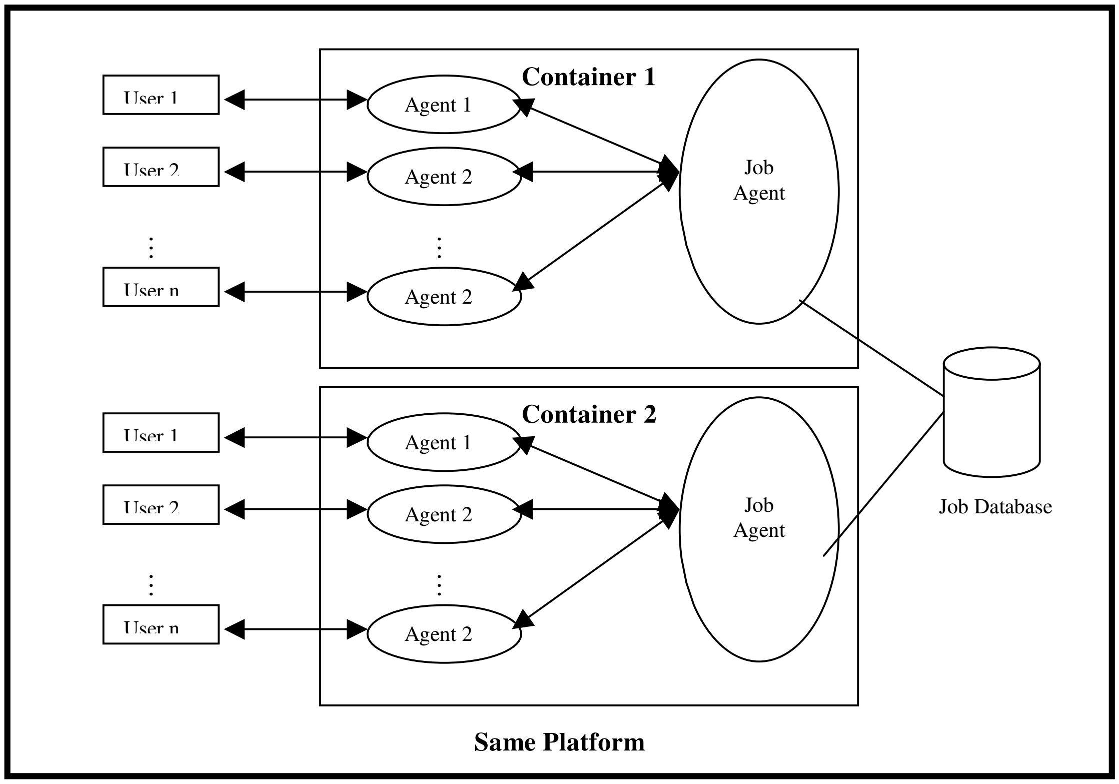

Figure 3.6. Model 2 with the shared job database in the same platform.

The second situation is similar to the second situation of model 1 and is depicted in Figure 3.7. Here, if he user's agent doesn't get the result from his/her local job agent, it needs to go to other container's job agent to search for a job.

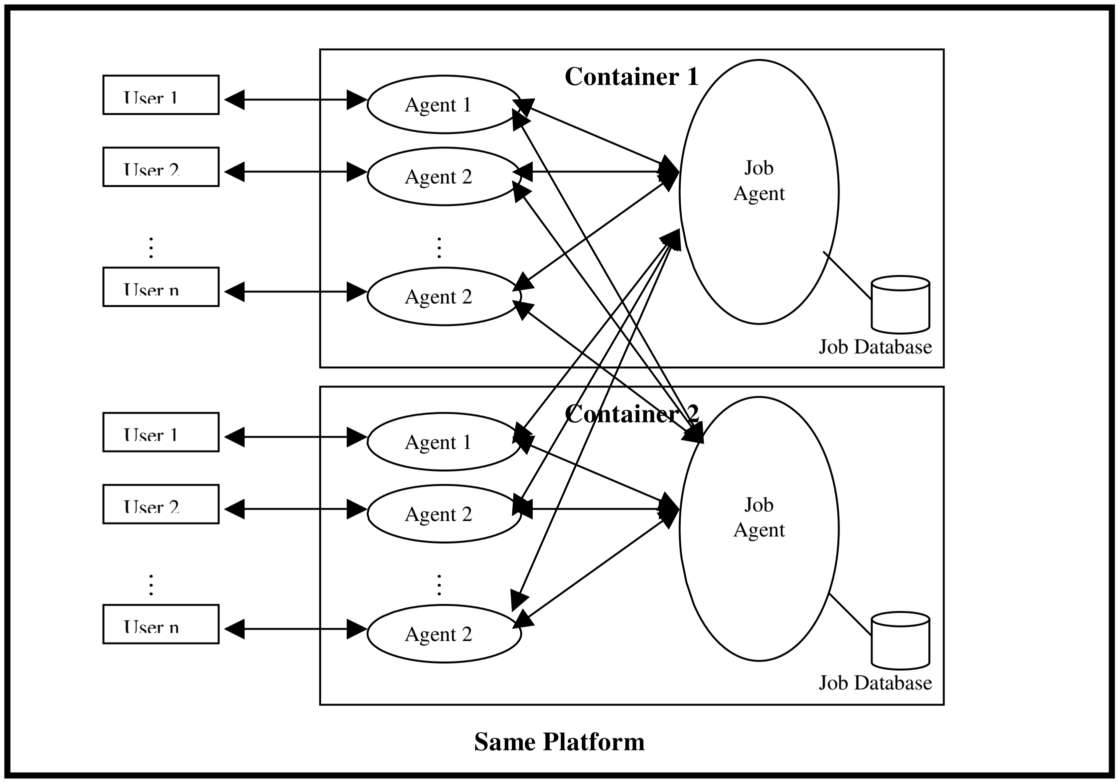

Figure 3.7. Model 2 with the different job database in different containers.

