A Study on Practical Utilization of Mobile Agent Technology

January 2014

Masayuki Higashino

Abstract

Distributed systems are becoming increasingly complex and massive with the advance of computer networks, and this makes it difficult for developers to understand the whole of systems. Thus, the technologies are required in order to develop distributed systems using a model which allows developers to understand them easily. Mobile agent technologies are expected to respond to these requirements.

A mobile agent is an autonomous software module that can migrate between different computers. A mobile agent is designed and implemented like a human, and mobile agents work together by interactions among them like a human community. Thus, a mobile agent technology is helpful when we develop distributed systems with an easy-to-understand design and implementations for humans. Many researchers have proposed various platforms, applications, and methods to debug with static analyses for mobile agent systems. However, technologies of performance improvement and methods to debug with dynamic analyses for mobile agent systems have unsolved problems. There are especially important problems shown as follows: (1) performance decrement of mobile agent systems caused by Increasing data traffic by transfer of program codes when mobile agents migrate between computers, (2) Difficulties of debugging of mobile agent systems caused by autonomous migrations of mobile agents.

For the first problem, we propose methods on different software architecture layers. The first method belongs to a platform layer. In this approach, we propose program code caching to reduce the data traffic caused by mobile agent migrations. A mobile agent consists of many program codes that define a task executed in each machine they migrate; thus, the mobile agent migration involves the transfer of their program codes. Therefore, our method reduces a number of the transfer of program codes by using program code cache. When agents of same type migrate concurrently, or when agents of different types which have same program codes migrate concurrently, AREs prevent unnecessary transfer of program codes by using caches of these agents. Additionally, when these agents migrate concurrently, the computers which run mobile agent platforms cancel duplicate transfers of same program codes. Therefore, even where many agents migrate concurrently at the same time, the performance degradation of the cache is prevented. This approach can apply to many applications because this approach belongs to a platform layer and differs from an application layer. The second method belongs to an application layer. In this approach, we propose a method that divides a large agent into small agents, keeps links of locations among these agents, and caches these agents in computers on a network. This method is applied to a distributed P2P e-Learning system, and the system is allowed to provide large multimedia contents smoothly.

For the second problem, we propose a remote debugger for mobile agent systems. Mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. Since many mobile agents work concurrently and migrate to many computers in a mobile agent system, when a problem occurs in a mobile agent, a programmer must find this mobile agent with difficulty from many mobile agents and many AREs.

Since a mobile agent migrates from a computer to another computer, in order to debug this mobile agent, it requires connections to these computers and functions of remotely debugging. Since a mobile agent migrates even when a programmer is debugging this mobile agent, the remote debugger must change a destination of a connection to a computer which the agent migrates to. Additionally, since the behavior of mobile agent is affected from interactions with other mobile agents, even if a mobile agent executes a same program of a task, then a result of it is not always same. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced. Thus, we discuss problems of mobile agents for debugging, and propose a remote debugger in order to solve these problems. Our remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system. The experimental result shows that our debugger for mobile agent systems finds bugs effectively.

Acknowledgments

First, I would like to thank my adviser, Prof. Takao Kawamura and Prof. Kazunori Sugahara, for their wonderful way of supervising my thesis. They have given me every chance to grow as a person and always motivated me to try out new things since I was an undergrad. I have learned the beauty of research and education from Prof. Takao Kawamura. I also have learned the great thing about making things and organizing of a laboratory from Prof. Kazunori Sugahara. I would like to thank Dr. Kenichi Takahashi who has carefully given me many skills such as designing a research, writing a paper, and approaching issues, to live as a researcher. Thanks to Dr. Toshihiko Sasama who introduced Ruby to me. Ruby is my most favorite programming language which sparked my desire to know more about information engineering. Thanks to Dr. Masaki Ito who taught me the importance of community of researchers. I thank Dr. Shinichi Motomura who have helped me as my senior as doctoral course. Working with them was a great pleasure. Many thanks for all.

I thank Shinya Ootagaki, Shin Osaki, Hideki Huwahara, Taku Nadamoto, Tomoyuki Mishima, Tadafumi Hayakawa, Kengo Onbe, Yuuki Niimoto, Takuya Murakami, Takashi Hirata, Daisuke Yamamoto, Shoji Oosedo, Daiki Okamoto, Takuya Fujino, Toshihiko Heike, Yuusuke Hamada, Kazunari Kuramochi, Youichirou Awata Takayuki Oonishi, for useful discussions and meetings. They are successive agent research team of students of the laboratory. This thesis is

built on their research.

I also would like to thank the open source community. Software developed for this research includes many products of the open source community, and the software not easily developed without their products. Their products are extremely useful. I want to return the results of this research as an open source software to the open source community too.

Last but not least, I would like to express my warm thanks to my family. This thesis would not have been written without the unconditioned support of my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Distributed systems are becoming increasingly complex and massive with the advance of computer networks, and this makes it difficult for developers to understand the whole of systems. Thus, the technologies are required in order to develop distributed systems using a model, which allows developers to understand them easily. Mobile agent technologies are expected to respond to these requirements.

A mobile agent is an autonomous software module that can migrate between different computers. A mobile agent is designed and implemented like a human, and mobile agents work together by interactions among them like a human community. Thus, a mobile agent technology is helpful when we develop distributed systems with an easy-to-understand design and implementations for humans. Many researchers have proposed various platforms, applications, and methods to debug with static analyses for mobile agent systems. However, technologies of performance improvement and methods to debug with dynamic analyses for mobile agent systems have unsolved problems. There are especially important problems shown as follows: (1) performance decrement of mobile agent systems caused by Increasing data traffic by transfer of

program codes when mobile agents migrate between computers, (2) Difficulties of debugging of mobile agent systems caused by autonomous migrations of mobile agents.

For the first problem, we propose methods on different software architecture layers. The first method belongs to a platform layer. In this approach, we propose program code caching to reduce the data traffic caused by mobile agent migrations. A mobile agent consists of many program codes that define a task executed in each machine they migrate; thus, the mobile agent migration involves the transfer of their program codes. Therefore, our method reduces a number of the transfer of program codes by using program code cache. When agents of same type migrate concurrently, or when agents of different types which have same program codes migrate concurrently, AREs prevent unnecessary transfer of program codes by using caches of these agents. Additionally, when these agents migrate concurrently, computers which run mobile agent platforms cancel duplicative transfers of same program codes. Therefore, even where many agents migrate concurrently at the same time, the performance degradation of the cache is prevented. This approach can apply to many applications because this approach belongs to a platform layer and differs from an application layer. The second method belongs to an application layer. In this approach, we propose a method that divides a large agent into small agents, keeps links of locations among these agents, and caches these agents in computers on a network. This method is applied to a distributed P2P e-Learning system, and the system is allowed to provide large multimedia contents smoothly.

For the second problem, we propose a remote debug for mobile agent systems. Mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. Since many mobile

agents work concurrently and migrate to many computers in a mobile agent system, when a problem occurs in a mobile agent, a programmer must find this mobile agent with difficulty from many mobile agents and many AREs. Since a mobile agent migrates from a computer to another computer, in order to debug this mobile agent, it requires connections to these computers and functions of remotely debugging. Since a mobile agent migrates even when a programmer is debugging this mobile agent, the remote debugger must change a destination of a connection to a computer which the agent migrates to. Additionally, since behavior of mobile agent is affected from interactions with other mobile agents, even if a mobile agent executes a same program of a task, then a result of it is not always same. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced. Thus, we discuss problems of mobile agents for debugging, and propose a remote debugger in order to solve these problems. Our remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system. The experimental result shows that our debugger for mobile agent systems finds bugs effectively.

This thesis is organized as follows. Chapter 2 proposes cache mechanism for migrations of mobile agents in a platform layer. Chapter 3 proposes an interaction scheme among mobile agents on distributed P2P e-Learning system in an application layer. These propositions are in order to solve the first problem. Chapter 4 proposes a remote debugger for mobile agent systems in order to solve second problem. Chapter 5 draws conclusions of this thesis.

Chapter 2

Performance Improvement in Platform Layer

Mobile agents are able to migrate among machines to achieve their tasks. This feature is attractive to design, implement, and maintain distributed systems because we can implements both a client- and server-side programming in one mobile agent. It, however, involves the increase of data traffic for mobile agent migrations. In this chapter, we propose program code caching to reduce the data traffic caused by mobile agent migrations. A mobile agent consists of many program codes that define a task executed in each machine they migrate; thus, the mobile agent migration involves the transfer of their program codes. Therefore, our method reduces a number of the transfer of program codes by using program code cache. We have implemented our method on a mobile agent framework called Maglog and conducted experiments on a meeting scheduling system.

2.1 Introduction

Advances of wireless connection technologies enable us to connect to the Internet in anywhere and anytime. Nowadays, not only personal computers but also various appliances such as personal computers, mobile phones, car navigation systems, and televisions, are connected to the Internet. Thus, ubiquitous network environment would be realized in near future. In a ubiquitous network environment, numerous their appliances will communicate and work together for providing helpful services to us. Such an environment requires us to implement complex network-based systems.

Socket programming and Remote Procedure Call (RPC) are most traditional one to implement such a network-based system. We, however, need to implement two programs, client- and server-side program. This complicates the maintenance of the system because the modification of the system involves the modification of both a client- and server-side program. As the countermeasure of this complication, many researchers pay attention to mobile agent systems.

A mobile agent system is constructed from many mobile agents. A mobile agent is able to migrate among machines to achieve its tasks. We can use mobile agent migration instead of communications between the server and the client. Since the mobile agent can continue its works over the machines, we do not need to elaborate a client- and server-side program pair. Thus, we can implement a server- and client-side program as one mobile agent. This enables us to implement a network-based system in a ubiquitous computing environment without being aware of communications APIs and protocols. Such a mobile agent system is attractive to design, implement and maintain a distributed system; however, it involves the increase of data traffic caused by mobile agent migrations.

Therefore, many researchers have proposed to mitigate data traffic caused by mobile agent migrations. [1] proposes a method to select an efficiency migration route calculated from the round-trip times and average packet loss rate of ping messages. [2] proposes that a machine transfers program codes before the migration of the agent happens actually. It is, however, difficult for a machine to predict where the agent migrates. [3] proposes to use a multicast message to distribute a mobile agent. This enables to use network bandwidth efficiently because the mobile agent is distributed on some machines simultaneously. However, the situation the multicast message enables to be utilized is limited. Thus, these approaches restrict situations to be applied within a specific situation and/or network environment. Further, these approaches require us to change the behaviors of each mobile agent.

Object Management Group (OMG) [4], which is a standards consortium of a mobile agent, mentions to apply a cache mechanism to mobile agent migration. AgentSpace [5] also mentions a cache mechanism. They, however, do not show the detail of the mechanism and protocols. Further, the effectiveness of a cache mechanism is not well-studied even if some agent systems such as [6] and [7] implement a cache mechanism. Thus, in this chapter, we discuss the details of a cache mechanism and protocol.

Before we discuss a cache mechanism and protocol, we should discuss mobile agent systems because the cache mechanism and protocol are for mobile agent migrations. Therefore, we define the internal structure of a typical mobile agent in Section 2.2. After that, in Section 2.3, we design the cache mechanism and protocol for an effective mobile agent migration. We implement our mechanism on a mobile agent framework called Maglog, and we show its effectiveness on various patterns of agent migrations and a practical application in Section 2.4. Finally, we conclude the chapter in Section 2.8.

2.2 Mobile Agent System

A mobile agent system is usually structured from mobile agents and Agent Runtime Environments (AREs), as shown in Figure 2.1. This structure is a typical structure mentioned in [8].

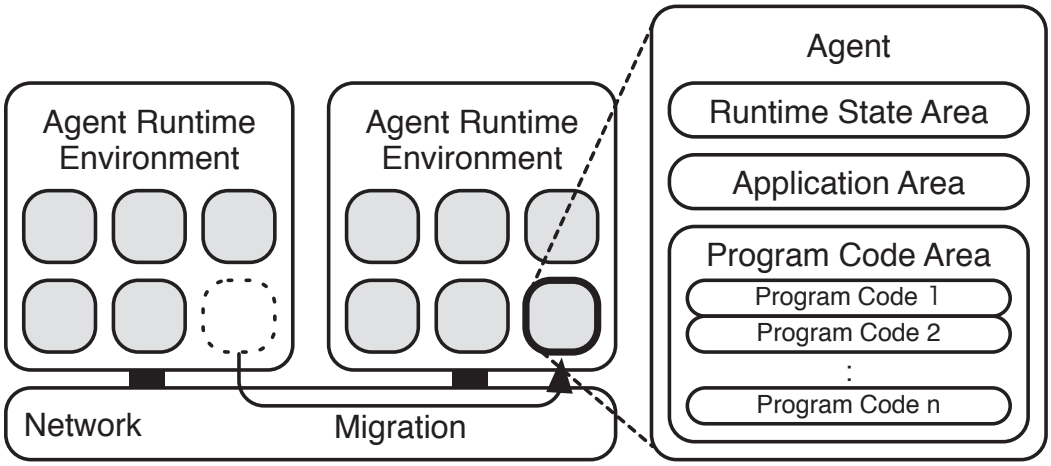


Figure 2.1: Overview of a mobile agent system.

An ARE is installed in each machine distributed on a network, and provides some functions to help the executions of tasks of mobile agents. A mobile agent migrates among the network by using the function of the AREs, and accomplishes its task using resources distributed on the network. For example, when a mobile agent tries to make a plan of sightseeing, the agent goes on airline company site for the reservation of an air ticket; and then, goes on hotel reservation sites to reserve a room; finally, shows the plan to a user.