3.3 Model 3

In this model, each user has his/her own personal agent and each party, organization or company also has its own job match agent (see Figure 3.8). Each user's agent sends his/her request data and some personal data to the job match agent according to the user's privacy policy. Job agent matches the user's request with job data stored in the job database. We call this a decentralized model.

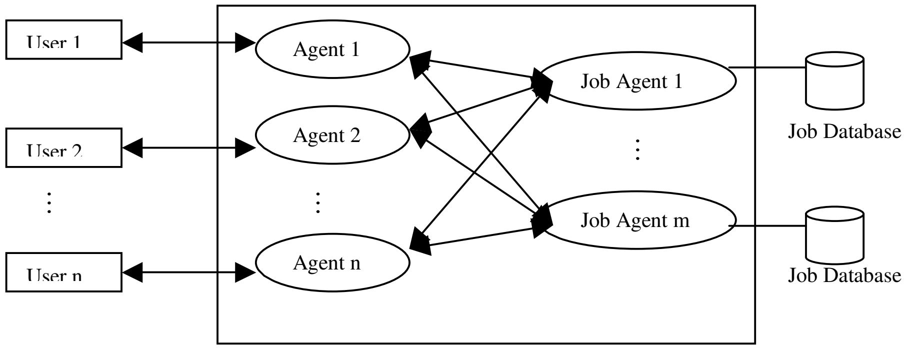

Figure 3.8. The Labor Market Application Model 3.

The advantages of this model are that it is more convenient for users searching jobs, and parties offering jobs and their negotiations. In addition, users have more control over privacy management. But since each user and party has its own agent and the labor market system uses JADE as agent platform (i.e. using proactive agents), this may limit the user and party agent size on a single host. Thus we must distribute the user and party agents in the different hosts (containers) for scalability. Another disadvantage is that the complexity cost of the privacy and security management will be more than that of model 2. The quantitative analysis of the scalability of this model is simulated in Section 5.2 and 5.3. In this model, there are also three situations that are similar to both model 1 and model 2, but more complex. Figure 3.9 depicts the communications between user agents and job agents.

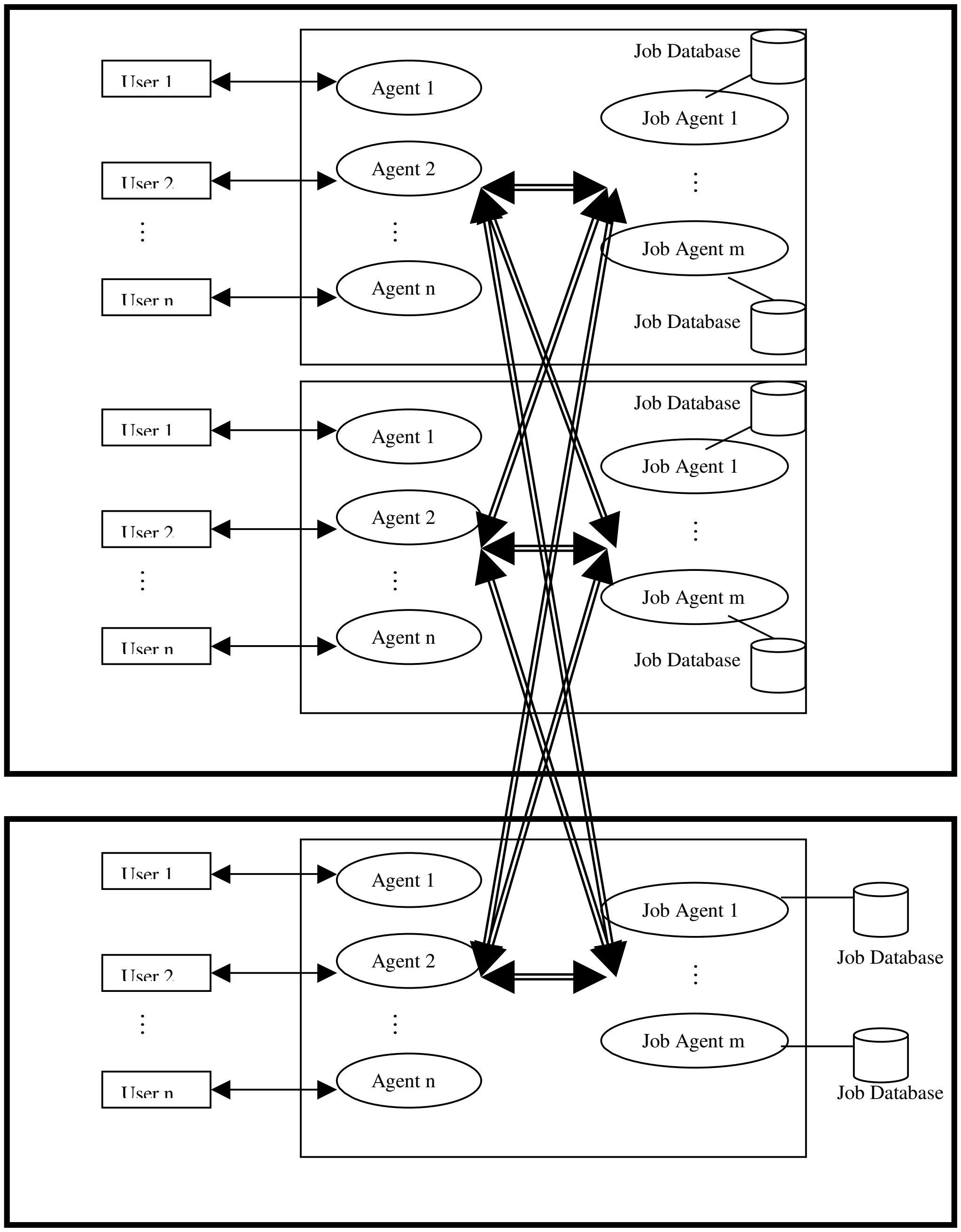

Figure 3.9. The Labor Market Application Model 3.

4. The Simulation of the Labor Market System Scalability

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.

4.1 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 733 MHz with 256 MB of system memory, and Windows 2000 operating system. The second computer is Intel Pentium 3 (named Spin41), its CPU is 733MHz and memory is 256 MB, and the operation system is Windows NT 4.0. Another computer, named NC84, is an Intel Pentium 4, operating at 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 do not deal with the processing time of the database. We use a time of zero for the database processing time. We only measure the processing time of the agents for request and reply.

4.2 Simulation Parameters

In the labor market 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. Since so far we have not received details of the privacy and security processing (e.g. protocols and algorithms, etc.) in the labor market system, we only give some simulation results on the computing complex cost for privacy and security processing for our Model 3 according to the measure results to some core public key algorithms. The parameters are defined as follows.

- User size: the number of the users (senders).
- U_Time: Average response time for a user sending a query and receiving a reply. It is an average time that the user agent sends a request message to a job agent and receives a reply from the job agent.

- J_Time: Average time for a job agent to process a query. It is an average time that the job agent receives a request message and sends a reply to the user agent.
- 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.

4.3 Simulation Results

In this Section, we describe our simulation results in two parts. The first part is a comparison between Model 1 and Model 2 in terms of processing time, CPU and Memory usage. The second part focuses on the simulation results for Model 2, since we believe that most people will want to have individual agents. As well, Model 2 is suitable for general situations.

4.3.1 Comparison between Model 1 and Model 2

The tests were performed on PC: NC12950. The users' agents and job agent were run in the same main container. In order to provide a comparison between Model 1 and Model 2, we use another simulation parameter—T_Time: total processing time for all request and reply messages in both users' agents and job agent side. The simulation tests are described as follows.

First, in order to test Model 1, we use one agent as the users' agent, and another agent as the job agent. The users' agent sends 1000, 2000, …, 128000 request-messages to the job agent, respectively. The simulation test results are as follows. Table 4.1 depicts the total processing time for the request and reply messages.

Table 4.1: The total processing time for request and reply messages in Model 1.

Messages	1000	2000	4000	8000	16000	32000	64000	128000
T_Time (ms)	501	751	1232	2203	4166	7711	14621	28912

Figure 4.1 depicts the CPU and Memory usage for the request and reply messages in Model 1. Considering the CPU usage history, the first two pulses are the simulation results for sending 1000 request and reply messages, the next two pulses are for sending 2000 request and reply messages, ..., the last two pulses are for sending 128000 request and reply messages. Each pulse depicts one testing and we do two times testing for one situation. In addition, each pulse has two peaks. 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. Since there is only one user agent, the user agent has the same effect on the Memory usage.

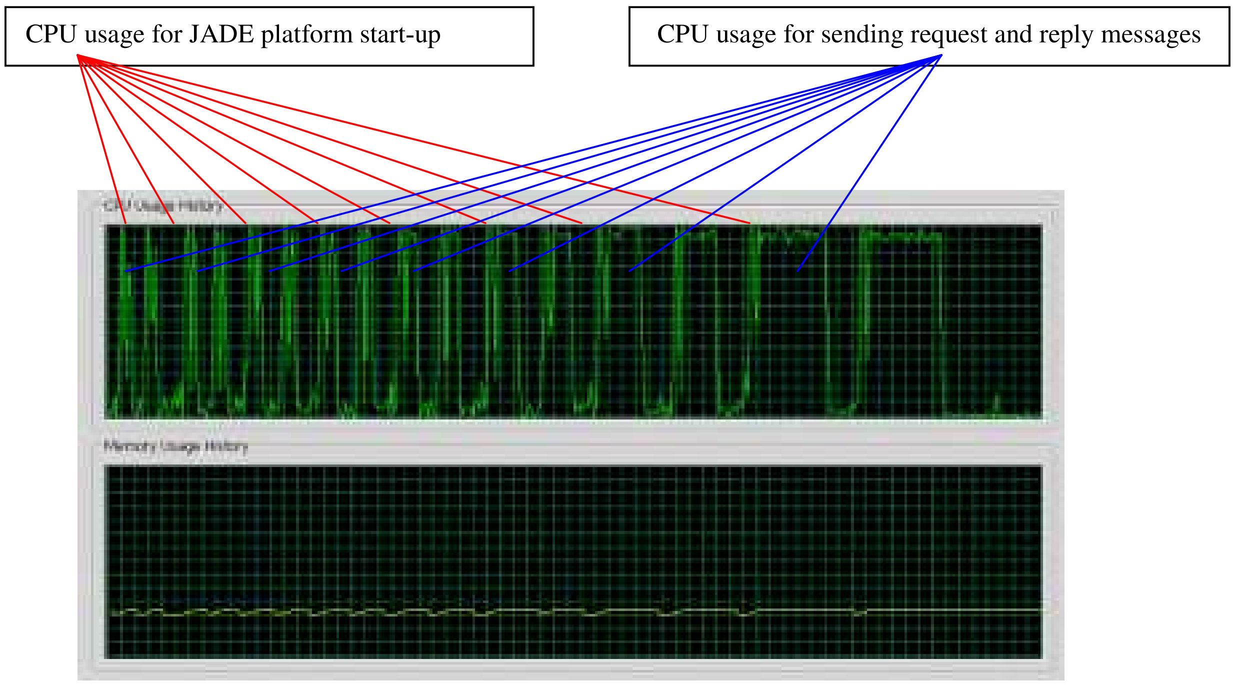
CPU usage for JADE platform start-up
CPU usage for sending request and reply messages
CPU Usage History
Memory Usage History