As mentioned here, a mobile agent system consists of AREs and mobile agents. Almost functions implemented in AREs are prepared to help the executions of tasks of mobile agents, thus, their functions are almost same in each AREs. However, tasks of each mobile agent are naturally different in each. For example, a mobile agent mentioned above would have a task to make a

trip plan. A mobile agent to adjust meeting schedules would have a task to adjust schedules among attenders. Thus, the program codes implementing in each mobile agent are different.

Considering the installation of a cache mechanism, there are two candidates a cache mechanism to be installed in, AREs and mobile agents. However, it is unsuitable to install a cache mechanism in mobile agents, because each mobile agent has different tasks. It will involve the change of the implementation of each mobile agent as same as approaches proposed in [1–3]. On the other hand, when we consider the installation of a cache mechanism into AREs, we do not need to pay attention to the difference of each AREs because almost AREs are composed from common functions. Therefore, we design a cache mechanism to install in AREs. Thus, programmers of a mobile agent application need not take care of a cache mechanism because the cache mechanism is working on the ARE layer.

2.2.1 Internal Model of a Mobile Agent

We propose to apply a cache mechanism for a mobile agent migration. What enables to be cached? To answer this question, we have to discuss the internal model of a mobile agent. We can make the typical model of a mobile agent from the discussion in [4], [8–10]. This model is much conformed to a lot of mobile agent systems such as [11], AgentSpace [5], Aglets [12], and so on.

A mobile agent consists of a runtime state, application data, and program codes. The runtime state manages variables such as call stack pointers, program counters, and so on. Data in the runtime state constantly change while a mobile agent is working. The application data are related to an application. For example, in a meeting scheduling system, the application data may include users' preferences and attenders' list of meetings. These data depend on each

mobile agent. The program codes represent the tasks of a mobile agent. The mobile agent proceeds with its tasks by the execution of program codes.

When we try to apply a cache mechanism to mobile agent migration, we have to classify them into cacheable or un-cacheable data. A cache mechanism stores the duplication of original data, the duplication data are reused later instead of getting the original data. If the duplication data change from the original data, its duplication is useless anymore.

A runtime state is meaningless to be cached because it constantly changes according to mobile agent's behavior. Thus, we conclude a runtime state is un-cacheable. A program code is cacheable because it does not change. However, we cannot conclude application data is cacheable or un-cacheable because it depends on an application. If we deal with the application data as the object of a cache mechanism, we have to entrust the classification of the application into cacheable or un-cacheable to programmers of a mobile agent application. This would not be preferred because programmers want to devote only to the implementation of an application. Thus, we focus on only program codes as the object of a cache mechanism.

2.2.2 Migration of a Mobile Agent

When an agent migrates from a source ARE to a destination ARE, the source ARE has to transfer a runtime state, application data and program codes to the destination ARE. The steps for agent migration are usually modeled as Figure 2.2.

1. A source ARE connects a destination ARE an agent tries to migrate to the destination ARE.

2. The destination ARE responses whether it allows the migration of the

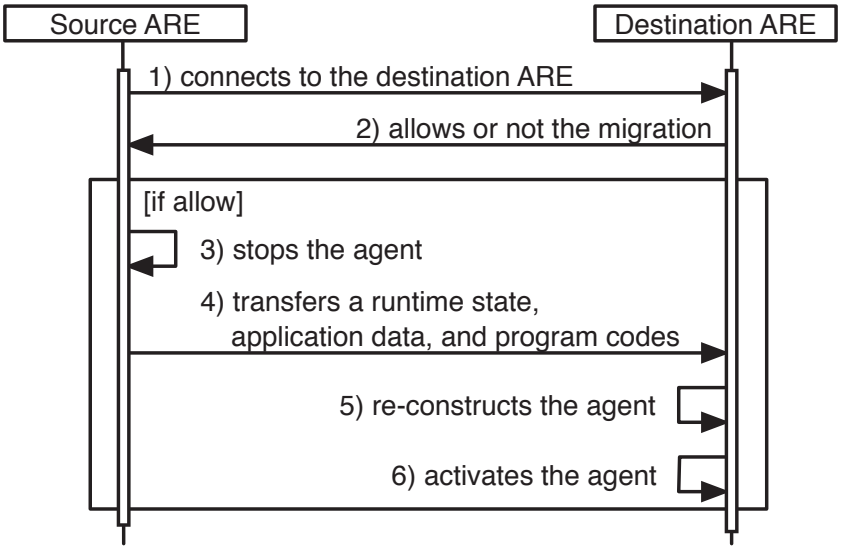


Figure 2.2: Steps for Mobile Agent Migration.

agent or not.

3. If the destination ARE allows the migration, the source ARE stops the agent.

4. The source ARE transfers a runtime state, application data and program codes to the destination ARE.

5. The destination ARE reconstructs an agent from program codes, a runtime state, application data received from the source ARE.

6. The destination ARE activates the agent.

When we apply a cache mechanism to mobile agent migration, we have to add steps for a cache mechanism in these steps.

2.3 Cache Mechanism for Single Agent Migration

A cache mechanism stores the duplication of original data (cached data), and provides cached data to a requester instead of getting the original data. A cache mechanism is used as CPU cache, web cache, DNS cache, and so on. These mechanisms cut the transfer of data from a requestee (e.g., web server in web cache) to a requester (e.g., web browser); thus, cached data are managed by the requester. On the contrary, data are transferred from a requester to a requestee in mobile agent migration because a mobile agent migrates from a source ARE (requester) to a destination ARE (requestee). When we apply a cache mechanism to mobile agent migration, a source ARE, first, has to check which data is cached. As discussed in Section 2.1, the object of our cache mechanism is program codes. Therefore, we need to identify each program code for the check of a program code cached in a destination ARE.

2.3.1 Identification of Program Code

One mobile agent usually consists of many program codes. Some of these program codes are implemented by the programmer of a mobile agent application, but some may be OSS (Open Source Software) downloaded from the Internet. Further, different programmers may use a same name in different program codes. If we create the identifier of each program code from a name such as a file name, a function name, and so on, it may cause cache poisoning. If cache poisoning occurs, a mobile agent migrated cannot work well anymore. The most import point is that anomaly can harm a mobile agent easily by causing cache poisoning. Thus, it is risky to identify program codes by a name. In order to avoid cache poisoning, program code identifiers must satisfy following

conditions:

1. It is difficult to create same identifier from different program codes.

2. Identifiers of same program codes must be same.

In order to satisfy above conditions, we use a hash value created from a content of program codes as a program code identifier. Since the identifier is a hash value created from a content of program code, the identifier changes if a program code changes; the identifiers are same whatever program codes are same. We use the Secure Hash Algorithm 1 (SHA-1) [13] as a hash function. SHA-1 takes a program code less than 2^64 bits in length and can produce a 160-bit identifier. The probability of that same hash value is created from different program codes is extremely small even when anomaly tries to find it. Even if the collision of identifiers occurs, it would throw an error. If an error occurs, the identifier of the collided two program codes is marked as pollution. A program code marked as pollution is regarded as if it is not cached. This prevents an anomaly from causing cache poisoning attacks.

2.3.2 Local Cache Table

We design a local cache table to store the pair of a program code and its identifier. The local cache table is prepared in each ARE. All program codes are stored in the local cache table with their identifiers. An ARE knows whether the program code corresponding to an identifier is cached or not to see the local cache table, and gets its program code.

2.3.3 Mobile Agent Migration with a Program Code Cache

In order to apply a cache mechanism to mobile agent migration, we add steps to check program codes cached in a destination ARE before the transfer of program codes. Program codes are transferred with a runtime state and application data at the step 4 in the typical model of mobile agent migration in Figure 2.2. Therefore, we extend the step 4 as following:

4-1 The source ARE transfers a runtime state, application data, and the identifiers of program codes to a destination ARE.

4-2 The destination ARE refers its local cache table, finds program code identifiers which are not listed in, and requests their program codes to the source ARE.

4-3 The source ARE finds requested program codes from its local cache table, and transfers their program codes to the destination ARE.

4-4 The destination ARE receives the program codes and stores them with their identifiers into the local cache table.

A first migration would involve the transfer of almost program codes because their program codes are not listed in the local cache table of a destination ARE. However, after that, the identifiers of their program codes are listed in the local cache table. Therefore, second migration does not involve their transfer. This mitigates the increase of data traffic caused by mobile agent migrations.

2.4 Evaluation of Single Agent Migration

2.4.1 Experimental Environment

We have implemented our cache mechanism on Maglog [11], which is a Java-based mobile agent framework. The ARE of Maglog has a Prolog-to-Java source code translator and a Prolog interpreter. The agent is implemented by Prolog language, and works by using the Prolog interpreter. Therefore, the ARE can access the runtime state of an agent from a Prolog interpreter's internal states. Program codes are Java byte codes translated from a program implemented by Prolog. Application data are the serialized data of a Java object by Java Object Serialization [14]. The identifier of a program code is generated from Java byte codes by using java.security.MessageDigest [15]. A local hash table is implemented by java.util.HashMap [16].

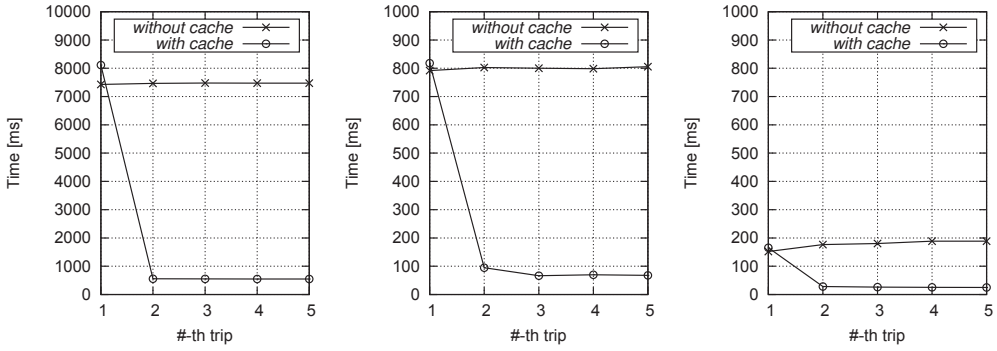
In experiments, we used a computer of Intel Core i7-2600 Processor (8 MiB Cache, 3.40 GHz) and 32 GiB RAM. Each ARE of Maglog runs on JRE 6 [17] on Debian GNU/Linux 6.0.4 (Kernel 2.6.32-5-686-bigmem) and is connected via a network. The network is constructed by that a physical NIC is assigned multiple IP addresses by IP Aliasing of Linux, and each IP address is connected by emulated Ethernet which can change the speed to 10 BASE-T, 100 BASE-T, and 1000 BASE-T by Dummynet [18]. A mobile agent consists of 4096 program codes, the size of each program code is 1024 KiB, the size of a runtime state is 70 KiB, and the size of application data is 0 KiB.

2.4.2 Overhead of Cache Mechanism

Our cache mechanism involves transferring of the identifiers of program codes, and checking of a local cache table. Firstly, we should clear the overhead of the cache mechanism. Therefore, we measured the time between an agent tries

to start migration and finishes it.

The results are shown in Figure 2.3a, 2.3b, and 2.3c. In this figure, with cache represents the migration times in where the cache mechanism is installed; without cache represents it is not installed. The migration time of with cache at the first migration is slower than without cache. This difference means the overhead of the cache mechanism. The overhead is only 8.5% in 10 BASE-T; 3.2% in 100 BASE-T; 8.1% in 1000 BASE-T. Thus, the overhead of the cache mechanism is relatively small. After the second migration, the migration times are dramatically improved in with cache. In 10 BASE-T, the migration time of with cache is 7.4% of without cache; 8.4% in 100 BASE-T; 13.3% in 1000 BASE-T.



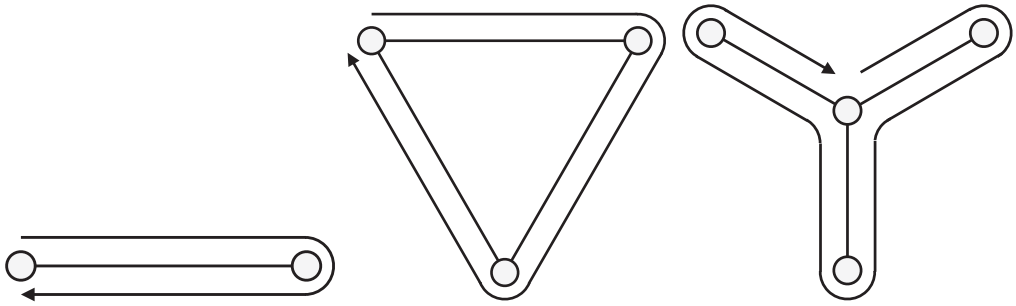
(a) 10 BASE-T (b) 100 BASE-T (c) 1000 BASE-T

Figure 2.3: Migration times on one-way pattern.

2.4.3 Evaluations on Agent Migration Patterns

In order to clear the impact on various patterns of mobile agent migration, we measured the time of agent migrations on a shuttle, a round, and a star pattern shown as Figure 2.4.