Figure 4.1. CPU and Memory usage for the request and reply messages in Model 1.

In order to compare with Model 1, we use one agent as the job agent and other agents as the users' agents for Model 2 testing. In this simulation, one user only has one user agent, and each user agent only sends 1000 request messages to the job agent. We simulate the results for one user agent, two user agents, …, 128 user agents, respectively. The simulation test results are listed in Table 4.2.

Table 4.2: The total processing time for request and reply messages in Model 2.

User agents	1	2	4	8	16	32	64	128
Messages	1000	2000	4000	8000	16000	32000	64000	128000
T_Time (ms)	571	962	1512	2624	5328	10305	23532	45615

Figure 4.2 depicts the CPU and Memory usage for the request and reply messages in Model 2. Considering first the CPU usage history, the first two pulses are the simulation results for one user agent sending 1000 request and reply messages, the next two pulses are for two user agents sending 2000 request and reply messages and each user agent just sending 1000 messages, ..., the last two pulses are for 128 user agents sending 128000 request and reply messages and each user agent sending 1000 messages. In addition, each pulse has two peaks in Figure 4.2, 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. Since the user agent size is small, the increase in the numbers of agents only has a small effect on the Memory usage.

Figure 4.3 depicts the comparison of the total processing time for sending request and reply messages between Model 1 and Model 2.

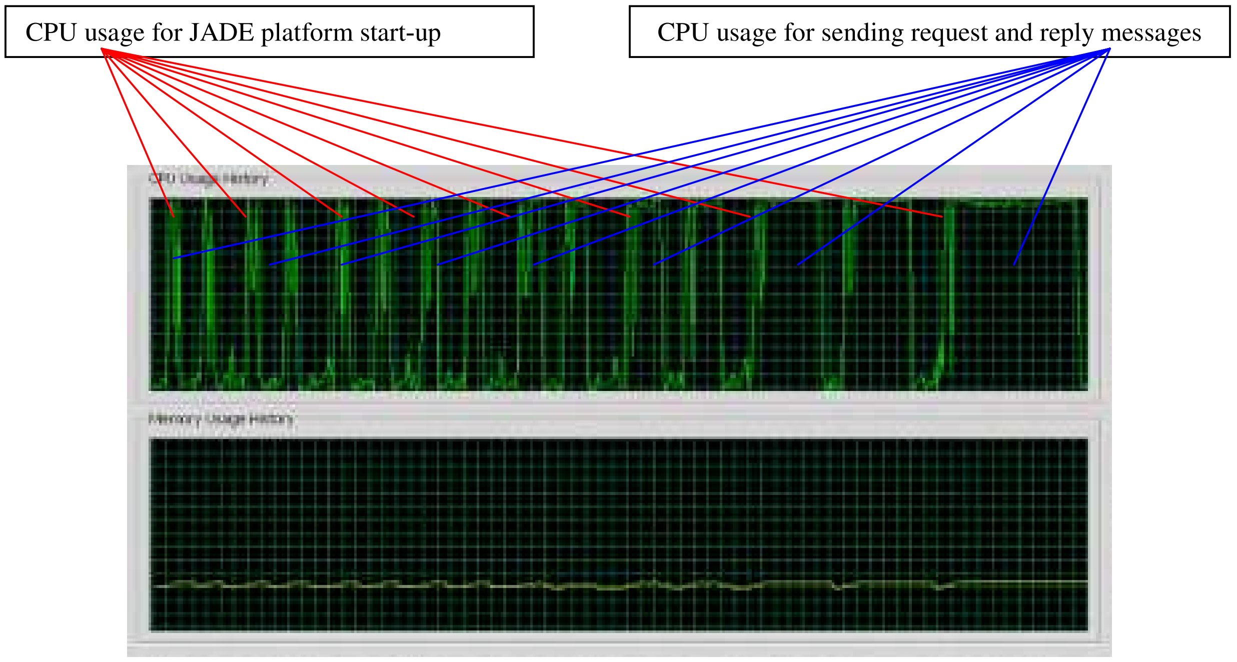
CPU usage for JADE platform start-up
CPU usage for sending request and reply messages
CPU Usage History
Memory Usage History

Figure 4.2. CPU and Memory usage for the request and reply messages in Model 2.