In the shuttle pattern, cache miss occurs at the outward migration in the

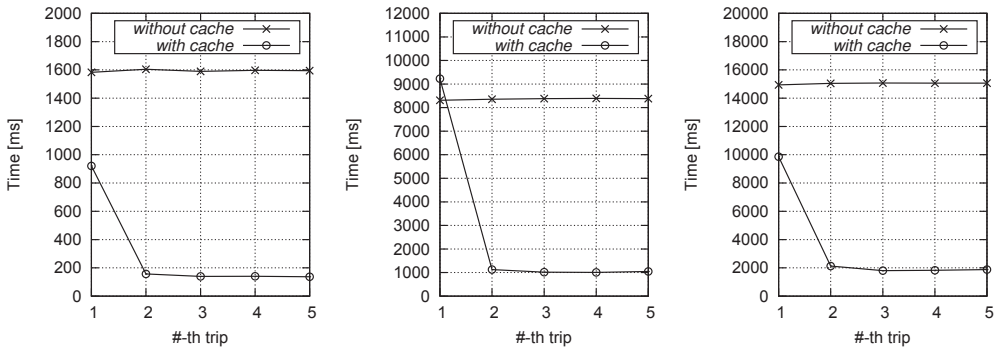


(a) Shuttle Pattern. (b) Round Pattern. (c) Star Pattern.

Figure 2.4: Agent Migration Patterns.

first trip; but it not occurs in other trips. In the round pattern, cache miss occurs in all the migrations of the first trip except the last migration; but it not occurs in all the migration after the second trip. The star pattern is repetition of a shuttle pattern.

The experimental results conducted on 100 BASE-T are shown in Figure 2.5a, 2.5b, and 2.5c. As shown in this figure, the migrations of with cache finished faster than without cache even in the first trip except the round pattern. After the second trip, the migration time of with cache are improved further in all patterns.



(a) Shuttle Pattern. (b) Round Pattern. (c) Star Pattern.

Figure 2.5: Migration times on each migration pattern.

2.4.4 Evaluation on Random Agent Migration

We evaluate the agent migration time when an agent migrates to somewhere randomly. In this experiment, we prepared 10 AREs. The agent migrates other ARE that is randomly selected from 9 AREs (10 - current ARE). Figure 2.6 shows results of 100 migrations.

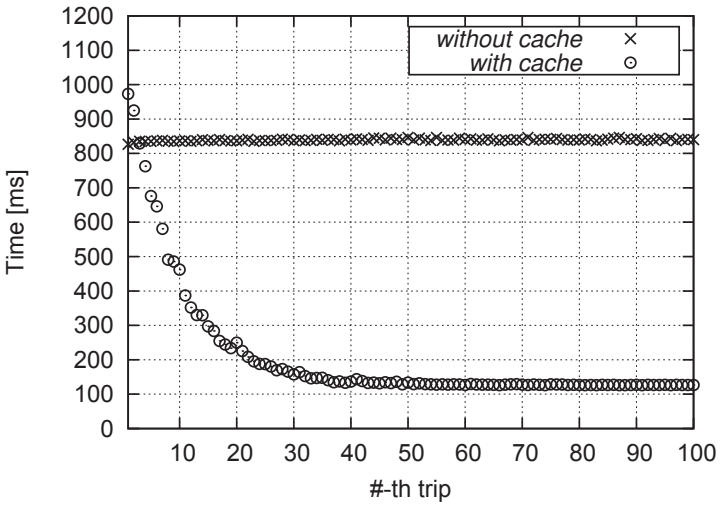


Figure 2.6: Results when Mobile Agents Migrate Randomly.

As shown in this figure, the migration times of without cache are around 840 msec even if the number of times of migrations increases. In contrary, the migration times decreases in with cache. It is approaching 126 msec.

2.4.5 Evaluation on a Meeting Scheduling System

We evaluate our mechanism on a meeting scheduling system [19, 20]. The meeting scheduling system is developed by Maglog and used in our university. In this system, a user's agent of an inviter migrates among AREs to gather the schedules of invitees. If there is no candidate day by a conflict with the schedules of invitees, a user's agent negotiates with invitees to decide the day

of a meeting. Thus, a user's agent migrates over AREs to gather the schedules of invitees, to negotiate with invitees, and to decide the day of a meeting.

In this experiment, we prepared an automatic manipulation program which substitutes for the operations of a human being. The program selects invitees randomly, tries to decide the day of a meeting with the invitees, makes users' schedule, replies its schedule to an inviter, and sometime makes a concession to the inviter, at a random time. We conducted this experiment on 11 computers that are installed Intel Pentium 4 processor (3.0 GHz) and 1 GiB RAM and are connected via 100 BASE-T Ethernet. An ARE of Maglog runs on JRE 1.5 on Turbolinux 10 (Kernel 2.6.0) on this computer.

The experimental result is shown in Figure 2.7. As shown in figure, even in the decision of first meeting day, the performance of the system of with cache is almost same with without cache. From the second decision, it is about 52% of without cache. This result shows that the cache mechanism is efficient in not only toy problems but also practical mobile agent applications.

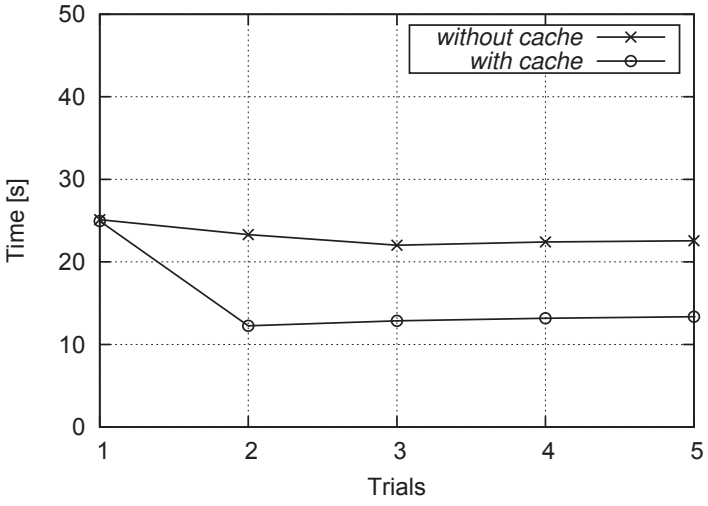


Figure 2.7: Result on a Meeting Scheduling System.

2.5 Cache Mechanism for Multiple Agent Migration

2.5.1 Agent Migration Session

We define a set of procedure from start to finish of an agent migration as a migration session. A runtime state area and an application area have to be transferred at every migration session because these areas are not cacheable data. However, program codes stored in a program code area has a possibility of that are cached in a destination ARE.

Therefore, in a migration session, the source ARE transfers firstly a runtime state area and an application area, and the identifiers of program codes to the destination ARE. Next, the destination ARE receives and saves the runtime state area and the application area, and identifiers of program codes; and the destination ARE checks whether the program codes have cached by using identifiers of program codes. Finally, the destination ARE restores an agent from a runtime state area and an application area, program codes; and the destination ARE finishes the migration session. Thus, the cache mechanism reduces the wasteful transfer of program codes.

2.5.2 Temporally-overlapped Migration Sessions

The program code area consists of multiple program codes. Additionally, in a large-scale system, the number of agents will increase, and a great number of agents work concurrently in the system. Thus, if multiple program codes are transferred after checking that they do not exists in the destination ARE, checking multiply program codes all at once is more efficient than checking singly. However, if a great number of agents migrate concurrently to one des-

tination ARE, such as a meeting pattern [21], requests for transfer of program codes are send multiply to source AREs of each migrating agents. Requests for transfer of program codes are send multiply to the source AREs of each migrating agents. As a result, same program codes are transferred multiply, and a significant performance degradation of cache mechanism occurs.

2.5.3 Cancellations of Transfers of Program Codes

In order to prevent duplicated transfers of program codes, our proposed method employ a mechanism that can cancel duplicated sent requests of transfer of program codes. The step of our method is as follows:

1. The source ARE sends a runtime state area and an application area, and identifiers of program codes.

2. The destination ARE checks that whether program codes have cached by using received identifiers of program codes and sends a list of identifiers of un-cached program codes.

3. The source ARE sends program codes requested by the destination ARE.

4. The destination ARE stores the received program codes into caches and sends cancellation messages to the source AREs that are requested program codes but does not send the program codes yet.

5. The source AREs that are received the cancellation message cancel transfer of canceled program codes.

Thus, transferring duplicated program codes from multiple source AREs to a destination ARE can be prevented.

2.5.4 Cache Mechanism Using GAP

In order to reduce data traffic when many mobile agents migrate at same time, we propose an algorithm using GAP (generalized assignment problem) as follows.

$$\begin{aligned} \text{min.} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m}\left|\sum_{j=1}^{n} a_j x_{ij} - s_i\right| \\ \text{s. t.} \quad & \sum_{i=1}^{m} x_{ij} = 1,\ \forall i \in I,\ \forall j \in J, \\ & x_{ij} \in \{0,1\},\ c_{ij} \in \{0,\infty\} \end{aligned} \tag{2.1}$$

$$s_i = \frac{\sum\{a_1,...,a_n\}}{\sum\{b_1,...,b_m\}} b_i,\ s_i \geq \sum_{j=1}^{n} a_j x_{ij} \tag{2.2}$$

- $I = \{1,...,m\}$: A set of source nodes, the size is m.
- $J = \{1,...,n\}$: A unique set of program codes, the size is n.
- x_{ij}: $0-1$ valiable which is 1 when a source node i transfers the program code j.
- a_j: The data size of the program code j.
- b_i: Connection speed between the source node i and the destination node.
- s_i: The capacity of data traffic size between the source node.
- c_{ij}: The cost valiable which is 0 when the source node i can transfer the program code j; $infty$ otherwise.

This equation can be solved with optimized value by greedy algorithm.

2.6 Evaluation of Multiple Agent Migration

2.6.1 Experimental Environments

We implemented our migration mechanism on Maglog (Mobile AGent system based on proLOG) [22], which is our proposed mobile agent framework. Maglog is implemented in Java and runs on any platform providing Java Runtime Environment (JRE). A mobile agent is a Java Object, which can run concurrently by using threads. A program code identifier is generated by a hash function (SHA-1) from a Java Bytecode. Agents are converted from Java Instance to binary array by using Java Object Serialization Technology, and are able to migrate between AREs via WebSocket Protocol. In order to show the effectiveness of our proposed method, we implemented three types of mechanisms, which are shown as follows:

- Non-cache: This method does not cache program codes. The program codes are transferred at every migration.
- Cache: This method caches program codes and does not execute cancellations.
- Cache-cancel: This method caches program codes and executes cancellations.

A configuration of the computer used in the experiments is as follows: Intel Core i7-2600 Processor (8 MiB Cache, 3.40 GHz), 32 GiB RAM, 1000 BASE-T NIC, and Debian GNU/Linux 6.0.0 (Kernel 2.6.32-5-686-bigmem). The NIC is assigned by multiple IP addresses with IP Aliasing. The bandwidths of each communication path between these IP addresses are configured with DummyNet [18]. The ARE of Maglog runs on a Java Platform Standard Edition 6 Development Kit.

2.6.2 Evaluation on Load Distribution System

In order to evaluate effectiveness of our proposed method, we conducted an experiment on a load distribution system, which is developed with a mobile agent framework. In this system, agents are distributed equally to each node in order to balance a load of the system. The agents are given different initial parameters, and solve a problem with a same algorithm. When a new node is added to this system, the agents are re-distributed equally in order to balance a load including the new node. At that time, multiple agents having tasks migrate concurrently to the new node at the same time. In this experiment, we measured processing time of scale-out procedures about non-cache, cache, and cache-cancel. We set the bandwidth of communication path to 100 Mbps and increased the number of nodes from 1 to 16.

Figure 2.8 shows experimental results. When the number of nodes is increased from 15 to 16, cache improves the performance by 8% than non-cache, and cache-cancel improves the performance by 58% than non-cache. Cache-cancel improves the performance of cache as the number of nodes increases; and if additionally increasing the number of nodes, the difference in performance between non-cache and cache will be greater.

Figure 2.9 shows experimental results for the GAP approach. GAP approach can reduce drastically data traffic as compared to other approaches.

2.7 Related Works

Several researchers have proposed a method that improves performance of a mobile agent system. [23–25] have proposed that agents who communicate remotely migrate to same node in order to communicate locally. [26] has proposed that agents determine whether migrate by size of data communication

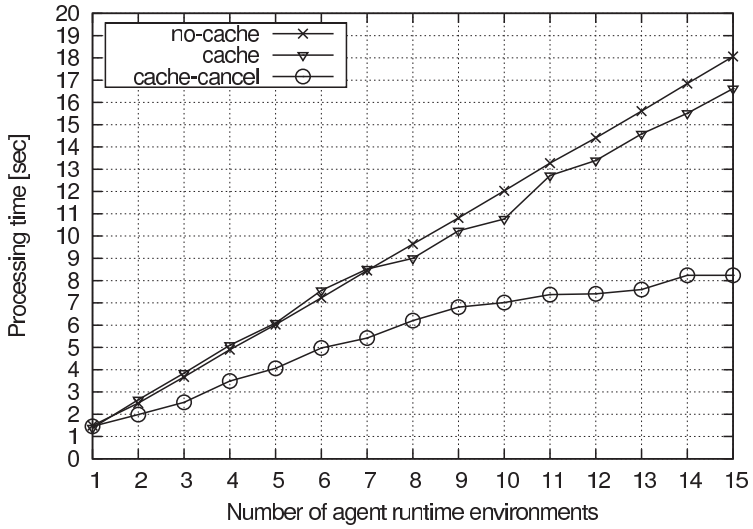


Figure 2.8: The time required for processing of a load distribution.

traffic that is compared interactions with migrations.

MASIF Specification [4] shows a method that sends only program codes, which do not exist at a destination ARE. The method checks whether program codes exist at a destination ARE by program code names. However, its design and implementation does not defined well as a specification. [27] has proposed that different agents can share program codes by using hash values generated from program codes as identifiers of program codes. However, it is not consider performances, when many agents migrate to one destination ARE at the same time.

2.8 Conclusion

In this chapter, we discussed to apply a cache mechanism to mobile agent migration. Our cache mechanism does not influence on the implementation of mobile agent applications because it works on an agent runtime environment. We implemented the cache mechanism on Maglog, which is a mobile agent

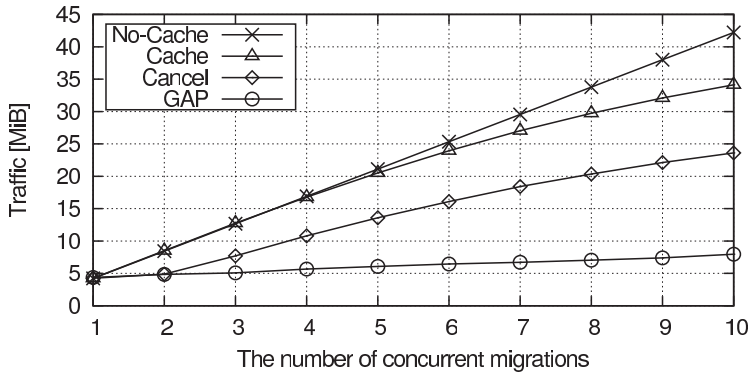


Figure 2.9: The data traffic when multiple agents migrate.

framework, and conduct experiments. The result on a meeting scheduling system shows that the mechanism of the cache enables to improve its performance by 52%.

Additionally, when multiple agents migrate from multiple sources to one destination at the same time, the proposed mechanism prevents duplicate transfers of program codes by using cancellations of requests of transfers of program codes. We implemented our mechanism on Maglog and conducted experimental results on a load distribution system. In this experiment at scale-out that increases the number of nodes from 1 to 16, our mechanism improved the performance of the system by up to 54.4%.

Chapter 3

Performance Improvement in Application Layer

Nowadays, many e-Learning systems are widely deployed in educational schools. Typical e-Learning systems are implemented as the client-server model, but it requires an expensive central server for scalability. Thus, in order to reduce the load on the central server, we have been developing a pure P2P-based distributed e-Learning system. All learning contents are realized as mobile agents in this system, which works by interactions of mobile agents. However, it is not enough that data are simply distributed learning contents into nodes. Since the size of multimedia content is different, it makes imbalances of resource usage among nodes. Additionally, users cannot play smoothly multimedia contents. Therefore, we propose a method that divides a multimedia content into small fragments, keeps links of locations among these fragments, and caches these fragments in nodes.

3.1 Introduction

E-Learning systems, especially asynchronous Web-Based Training systems (WBT) [28,29] are very popular. A WBT does not require interactions with instructors and allows a learner to study on his/her own time and schedule. The mainstream e-Learning system is based on a client-server model. Although the client-server model has an advantage of easy construction and maintenance, however, the client-server model requires expensive initial investments to execute management and to offer the contents by the server machine. Therefore, in order to reduce the load on the central server, we have proposed a pure P2P-based distributed e-Learning system. All learning contents are realized as mobile agents in this system, which works by interactions of mobile agents.

In the past, the Internet users usually communicate with each other by using texts or images mainly; but nowadays, they interact with each other by using multimedia. Thus, multimedia distribution services (e.g., iTunes Store [30] and Netflix [31].) that provide music and videos via the Internet are become popular. It is same even in the field of education. E-Learning systems using multimedia contents are also popular because multimedia contents increase efficiency of learning. By using multimedia contents, e-Learning systems enable to provide more understandable learning contents compared with texts and/or images.

However, it is not enough to simply distribute learning contents to nodes in a pure P2P-based distributed e-Learning system, because the size of a multimedia content is different. It makes imbalances of resource usage among nodes. Additionally, users cannot play smoothly multimedia contents because the load of providing multimedia contents concentrates on a node which manages popular multimedia contents. Therefore, in this chapter, we describe basic features of our pure P2P based e-Learning system and propose a method of multimedia

contents management. The method divides a multimedia content into small fragments to balance usage of nodes' storage, keeps links of locations among these fragments to reduce costs of look-up for learning contents, and caches these fragments in nodes.

The rest of this chapter is organized as follows. Section 3.2 describes our pure P2P based distributed e-Learning system. Section 3.3 presents our method that manages multimedia contents. Section 3.4 describes the experimental results of the method. Section 3.5 discusses related work. Finally, Section 3.6 draws the conclusions.

3.2 Pure P2P-based Distributed e-Learning System

3.2.1 Design Concepts

We have been proposing a pure peer-to-peer-based distributed e-Learning system that is designed from gathered inexpensive computers. Our e-Learning system can distribute not only data but also processes into computers by mobile agent technologies.

Data in our e-Learning system consist of learning contents and personal information of users, such as account information, history of studies, answers, score, etc. Processes consist of providing of contents, displaying of contents, scoring of answers, login/logout, etc. In order to distribute data into nodes, our e-Learning system uses a DHT (Distributed Hash Table). However, a node that has popular learning contents gets concentrated requests of learning contents from many nodes. The node may not have enough performances to provide learning contents to many nodes since the system consists of inexpen-

sive computers. In order to solve this problem, in our e-Learning system, a learning content is managed by a mobile agent that has ability to migrate to the requested nodes and to process own tasks. Thus, these processes are distributed into users' node. Because these mobile agents work only in a sandbox of user's node, they do not cause problems about a security on user's node.

3.2.2 Mobile Agents

Our e-Learning system is mainly constructed from mobile agents in order to realize scalability and understandability of the system. In our e-Learning system, there are following mobile agents (Figure 3.1).

- EA (Exercise Agent)
- CA (Category Agent)
- UA (User Agent)
- GA (Group Agent)
- NA (Network Agent)
- IA (Interface Agent)

There are various methods for scoring users' answers of the learning contents. However, a general e-Learning system provides only several templates of the learning contents and methods of scoring. On the other hand, our e-Learning system has no limits of methods of scoring learning contents and can provide methods to make high-flexible learning contents because a learning content is managed by a mobile agent.

A mobile agent can migrate to users' node and execute optional processes at the users' node. Therefore, a mobile agent as a learning content can score

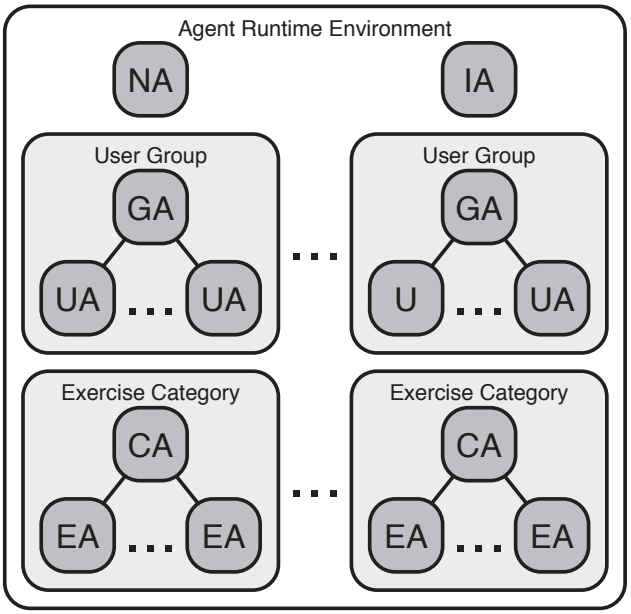


Figure 3.1: Composition of mobile agents on an ARE.

users' answers appropriate to the learning contents. In our e-Learning system, such a mobile agent is named an **EA (Exercise Agent)**. An EA has a learning content. When an EA receives a message of a request of learning, the EA create a clonal EA of self; and a cloned EA migrates to learner's computer, provides a learning content, displays a learning content, scores its answers. After that learning is finished, the clonal EA migrates to node which the original EA exists and informs the original EA that the learning finished.

Generally, a user chooses sequentially learning contents and solves them. For example, a user chooses sequentially learning contents such as 1th, 2nd, 3rd, ..., and n-th. Therefore, in order to reduce a cost of look-up, managing learning contents by a group is more efficient than managing them individually. In our e-Learning system, EAs of a group are managed by **Category Agent (CA)**. A CA has a list of identifiers of EAs, and each identifier is a hash value of learning contents. Additionally, a CA has a category name which is used

as a key of a DHT to look up a location of its CA. A location of a CA can be looked up with its category name from DHT, and a location of an EA can be looked up from a list of identifiers that the CA has. When churn (joining or leaving) of a DHT occurs, a CA and managed EAs by the CA migrates together to a location specified by DHT. Thus, if a user looks up learning content as an EA, the user can obtain effectively the EA by using a category name of a CA and an identifier of an EA.

In our e-Learning system, personal information of a user is stored into a mobile agent called **User Agent (UA)**. An UA has personal information and controls accesses of the information from users. Personal information contains a user name, a password, history of studies, answers, scores, etc. Because the size of history of studies, answers, and scores, becomes large according to the time, UAs are distributed into nodes in order to balance the usage of storages of nodes. When an UA receives a message of a request of login, the UA create a clonal UA of self; and the clonal EA migrates to learner's computer and authorizes the user to join the system by checking a user name and a password. After authorizing the user, the UA stays at the user's computer and has responsibility of updating of personal information (e.g., history of studies, answers, scores) until the logout. Thus, processing of login/logout and recording of scores is executed locally at the user's node and the load of the processes is distributed to user's node.

Generally, a teacher manages scores of students by group such as academic subjects, classes, or courses. Therefore, in order to reduce a cost of look-up of a DHT, managing UAs by a group is more efficient than managing individually them. In our e-Learning system, UAs of a group are managed by **Group Agent (GA)**. Here, relationships between GA and UAs are based on same idea of relationships between CA and EAs. A GA has a group name that is used as a key of a DHT to look up a location of its GA. A location of a GA can

be looked up with its group name from DHT, and a location of an UA can be looked up with a user name that the UA has. When churn (joining or leaving) of a DHT occurs, a GA and managed UAs by the GA migrates together to a location specified by DHT. Thus, if a user looks up an UA, the user can obtain effectively the UA by using a group name of a GA and a user name of an UA.

As previously mentioned, the location of a CA and a GA is managed on an overlay network of a DHT as a pair of a name and a location. In our e-Learning system, this overlay network is provided by agents called **Network Agent (NA)**. One NA exists on one node and has responsibility of providing of an overlay network. A NA checks living of near nodes by that whether a child agent of the NA can migrate to the node. When churn (joining or leaving) of a DHT occurs, NAs send a message of a request of migration to EAs and GAs in order to re-distribute mobile agents and to balance usages of resources. Additionally, a NA provides API of a key-value interface of a DHT and delivers its messages.

Since humans cannot understand messages of mobile agents, mediators between humans and mobile agents are required. In our e-Learning system, there is **Interface Agent (IA)** in order to mediate between a user and mobile agents. An IA contains a GUI component based on a web application, and a user can access a GUI of the IA through a web browser. When a user operates a GUI (e.g., a button, a text box, or etc.), the IA translates from these operations into messages that mobile agents can understand, interacts among other mobile agents, and returns the results to the user.

3.3 Management of Streaming Multimedia Data

3.3.1 Multimedia Content Distribution

If a learning content is a multimedia such as a video, a size of a multimedia will be reached MiB or GiB. Multimedia contents are really different in data size to text contents. Thus, if simply distributing a learning content as a unit of a file or a group, the inequality of resource usage (e.g. CPU time, memory usage, disk usage, network bandwidth, etc.) between nodes occurs. Additionally, a transfer of a text content is finished at short time. However, in the case of a video content, it takes a long time. We have to devise the management of multimedia contents.

Therefore, if a learning content contains multimedia data, we divide the multimedia data into small fragments. Here, we define a multimedia agent (MA) to manage these small fragments. An EA needs not manage multimedia data anymore; but manages only a description of a multimedia and references of locations to these MAs. Thus, the load of EAs becomes low, because only a description and references are required to be transfered. Additionally, the transfer of multimedia data is distributed on each MAs.

3.3.2 Distributed Multimedia Content Streaming

Each fragment is mapped on DHT based on its keys. Since a multimedia content is composed of some fragments, we have to know the location of these fragments on DHT. Therefore, each fragment has the key of a next fragment. Then, each fragment can find next fragment according to its key. However, it needs many messages to find a node which manages next fragment, because a DHT requires a cost of a look-up up to $O(n)$.

Here, n is the total number of nodes. This may impede smooth playing

of the multimedia data. In order to solve this issue, before the multimedia content is required from learners, each fragment finds a node which manages next fragment and records its location. Thus, every fragment is linked in the order of time series of the multimedia data. Consequently, we can reduce messages to find the location of fragments except first fragment's search.

When a node joins or leaves (churn), fragments are transferred to other nodes. If a fragment is transferred, the link between fragments becomes useless anymore. To keep the link, when a node joins or leaves, each fragment records not only the location of a next fragment but also a previous fragment. When a fragment is transferred, the fragment notifies its new location both to a next and previous fragment. Thus, the link between fragments is kept continuously even when a node joins or leaves.

3.3.3 Distributed Multimedia Content Caching

When a user learns using a multimedia content, fragments of the multimedia content temporarily gather in the learner's node. Therefore, this node can be used as a cache node. When a user learns using a multimedia content, the user's node stores the caches of the fragments into own node; and a node that has an original fragment, memorizes references of nodes storing cache. Thus, the node with original fragments has references of caches of fragments. Therefore, this node can balance a self-load like a DNS round robin (Figure 3.2).