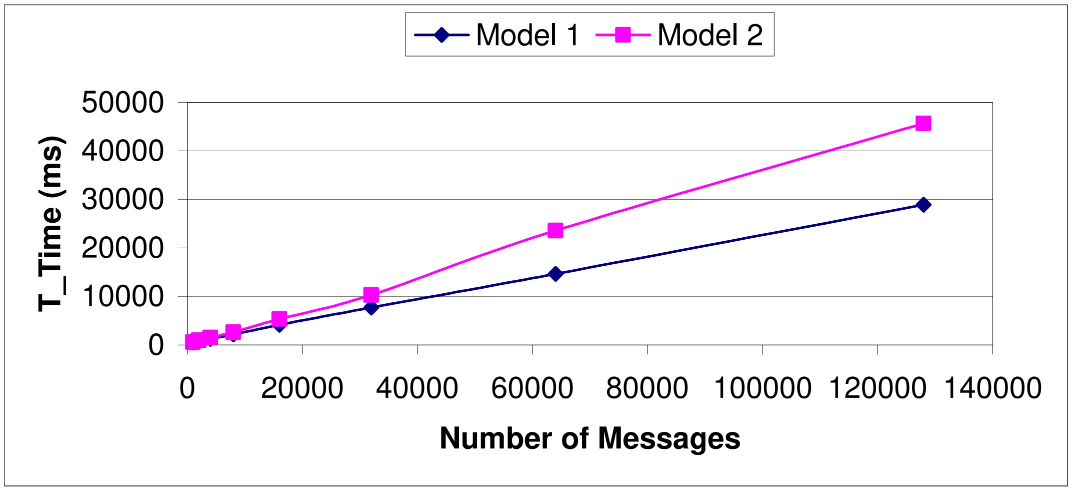

Figure 4.3. The comparison of the total processing time for sending request and reply messages between Model 1 and Model 2.

Although the performance of Model 1 is better than the performance of Model 2 from the above simulation, we think Model 2 is more suitable for general situations since we believe that most people would like to have their own individual agent. Please note that the above simulations only deal with the simple request-reply messages and don't consider the complex software processing in real world.

4.3.2 Simulation Testing on Model 2

The simulation tests were performed on the PCs: Spin41 and NC84. In this part, we test the U_Time and J_Time under different situations. First, we simulate the effect on the U_Time and J_Time when the agents are run in the different containers, different hosts and different platforms. Final, we simulate the effect of the containers' size on the U_Time and J_Time.

In addition, one user only has one user agent in Model 2, and each user agent sends 1000 request messages to the job agent during testing. There is only one job agent receiving all request messages and sending the reply messages during the following simulation.

(1) The effect of the different containers, different hosts and different platforms

The simulation testing is done according to the users' agents and job agent in the same main container, the users' agents and job agent in the different container but in the same host, the users' agents and job agent in the different container and different hosts, and the users' agents and job agent in the different platforms. In this part, we test the effect of the number of the user agents on the U_Time and J_Time under the above different situations.

(a) Job agent and user agents in the same main container

The testing was performed on PC: Spin41. The users' agents and job agent were run in the same main container, that is the only container during the simulation. Table 4.3 and Figure 4.4 depict the simulation results.

Table 4.3: U_Time and J_Time when the users' agents and job agent are run in the same container.

User agents' size	1	2	4	8
J_TIME (ms)	0.34034	0.290645	0.262816	0.241655
U_TIME (ms)	0.3	0.28	0.3605	0.795

User agents' size	16	32	64	128
J_TIME (ms)	0.2884	0.3826	0.2995	0.6118
U_TIME (ms)	2.261938	4.630063	10.657	33.3004

User agents' size	256	512	1024	1600
J_TIME (ms)	0.4487	0.3786	0.4617	0.543976
U_TIME (ms)	48.137	58.84357	96.20767	204.8378

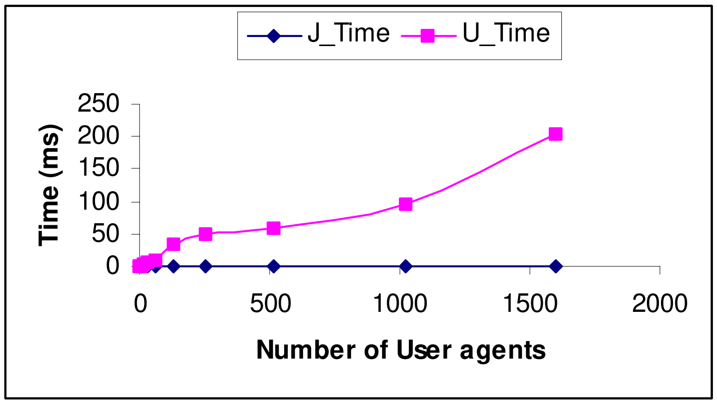

Figure 4.4. J Time and U Time for the user agents and job agent in the same container.

Note that when the number of user agents reaches 1124, some agents cannot write data into the files. But before the number of user agents reaches 1024, all agents work properly.

(b) Job agent in the main container, user agents in the other container, and the two containers in the same host and same platform

The testing was performed on PC: Spin41. The job agent was run in the main container, the users' agents were run in the other container, and the two containers were run in the same host and the same JADE platform. Table 4.4 and Figure 4.4 depict the simulation results.

Table 4.4: U_Time and J_Time when the users' agents and job agent are run in the different containers.

User agents' size	1	2	4	8
J_TIME (ms)	8.05005	7.71986	7.5276	7.39905
U_TIME (ms)	8.112	8.6625	19.1825	43.8355

User agents' size	16	32	64	128
J_TIME (ms)	7.6749	7.9825	8.2209	8.0755
U_TIME (ms)	97.04775	205.8363	417.6849	823.4324

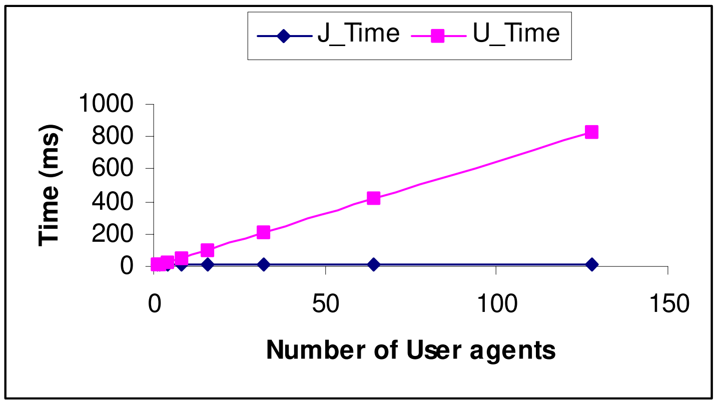

Figure 4.5. J_Time and U_Time for the user agents and job agent in the different containers.

(c) Job agent in the main container, user agents in the other container, and the two containers in the different hosts but in the same platform

The testing was performed on PC: Spin41 and NC84. The job agent was run in the main container of the Spin41, the users' agents were run in the non-main container of the NC84, and the two containers were run in the same JADE platform. Table 4.5 and Figure 4.6 depict the simulation results.

Table 4.5: U_Time and J_Time when the users' agents and job agent are run in the different hosts.

User agents' size	1	2	4	8
J_TIME (ms)	6.686687	4.393197	4.217304	4.077635
U_TIME (ms)	6.699	8.788	16.361	28.67

User agents' size	16	32	64	128
J_TIME (ms)	4.20352	4.31534	4.15462	4.656166
U_TIME (ms)	61.10969	134.0964	267.5084	529.2665

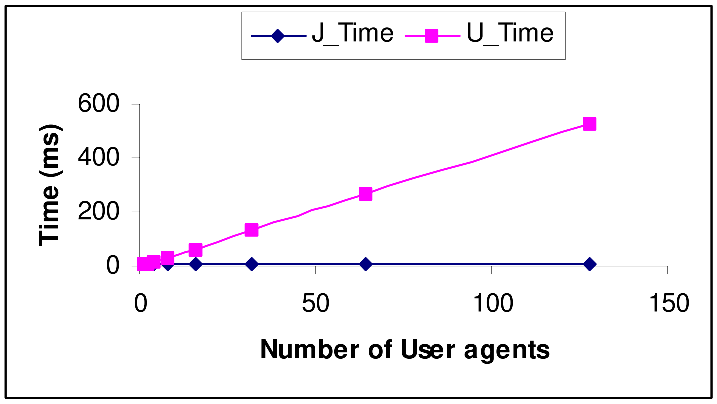

Figure 4.6. J Time and U Time for the user agents and job agent in the different hosts.

(d) Job agent and user agents in the different platform

The testing was performed on PC: Spin41 and NC84. The job agent was run in the main container of the Spin41, the users' agents were run in the main container in NC84. In addition, the two containers were run in the different JADE platforms. Table 4.6 and Figure 4.7 depict the simulation results.

Table 4.6: U_Time and J_Time when the users' agents and job agent are run in the different platforms.

User agents' size	1	2	4	8
J_TIME (ms)	15.62	9.93	8.17	7.85
U_TIME (ms)	15.88	20.10	32.99	63.08

User agents' size	16	32	64	128
J_TIME (ms)	7.74	7.68	7.62	7.44
U_TIME (ms)	120.7	233.92	458.55	887.14

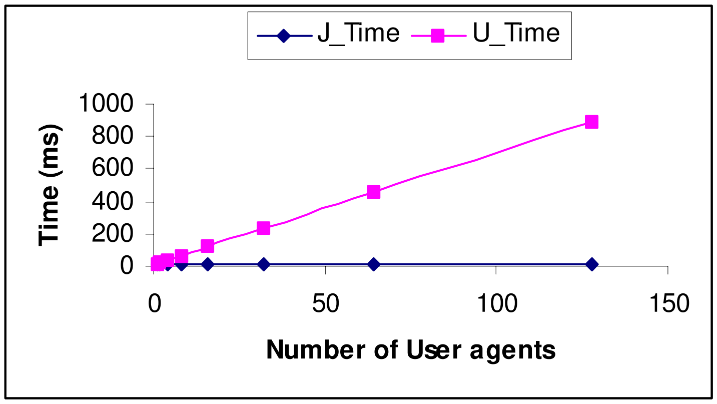

Figure 4.7. J Time and U Time for the user agents and job agent in the different platforms.