In Figure 3.2, a PA is a pointer agent which manages MAs and caches of MAs. A MA is a multimedia agent which holds a fragment of multimedia data. These agents are managed by a EA. Each agent is mapped on an DHT network.

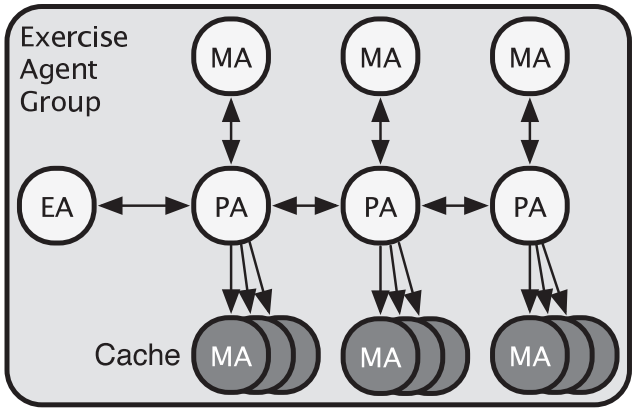


Figure 3.2: Distributed multimedia content caching by mobile agents.

3.4 Experiments

3.4.1 Experimental Environment

Experiments have done with our e-Learning system that runs on computers as shown Table 3.1. Our e-Learning system uses a CAN [32] as the P2P network and is implemented in Java-based mobile agent framework called Maglog [33]. An ARE runs on a computer. A format of a video files for experiments is a FLV (Flash Video) format. The physical network layout is shown in Figure 3.3. In the experiments of Section 3.4.2 to 3.4.5, we have used 12 nodes. In the experiment of Section 3.4.6, we have increased the number of nodes from 1 to 9.

Table 3.1: The experimental conditions.

CPU	Intel Core i5 processor 3.2 GHz
RAM	4 GiB
JRE	1.6
OS	Debian GNU/Linux 5.0.5
Network	100 BASE-T Ethernet

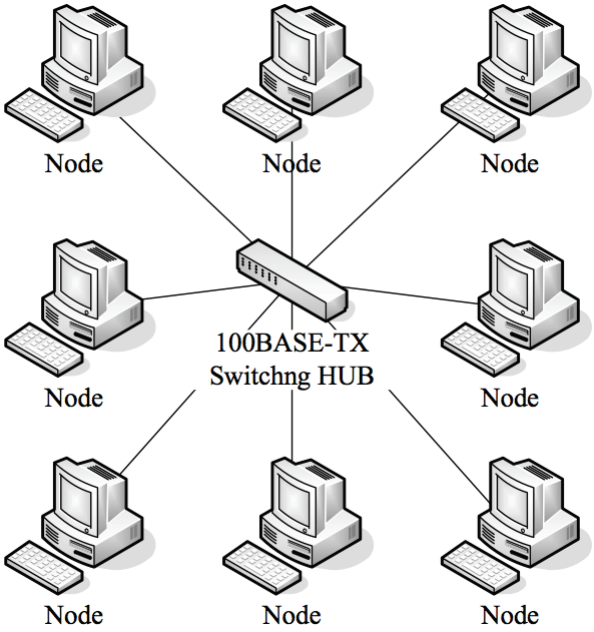


Figure 3.3: The physical network layout of the experimental environment.

3.4.2 Smooth Play of Multimedia Data

In order to show the effectiveness of our proposed method, we have evaluated whether multimedia data are played smoothly. Measuring a difference time length between a time length of multimedia data and an actual playing time length, we can evaluate whether multimedia data are played smoothly. This difference time length is a total pausing time length when the multimedia data are played; and close to zero means that the multimedia content is played smoothly. Thus, we have measured a total pausing time length in the case of *with link* and *without link*.

In this experiment, we have defined an about 16 MiB sized video file which has 21 seconds as multimedia data, and the video file has divided into fragments of 200, 225, 250, 275, or 300 KiB by time series.

Figure 3.4 shows the result of this experiment. In *without link*, a total

pausing time length increases when the division size decreases. For example, when division size is 200 KiB, the total time of pausing is 25 seconds. This is because each MA has to lookup next MA on DHT. In contrary, in *with link*, a total pause time length does not depend on division size. Thus, the multimedia data can be played smoothly. This result shows that as the division size smaller, in other words, as more the multimedia data are divided and distributed in a DHT network, our proposed method is more effective.

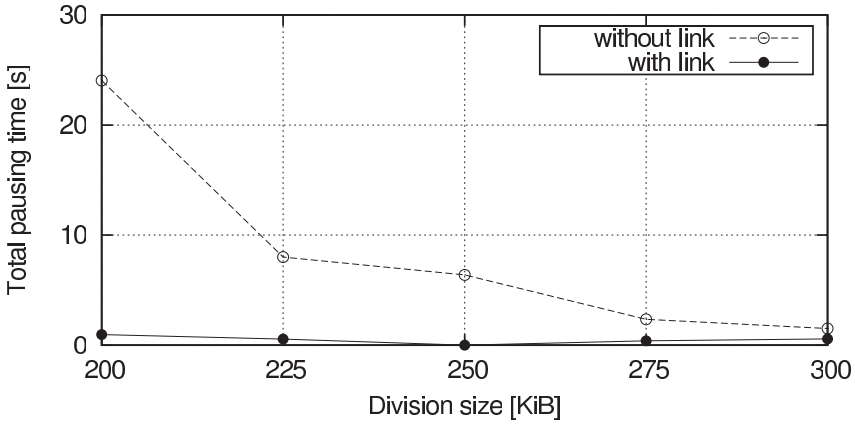


Figure 3.4: Total pausing time according to division size.

3.4.3 Total Number of Arrival Agents in Each Elapsed Time

In order to examine more details of effectiveness in our proposed method, we have measured a total number of arrival agents in each elapsed time.

This experiment has done with the same configurations of the e-Learning system, computers, and video file in Section 3.4.2; but a division size has set only to 200 KiB.

Figure 3.5 shows this experimental result. In *without link*, many agents are migrating after exceeding the 21 seconds; therefore, the file data cannot be

played smoothly. In contrary, in *with link*, many agents have arrived until 21 seconds, however, the load on the node will be concentrated. The influences about this situation are examined after this Section.

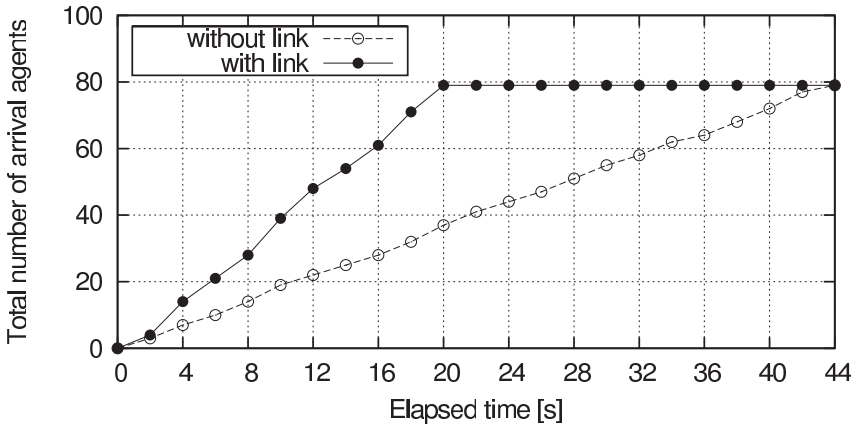


Figure 3.5: Number of arrival agents according to elapsed time.

3.4.4 Limitation of Simultaneous Agent Migrations

If division size is large, the multimedia file may not be smoothly played because some agents migrate to a requesting node simultaneously. The simultaneous agent migrations will cause extreme resource consumption. Thus, when multimedia files play, the node has to limit the number of simultaneous agent-migrations.

We have investigated in four cases. In case 1, an agent sends a requesting message to next agent immediately. In case 2, an agent sends a requesting message to next agent after it waits for 5 seconds. In case 3, an agent sends a requesting message to next agent after it waits for 10 seconds. In case 4, an agent sends a requesting message to next agent after the migration to a requesting node finishes.

In this experiment, we have investigated duration of agent migrations to a

requesting node. Each agent manages a fragment of 10 MiB that is a part of a video file of 192 MiB, and the video file has 372 seconds. All other experiment configurations are the same as Section 3.4.2.

Figure 3.6, 3.7, 3.8, and 3.9 shows the results of these experiments that is duration from the finish time of a migration of an agent to the start time of playing of a multimedia content. We can see from Figure 3.6 that the case 1 spends much time for agent migrations. For example, a 1st agent starts migration at time 0 and finishes at time 2, the 15th agent starts migration at time 14 and finishes at time 24. However, in other cases appeared in Figure 3.7 and Figure 3.8, almost agents can migrate to a requesting node in shorter time. Thus, the agent migration time decreases with increasing delay time. However, the duration of agent migrations has to less than the playing time of the multimedia file. For example, in the case 1 appears in Figure 3.9, agent migration time is shorter than other cases, but the video file played not smoothly.

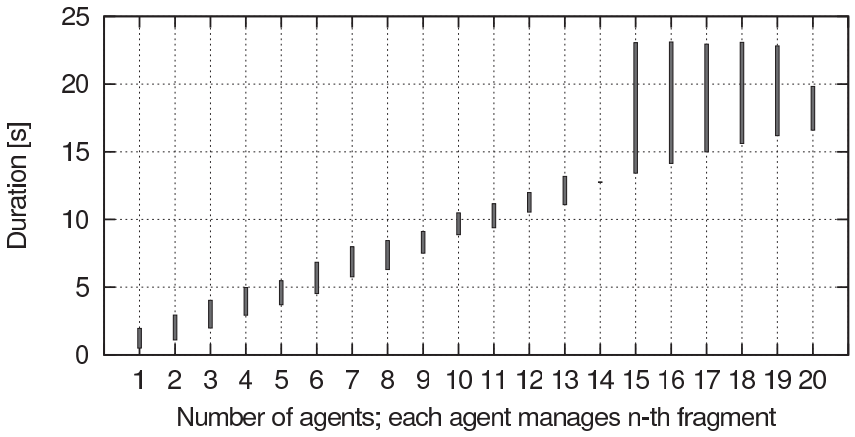


Figure 3.6: Case 1: Serial requests each has **no** delay; and parallel agent migrations occur.

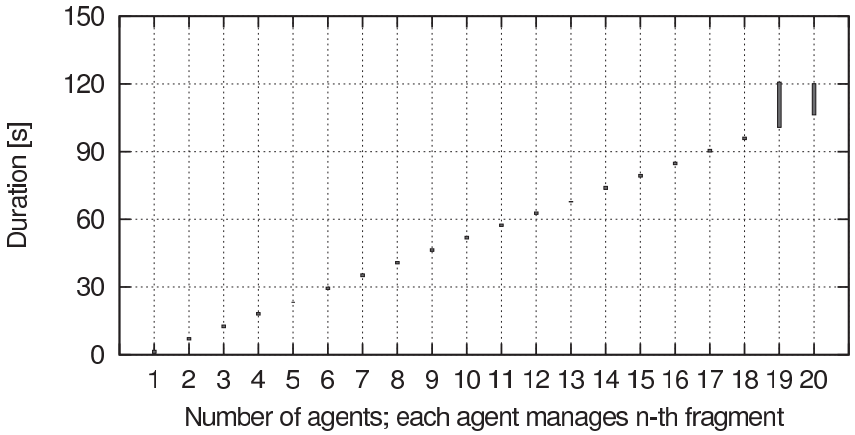


Figure 3.7: Case 2: Serial requests each has **5** seconds of delay; and parallel agent migrations occur.

3.4.5 Influence of Video Smoothness on Memory Size

We have investigated the total pausing time in the case 1 (no delay) and the case 3 (10 seconds of delay) in each maximum memory size.

In this experiment, we have adopted the maximum heap size of JVM [34] to the maximum memory size of a node, and the heap size has set a value which from 156 to 236 with step 20 MiB. All other experiment configurations are the same as Section 3.4.2.

Figure 3.10 shows the result of this experiment. In the case 1 (no delay), since many agents have to migrate simultaneously, the heap usage becomes full soon. Therefore, many agents are swapped out, and it obstructs the smooth play of multimedia data. In contrary, in the case 4 (10 seconds of delay), this does not happen. Thus, it is necessary to control the migration of agents when a many agents try to migrate simultaneously. In the case 1, the maximum heap size smaller provides total pausing time longer. However, in the case 3, by delay to get fragments, total pausing time is reduced if maximum heap size is small.

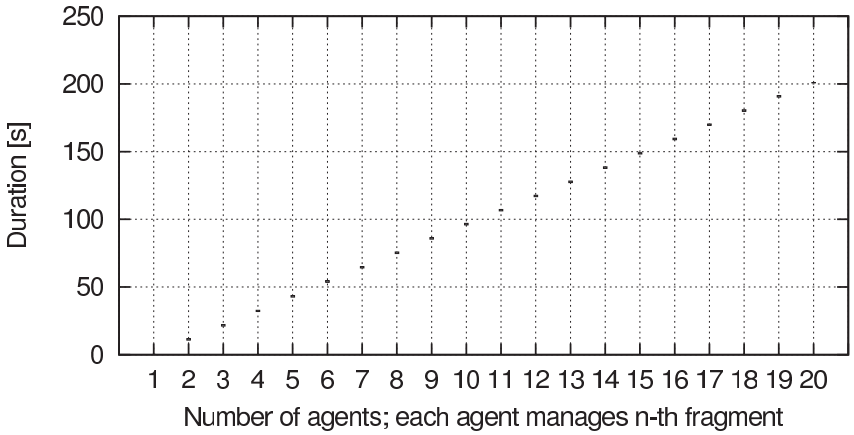


Figure 3.8: Case 3: Serial requests each has **10** seconds of delay; and parallel agent migrations occur.

3.4.6 Effectiveness of Distributed Multimedia Data Cache

We have investigated the effectiveness of distributed multimedia data cache.

This experiment has done with the same configurations of the e-Learning system, computers in Section 3.4.2; but the video file size is 5.6 MiB and its file is divided into three fragments.

Figure 3.11 shows the result of this experiment. When the number of simultaneous requests to get a fragment from distributed nodes is 1 or 2, the response time of *with cache* is longer than *without cache*. Because, in spite of the requested node has no large load, the requested node routes these requests to other nodes which have cache of fragment of video file. However, when the number is greater than or equal to 3, the response improving. Thus, greater the number of requests, the cache is more effective.

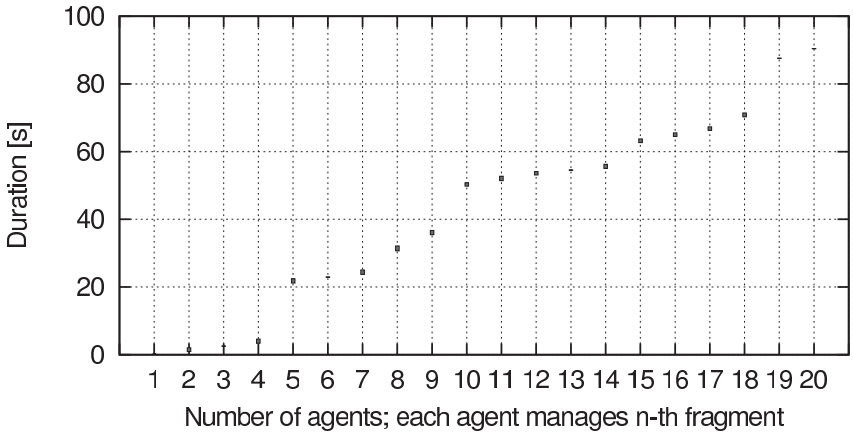


Figure 3.9: Case 4: Serial requests and agent migrations with **no** delay.

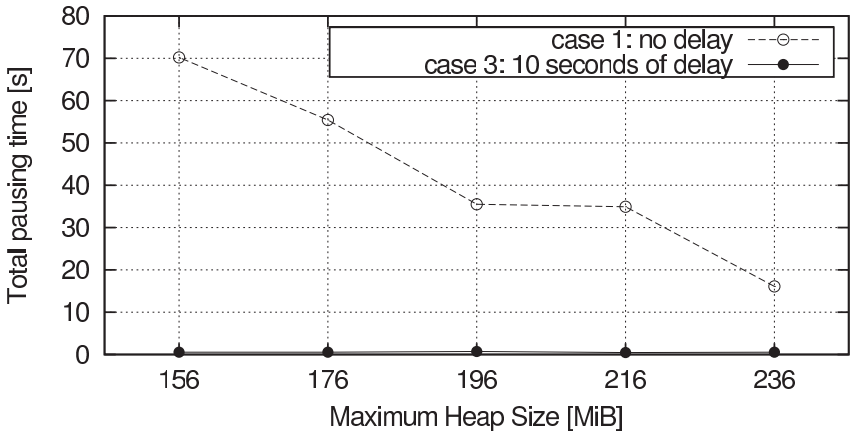


Figure 3.10: Effect of maximum heap size on total pausing time.

3.5 Related Works

Several researches have proposed P2P-based e-Learning systems. EDUTELLA [35] is a one of the earliest P2P-based e-Learning system. This is based on JXTA [36] that is a P2P application framework. There are JXTA-based e-Learning systems such as PLANT [37] and HYDRA [38]. In addition, several researches have proposed a new overlay network for a P2P-based e-Learning system, such as PROSA [39] and PeerLearning [40], to support efficient method

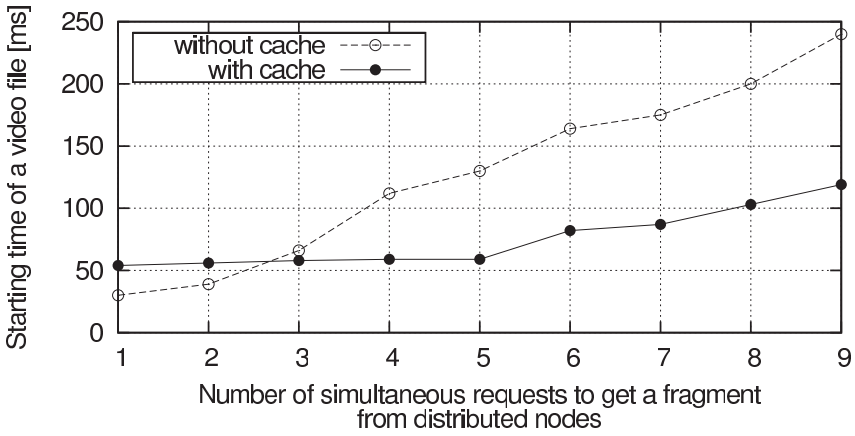


Figure 3.11: Comparison of response time between *with cache* and *without cache*.

to look up learning contents. However, these systems are different from our proposed system in that multimedia contents are divided into fragments and managed by DHT.

[41] has proposed an index caching mechanism for CAN. On the other hand, our proposed system caches not only indexes but also data of the learning contents. [42] is proposed an agent-based collaborative virtual environment architecture using grid technologies, however, this is not based on a pure P2P model. On the other hand, our proposed system is based on a pure P2P model.

In Nearcast [43], a locality-aware P2P live streaming method has proposed. However, in a live streaming, a source node will be a single point of failure. If the source node is forced outage, then the streaming will stop. On the other hand, in our proposed system, a multimedia content is preliminarily divided and is preliminarily distributed into many nodes before all nodes are notified of the multimedia content existence. Therefore, the multimedia content streaming will not be forced the outage by economic or social reasons. This characteristic is similar to pure P2P-based file sharing systems, such as Winny [44], Gnutella [45], etc., but these systems do not expect to the multi-

media streaming and cannot stream multimedia.

3.6 Conclusion

In this part, a method to play multimedia data smoothly on a pure P2P-based distributed e-Learning system is proposed. In our system, multimedia data are divided into multiple fragments by time series, each media agent manages each fragment, and their media agents are linked bidirectional. If division size of multimedia data are huge, multimedia data cannot be played smoothly because many agents have to migrate to a requesting node simultaneously. Therefore, we devised the timing of sending a requesting message to next agent. This method enables the smooth play of multimedia data on P2P-based distributed e-Learning system.

Chapter 4

Debugging Mobile Agent Systems

A mobile agent is an autonomous software module that can migrate between different computers. A mobile agent is designed and implemented like a human, and mobile agents work together by interactions among them like a human community. Thus, a mobile agent technology is helpful when we develop distributed systems with an easy-to-understand design and implementations for humans. Many researchers have proposed various applications through mobile agent technologies. However, mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. This chapter discusses problems of mobile agents for debugging, and proposes a remote debugger in order to solve these problems. Our proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system.

4.1 Introduction

A mobile agent is an autonomous software module that can work on different computers and migrate between these computers. These computers install an agent runtime environment (ARE) which is connected by a network, and a mobile agent can work on these AREs. A mobile agent is designed and implemented like a human, and these mobile agents work together by interactions among them like a human community. Thus, mobile agent technologies are helpful when we develop distributed systems with easy-to-understand design and implementations for humans.

Many researchers have proposed various design, implementations, applications, and platforms of mobile agent systems [46, 47]. However, mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. Since many mobile agents work concurrently and migrate to many computers in a mobile agent system, when a problem occurs in a mobile agent, a programmer must find this mobile agent with difficulty from many mobile agents and many AREs. Since a mobile agent migrates from a computer to another computer, in order to debug this mobile agent, it requires connections to these computers and functions of remotely debugging. Since a mobile agent migrates even when a programmer is debugging this mobile agent, the remote debugger must change a destination of a connection to a computer which the agent migrates to. Additionally, since behavior of mobile agent is affected from interactions with other mobile agents, even if a mobile agent executes a same program of a task, then a result of it is not always same. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced.

A debugger for ordinary software other than a mobile agent system does not support functions for characteristics described above of a mobile agent system, and it is hard to debug mobile agent systems. Therefore, this chapter discusses problems on debugging of mobile agent systems, and proposes a remote debugger for mobile agent systems in order to solve these problems. Proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system.

The rest of this chapter is organized as follows. Section 4.2 points out related works. Section 4.3 discusses problems and approaches in debugging of mobile agent systems. Section 4.4 describes the design, and Section 4.5 shows a implementation of our approaches. Section 4.6 shows the evaluations and their results. Finally, Section 4.7 draws the conclusions.

4.2 Related Works

In the research area of multi-agent systems, there have been proposed several technologies for developers. IOM/T [48], *Q* [49], and SDLMAS [50] is a description language to define interactions and scenarios among agents for multi-agent systems. However, the definitions supported by these languages are based on only remote messaging among agents who basically stay on a computer and do not basically migrate. Thus, it is difficult to apply interactions proposed on the area of multi-agent systems to interactions including mobility.

Also, in the research area of mobile agent systems, there have been proposed approaches like mentioned above. Mobile Object-Z (MobiOZ) [51, 52] can specify mobile agent applications with a specification notation extended the Z formal specification notation [53] and verify the model of the applications

by the SPIN (Simple Process meta language INterpreter) model checker [54]. LAM (Logical Agent Mobility) [55, 56] also can verify models of applications by the SPIN. π-ADL [57] is an architecture description language based on the higher-order typed π-calculus, which is one of a model for a process calculus, for specifying dynamic and mobile software architectures, and models described by π-ADL can be verified by CADP (Construction and Analysis of Distributed Processes) toolbox [58], and it can be applied to mobile agent systems.

If definitions with these languages (notations) are valid logically, generating native source codes as implementations from these definitions, bugs by implementing do not occur. However, in order to apply these definitions to existing applications, developers must design and implement additional definitions and generators for applications. Therefore, these approaches make the applications robust, but choosing these approaches becomes difficult in practice, because the costs increase with increasing a scale and complexity of the applications. Additionally, there are applications that cannot be applied to approaches mentioned above. Therefore, a debugger for mobile agents is required.

Several researchers have proposed a tool to visualize and debug interactions among agents [59–63], but these tools do not focus on mobility of agents. JADE [64] and Agent Factrory [65, 66] is a platform for multi-agent systems that includes debuggers. These platforms support mobility of agents, but their debuggers do not support and focus on mobility of agents, because these platforms mainly focus on multi-agent systems and subsequently introduced mobility. Also [67] suggested requirements for IDEs (Integrated Development Environments) to support construction of multi-agent systems, but they touched only on mobility just a little. MiLog [68] is a framework (platform) including an IDE for mobile agent systems that can visualize locations of agents in a network and dump logs of each agent. However, because its features are sim-

ple, in massive mobile agent systems, debugging processes with these features is laborious.

Therefore, this chapter discusses problems of mobile agents for debugging, and proposes an effective remote debugger in order to solve these problems.

4.3 Problems and Approaches on Debugging Mobile Agent Systems

Mobile agent systems have characteristics that multiple mobile agents interact with other agents, work on multiple nodes, and migrate to these nodes. However, these characteristics make it difficult to debug the system for the following reasons:

1. *Multiple targets to debug* A debugger for general software targets one application (called a process in OS). However, in a mobile agent system, many multiple mobile agents work concurrently and migrate concurrently to many nodes. Thus, at first, a debugger for mobile agent systems must search targets from a network with search queries including various condition of mobile agents.

2. *Working on remote nodes:* In mobile agent systems, a mobile agent works on distant nodes. Thus, in order to check behavior of an agent, a debugger must connect to a node on which this agent is working. A remote debugger is required in order to debug mobile agents.

3. *Migrating on debugging:* A mobile agent migrates among nodes even when a programmer is debugging them. However, a debugger for general software or multi-agent systems does not consider frequent changing

a destination according as migrations of agents. Thus, in order to debug this mobile agent, the remote debugger must change a destination of a connection to a node to which this agent migrates. Additionally, the debugger must continuously support features for debugging such as single-stepping, breaking, and tracking the values of variables, even when the agent migrates.

4. *Side effects by interactions other agents and nodes:* Behavior of a mobile agent is affected from interactions with other mobile agents and AREs. Even if a mobile agent executes a same program of a task, then a result of it is not always same by these side effects. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced.

4.4 Design

This section considers a design of a debugger for mobile agent systems on the basis of approaches described above.

4.4.1 Searching Function

Parameters of Search Query

A programmer cannot grasp all mobile agents in the system; because there are multiple targets to debug in mobile agent systems and the number of mobile agents and nodes is large. Therefore, a searching function is required which finds agents from a network of mobile agent systems by various parameters. In order to search mobile agents, a name, a state, and a location of a mobile agent is a useful information. A name gives a classification to mobile agents such

as a class name of instances, a role, etc. A state represents a runtime state of a mobile agents like a state of a thread in Java, such as *running*, *waiting*, and *terminated*. A location is an identifier of node on which a mobile agent is working currently. By using these parameters as a query for searching, a programmer can filter mobile agents which have suspicious behavior from many mobile agents in a system.