(e) Comparison of the above simulation results

In order to evaluate the scalability of the above situations, we give a comparison of the above simulation results as follows. Figure 4.8 depicts the comparison of the J_Times for the above simulations.

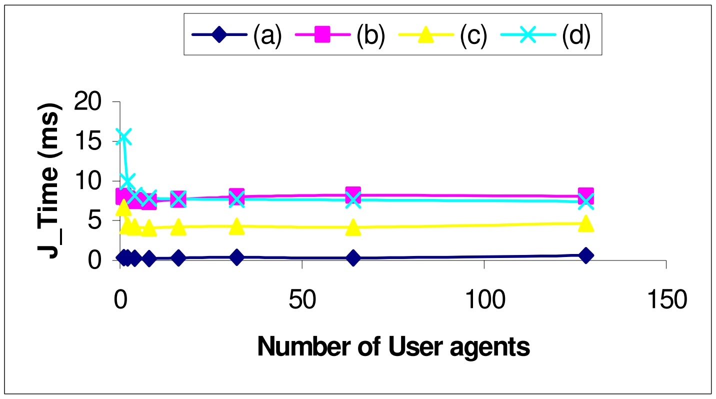

Figure 4.8. The comparison of the J_Time for the different situations in Model 2.

Figure 4.9 depicts the comparison of the U_Times for the above simulations.

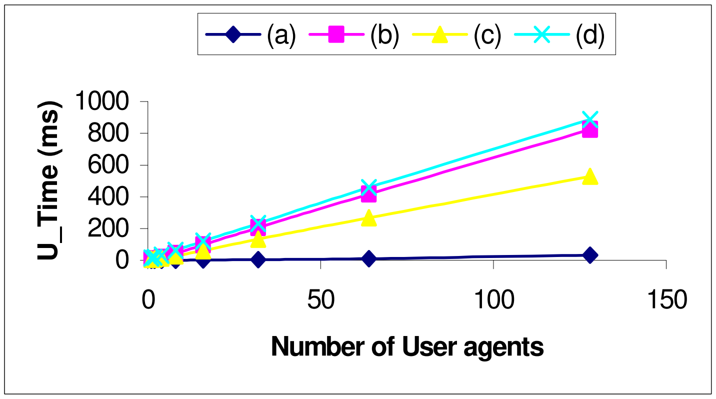

Figure 4.9. The comparison of the U Time for the different situations in Model 2.

We get the following conclusions from the above simulation and comparison results.

- The increase of the number of user agents has an effect on the U_Time, but no effect on J_Time for all situations. This is mainly because the waiting time in the job agent queue increases with the number of user agents. In addition, the U_Time increases linearly with the increase of the number of user agents.
- From the above comparison, we know that the situation (a) is better than the situation (c), the situation (c) is better than the situation (b) and (d), and the situation (b) is similar with the situation (d) for both U_Time and J_Time.

(2) The effect of the number of the containers

The testing was performed on PC: Spin41 for situations wherein the users' agents and job agent were in the same container, and the users' agents and job agent were in the different containers, but all agents in the same platform. In this part, we test the effect of the number of the containers on the U_Time and J_Time under the above different situations.

(a) Job agent and user agents in the same main container

For this testing, we ran five agents in the same main container. Four of the agents are user agents, and last agent is a job agent. We then increase the number of the containers for testing their effect on the U_Time and J_Time. Table 4.7 and Figure 4.10 depict the simulation results.

Table 4.7: U_Time and J_Time when the users' agents and job agent are run in the same main container.

Containers' size	2	4	8	16
J_TIME (ms)	0.338085	0.398	0.4255	0.43825
U_TIME (ms)	0.7985	0.71625	0.714	0.711

Containers' size	32	48	64	128
J_TIME (ms)	0.47325	0.52075	0.53075	0.6835
U_TIME (ms)	0.7765	0.7925	0.78875	0.84375

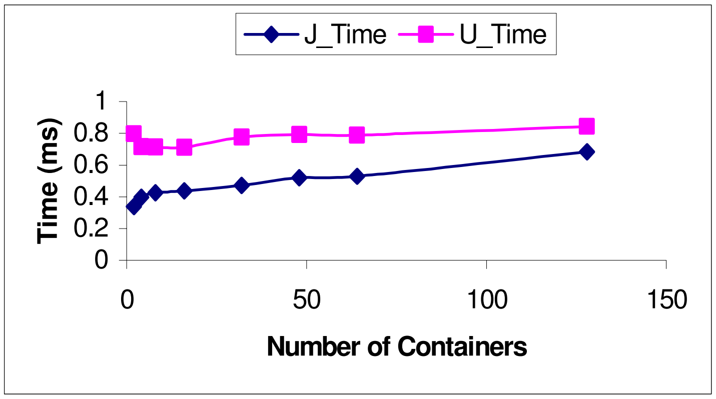

Figure 4.10. U Time and J Time for the user agents and job agent in the same container.

(b) Job agent and user agents in the different containers

For this testing, we ran four agents as user agents in a non-main container and another agent as job agent in the main container. We then increased the number of the containers for testing their effect on the U_Time and J_Time. Table 4.8 and Figure 4.11 depict the simulation results.

Table 4.8: U_Time and J_Time when the users' agents and job agent are run in the different containers.