Identifiers for Mobile Agents

Because some platforms (frameworks) of mobile agents have no identifiers to give uniqueness for mobile agents, a debugger for mobile agent systems must give a new identifier to searched mobile agents. Thus, a debugger gives UUID (Universally Unique Identifier) [69] to mobile agents as unique identifiers. Also, these identifiers are also used by other functions described below.

Method to Search

Since mobile agents are used on various network topologies such as a P2P (peer-to-peer) and a client-server, a suitable method to search depends on a network topology used by an application. Thus, our approach does not limit the kind of method to search mobile agents. In our implementation, we use a query flooding simply because this our proposition does not focus on a performance of searching.

Figure 4.1 shows a overview of the search function. A developer sends queries, which contains a name, a location, and a state of agent to search this agent from computers in a network by using flooding scheme. Thus, agents matched by the query are detected from the network.

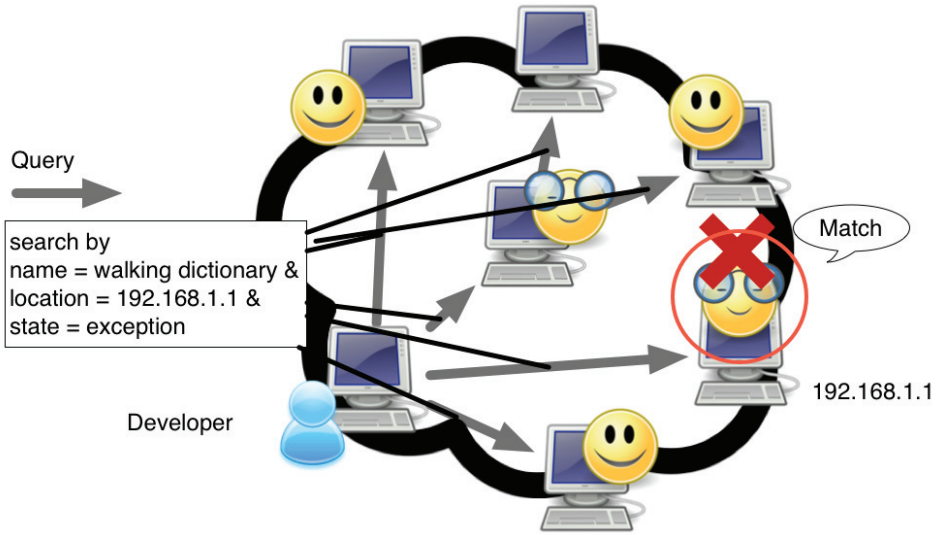


Figure 4.1: Search Function.

4.4.2 Monitoring Function

Because it is not always find mobile agents which have suspicious behavior when searching, the notifications when mobile agents cause problems are required. Therefore, if a parameter of a search query matches a mobile agent when the mobile agent causes problems, the node on which this agent stay sends a notification to a node of developers. This function is constructed with a two-step process. Figure 4.2 shows the step 1 that the process spreads queries to computers in the network. The queries contains a name and a state of a agent and stays on each computer.

Next, Figure 4.3 shows the step 2 that notification messages are send from computers which agents change a state to the exception. The notifications contain a name of an agent, a location of an agent, a id of an agent, a name of an event, a name of a previous state, and a name of a next state. Therefore, the developer can be notified changing of states of agents with realtime, thus, the developer can monitor these agents.

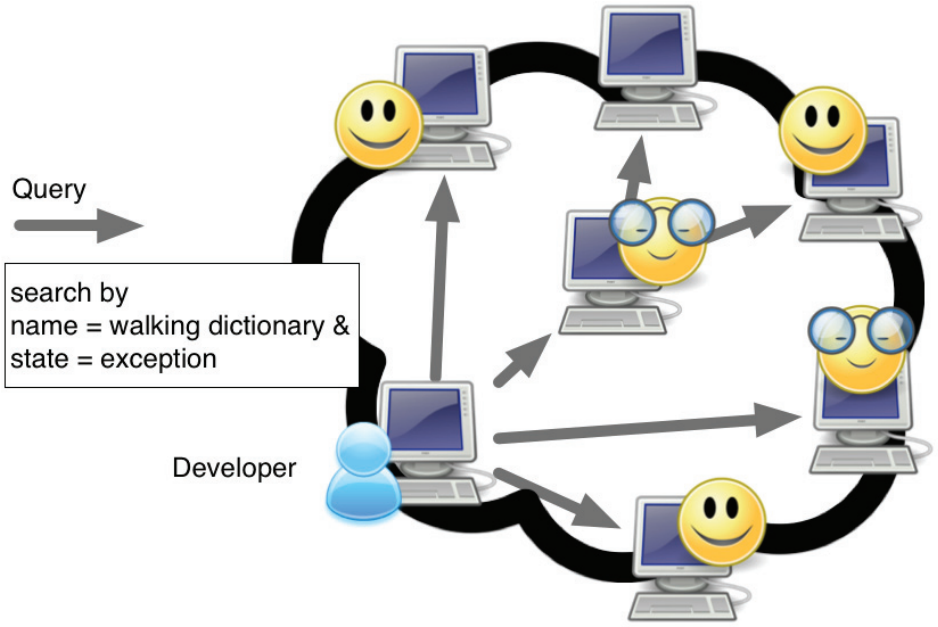


Figure 4.2: Monitoring Function: Step 1.

4.4.3 Debugging Function

Breaking of Mobile Agent

In order to debug a mobile agent found by the search function or a monitoring function, a developer sets the mobile agent to a debug mode.

Figure 4.4 shows a overview of the breaking function. Agents flagged as a debug mode are sent source codes. The source codes are set a break point by a developer. Now, the source codes with break points are transferred to the all of agents which has these same source codes. Because there is a case that multiple instances of mobile agents are generated from one class or cloned, it is possible that multiple mobile agents which are generated from same erroneous source codes cause problems. Thus, breakpoints must be specified in source codes, and these mobile agents must have source codes on runtime.

Figure 4.5 shows a ovewview of notifications. When an agent tries to

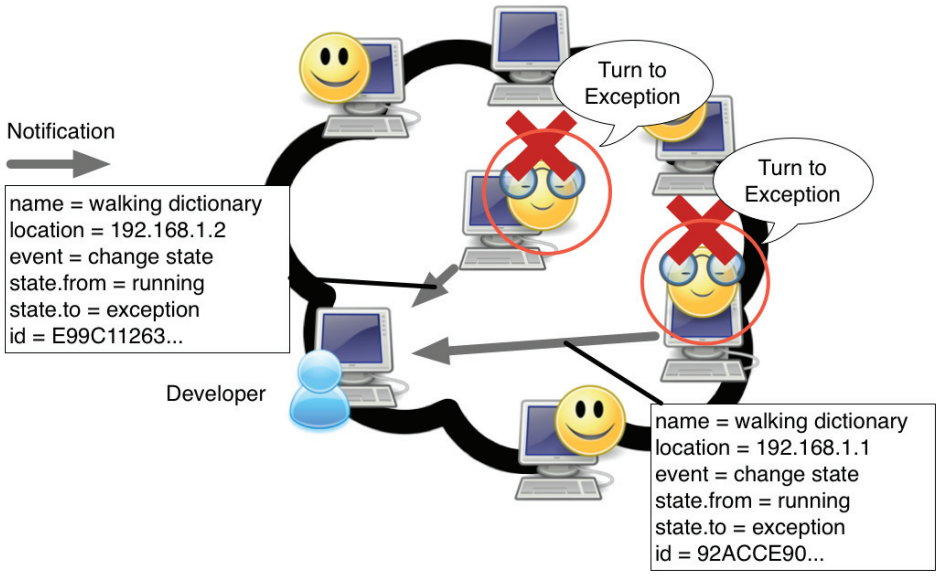


Figure 4.3: Monitoring Function: Step 2.

execute a program code at a break point on a source code, a notification is sent to a node of a developer. The notification contains a name of an agent, a location of an agent, a name of an event, and a id of an agent. Thus, a developer can handle an agent which has break points.

Single-Stepping of Mobile Agent

Basically, a function of single-stepping can be realized by applying breaking but there are problems when a migration occurs between steps. Generally, nodes are connected by different sessions of TCP (transmission control protocol). Since TCP maintains order of packets in a session but TCP does not maintain order between different sessions, a remote debugger must maintain order by other method. Thus, a mobile agent in a debug mode has a counter of the number of migrations. When the mobile agent migrates, this agent sends a notification of a migration with the counter. Additionally, mobile agent

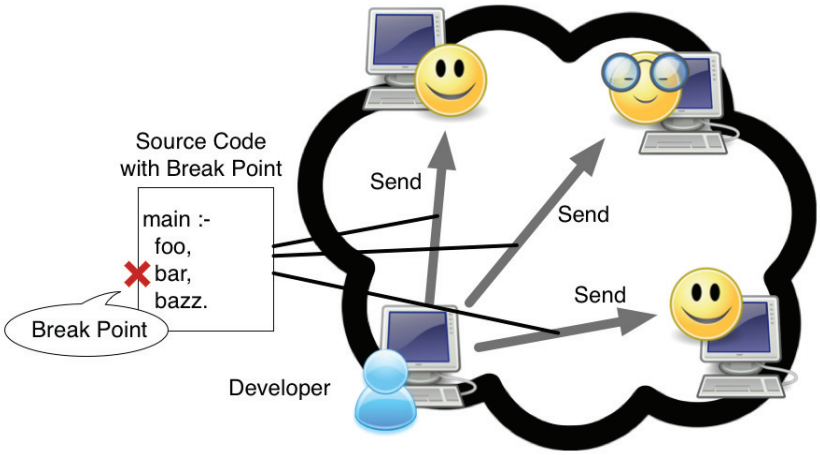


Figure 4.4: Breaking of Mobile Agent.

migrates with an own runtime state such as a function call stack, a program counter, and information of breakpoint. Thus, a programmer can continuously execute single-stepping of a mobile agent with migrations.

Also, this function is constructed with a two-step process. Figure 4.6 shows the step 1 which the agent sends a notification message when who migrates. The notification message contains a source location and a destination location. Next, Figure 4.7 shows the step 2 that the developer can change a destination of a connection to the agent by using notification messages.

4.4.4 Logging Function

In order to do debug, bugs must be reproduced. In the case of debugging of an agent, an environments of not only agents but also AREs must be reproduced to reproduce bugs. Because, an agent writes and reads data to various other agents and various AREs and an agent spreads data into a system. Thus, in order to support reproducing of bugs, each ARE must save data of interactions between mobile agents and nodes. Figure shows a overview of the logging

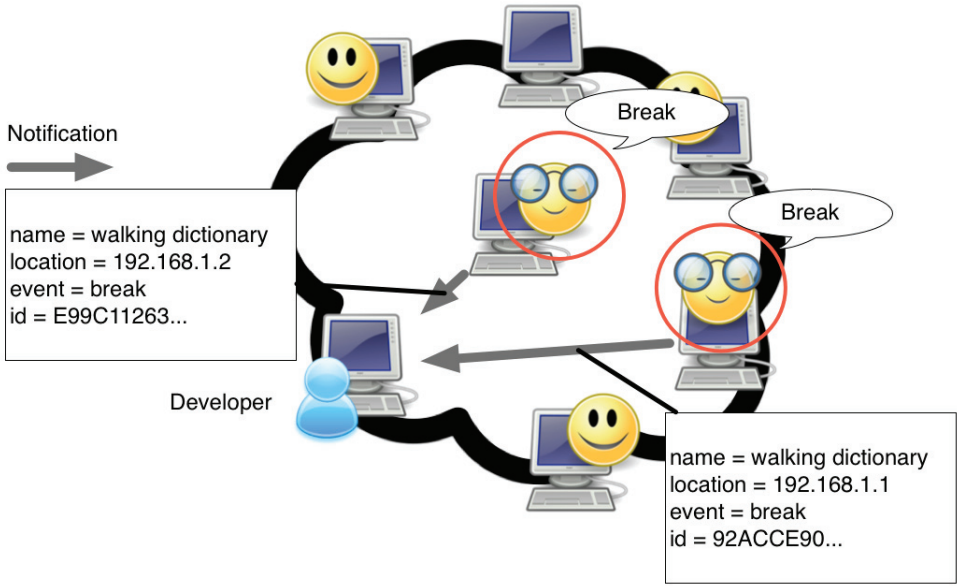


Figure 4.5: Notification of Breaking.

function. An agent works as a debug mode who stores logs which contains locations, the type of I/O, and contents of I/O into own. Therefore, a developer can trace logs of input and output data linked with locations.

4.5 Implementation

In order to evaluate our debugger for mobile agent system, we have implemented above design.

Our debugger is implemented to the mobile agent framework called Maglog (Mobile AGent system based on proLOG) [70]. The agent of Maglog is implemented by extending PrologCafé [22]. PrologCafé is a 100% pure Java implementation of the Prolog programing language, which contains a Prolog-to-Java source-to-source translator and a Prolog interpreter.