Containers' size	2	4	8	16
J_TIME (ms)	7.556	8.78525	11.8245	19.82625
U_TIME (ms)	19.1825	20.62225	31.0495	73.6885

Containers' size	32	48	64	128
J_TIME (ms)	32.97725	46.294	54.601	101.163
U_TIME (ms)	125.703	177.1873	207.0505	386.571

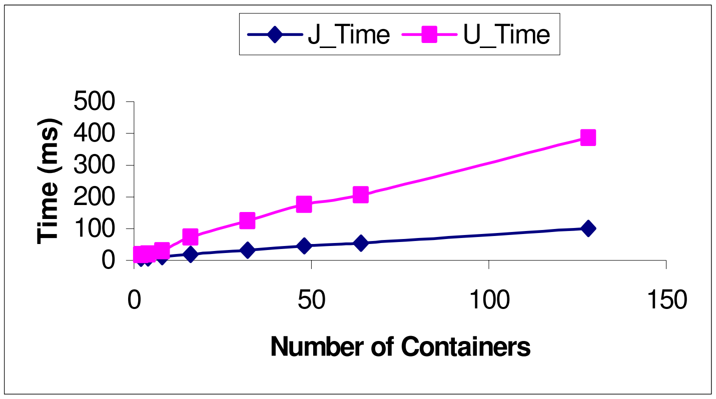

Figure 4.11. U Time and J Time for the user agents and job agent in the different containers.

(c) Comparison of the above simulation results

In order to evaluate the scalability of the above situations, we give a comparison of the above simulation results as follows. Figure 4.12 depicts the comparison of the J_Times for the above simulations.

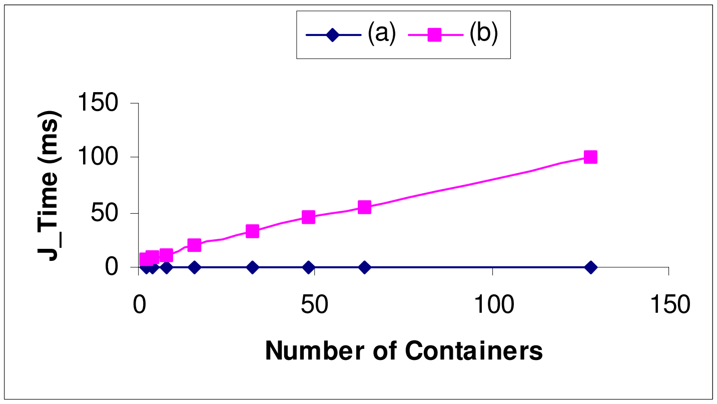

Figure 4.12. The comparison of the J_Time for the above situations.

Figure 4.13 depicts the comparison of the U_Times for the above simulations.

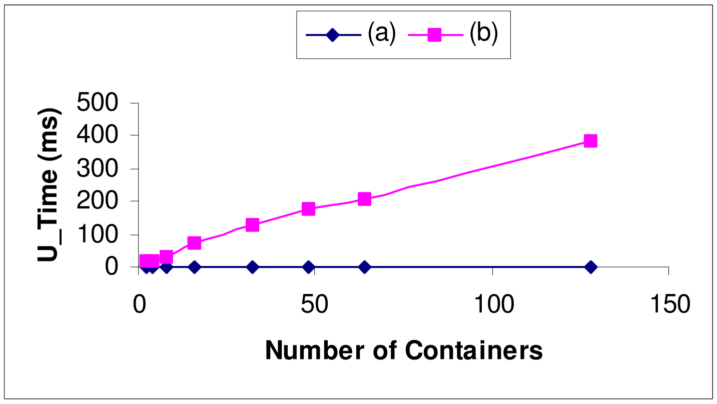

Figure 4.13. The comparison of the U_Time for the above situations.

From the above simulation and comparison results, we derive the following conclusions.

- Increasing the number of containers has little effect on U_Time and J_Time when the user agents and job agent are run in the same container.
- Increasing the number of containers will cause the linear increase of U_Time and J_Time when the user agents and job agent are run in the different containers.

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. We have proposed several probable models for the labor market application, and have tested system scalability under different situations. Based on the simulation results, we get the following conclusions.

- A distributed labor market multi-agent system provides better scalability than a single labor market system.
- Scalability would be better if one machine runs only in one container, and the user agents and job agent are run in the same container.
- The increase of the number of user agents has an effect on the average processing time for the request-reply messages in the user agent side, but no effect on the average time for processing the request-reply messages in the job agent side. The average processing time for the request-reply messages on the user agent side increases linearly with an increase of the number of user agents.
- The increase of the number of containers will cause a linear increase in the average processing time for the request-reply messages in the user agent side and the average time for processing the request-reply messages in the job agent side when the user agents and job agent are run in the different containers. But the increase of the number of containers has little effect on the system scalability when the user agents and job agent are run in the same container.

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 paper focuses only on the number of agents and messages, the resource consumption and performance effect of agents for the labor market multi-agent system. Other areas requiring more research include the following.

- The multi-agent system scalability based on the adaptable organization forms.
- Scalability of security services, including privacy enhancing technologies.

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 '9*7, 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

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.

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, P.R.China, and Postdoctoral Fellow at University of Ottawa, Canada. Now, he is working at NRC of Canada. His research interests are privacy protection, network security, e-commerce, IP mobility and QoS.