Therefore, Java programmers are able to completely access to a Prolog

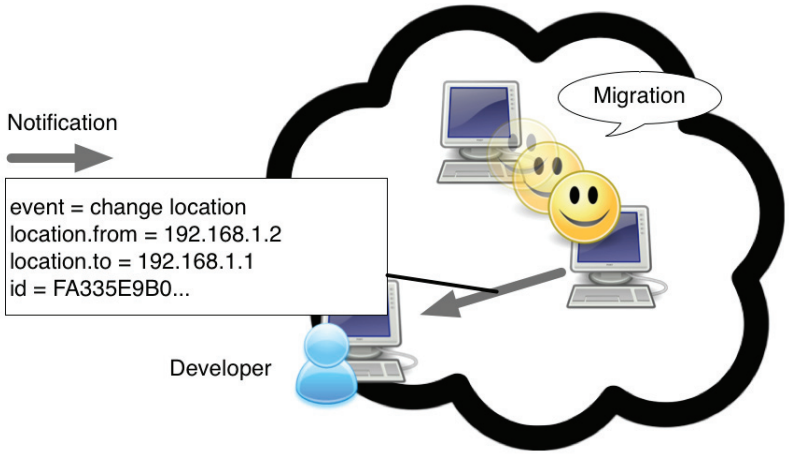


Figure 4.6: Single-Stepping of Mobile Agent: Step 1.

interpreter's internal execution states such as choice point stack, trail stack, a set of variable bindings, and etc. When an agent migration, agent execution state, application data, program codes, and identifiers are converted to byte array by Java Object Serialization [14], and transferred among AREs through HTTP/1.1 protocol. Thus, the information of breakpoint is realized as an instance of a prolog interpreter.

FIgure 4.9 shows a user interface for an agent. This user interface can be created with each instance of an agent.

Agent ID: A unique identifier for a instance of an agent.

Current Location: A current location of an agent when this agent stops at a break point.

Code Name: A name of a source code to do debug.

Break Point: A predicate specified as a break point is colored to the red.

Call Stack: A call stacks of predicates of an agent.

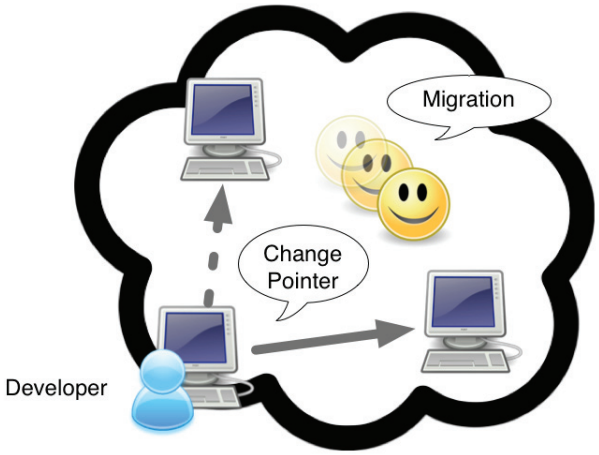


Figure 4.7: Single-Stepping of Mobile Agent: Step 2.

Variables as Break Point: Contents of variables of a predicate specified as a break point.

Internal DB: Data stored in an agent.

Output: A standard output of an agent.

Single-Stepping Controller: This button can execute step-in, step-over, clear, and resume.

4.6 Experiment

In order to evaluate our debugger for mobile agent system, we have experimented to debug a test application which includes a bug.

Our debugger is implemented to the mobile agent framework called Maglog (Mobile AGent system based on proLOG) [70]. The agent of Maglog is implemented by extending PrologCafé [22]. PrologCafé is a 100% pure Java

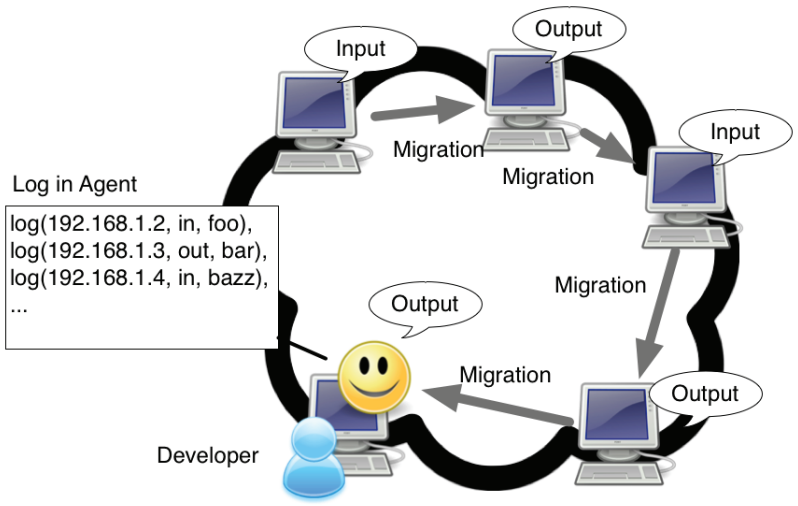


Figure 4.8: Logging Function.

implementation of the Prolog programing language, which contains a Prolog-to-Java source-to-source translator and a Prolog interpreter.

Therefore, Java programmers are able to completely access to a Prolog interpreter's internal execution states such as choice point stack, trail stack, a set of variable bindings, and etc. When an agent migration, agent execution state, application data, program codes, and identifiers are converted to byte array by Java Object Serialization [14], and transferred among AREs through HTTP/1.1 protocol. Thus, the information of breakpoint is realized as an instance of a prolog interpreter.

In this application, there is an agent that migrates to nodes at random sown as Figure 4.10. The application is implemented by a third party, and a programmer has tried to debug the application without knowing the bug.

In the experiment, the agent has stopped with an exception on somewhere in a network. The programmer has found the location of the agent by using search function and tried to clear cause of this by using single-stepping function shown as Figure 4.11. As a result, the programmer could find out cause that

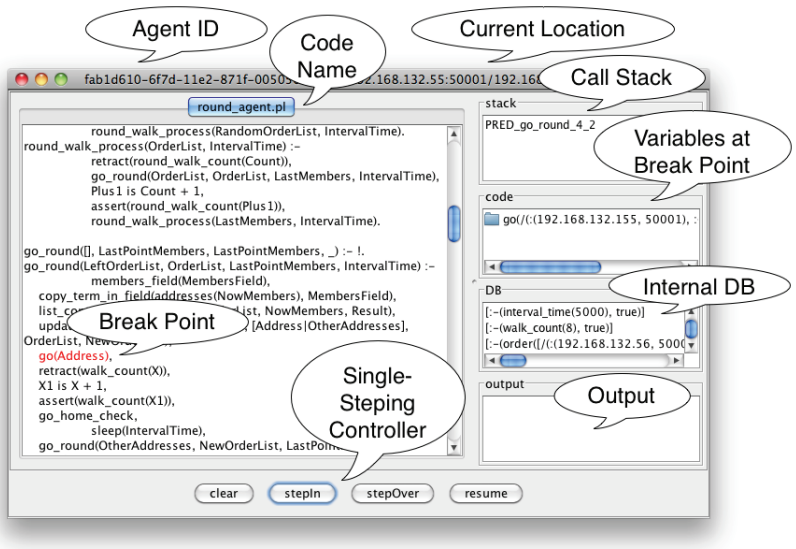


Figure 4.9: User Interface.

the agent mistakes the increment for the decrement of a list contains random destination locations.

4.7 Conclusion

This chapter has discussed problems of mobile agents for debugging, and proposed a remote debugger in order to solve these problems. Our proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system. The experiment shows that our debugger for mobile agent systems finds effectively bugs.

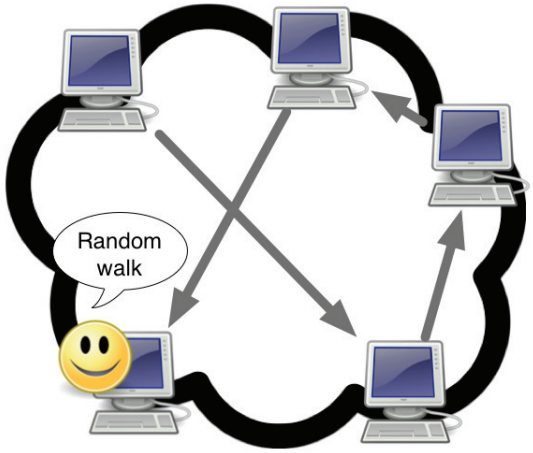


Figure 4.10: Random Walk.

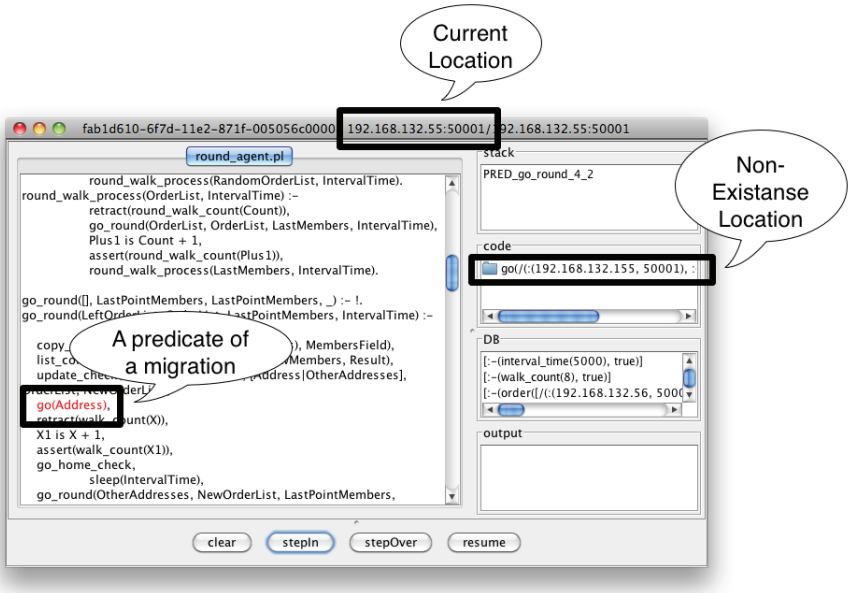


Figure 4.11: Detection of a Bug.

Chapter 5

Conclusion

In Chapter 2, we discussed to apply a cache mechanism to mobile agent migration. Our cache mechanism does not influence on the implementation of mobile agent applications because it works on an agent runtime environment. We implemented the cache mechanism on Maglog, which is a mobile agent framework, and conduct experiments. The result on a meeting scheduling system shows that the mechanism of the cache enables to improve its performance by 52%. Additionally, when multiple agents migrate from multiple sources to one destination at the same time, the proposed mechanism prevents duplicate transfers of program codes by using cancellations of requests of transfers of program codes. We implemented our mechanism on Maglog and conducted experimental results on a load distribution system. In this experiment at scale-out that increases the number of nodes from 1 to 16, our mechanism improved the performance of the system by up to 54.4%.

In Chapter 3, a method to play multimedia data smoothly on a pure P2P-based distributed e-Learning system is proposed. In our system, multimedia data are divided into multiple fragments by time series, each media agent manages each fragment, and their media agents are linked bidirectional. If

division size of multimedia data are huge, multimedia data cannot be played smoothly because many agents have to migrate to a requesting node simultaneously. Therefore, we devised the timing of sending a requesting message to next agent. This method enables the smooth play of multimedia data on P2P-based distributed e-Learning system.

Chapter 4 has discussed problems of mobile agents for debugging, and proposed a remote debugger in order to solve these problems. Our proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system. The experiment shows that our debugger for mobile agent systems finds effectively bugs.

References

[1] YonSik Lee and KwangJong Kim. Optimal migration path searching using path adjustment and reassignment for mobile agent. In *Proceedings of the 4th International Conference on Networked Computing and Advanced Information Management*, volume 2, pages 564–569, 2008.

[2] Guilherme Soares and Luís Moura Silva. Optimizing the migration of mobile agents. In *Proceedings of the 1st International Workshop on Mobile Agents for Telecommunication Applications*, pages 161–178, 1999.

[3] Damianos Gavalas. An experimental approach for optimising mobile agent migrations. *Mediterranean Journal of Computers and Networks*, 1(1):47–56, 2005.

[4] Object Management Group, Inc. *Mobile Agent System Interoperability Facilities Specification*, 1997.

[5] I. Satoh. Agentspace : a higher order mobile agent system. *The Special Interest Group Notes on Programming of Information Processing Society of Japan*, 98(30):41–48, 1998.

[6] R. Kowalczyk P. Braun, I. Muller and S. Kern. Increasing the migration efficiency of java-based mobile agents. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 508–511, 2005.

[7] Mobile code toolkit v1.6.2. `http://www.sce.carleton.ca/netmanage/mctoolkit/mctoolkit162/mct.html`, 2013.

[8] Alfonso Fuggetta, Gian Pietro Picco, and Giovanni Vigna. Understanding code mobility. *IEEE Trans. on Software Engineering*, 24:342–361, 1998.

[9] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification (SC00023K)*, 2004.

[10] Foundation for Intelligent Physical Agents. *FIPA Abstract Architecture Specification (SC00001L)*, 2002.

[11] Shinichi Motomura, Takao Kawamura, and Kazunori Sugahara. Logic-based mobile agent framework with a concept of "field". *IPSJ Journal*, 47(4):1230–1238, 2006.

[12] Aglets. `http://aglets.sourceforge.net/`, 2013.

[13] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). Request for Comments 3174, 2001.

[14] Oracle and/or its affiliates. Java Object Serialization Specification. Available at `http://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html`, 2013.

[15] Oracle and/or its affiliates. MessageDigest (Java Platform SE 6). Available at `http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html`, 2013.

[16] Oracle and/or its affiliates. HashMap (Java Platform SE 6). Available at `http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html`, 2013.

[17] Oracle and/or its affiliates. Java SE 6 Documentation. Available at `http://docs.oracle.com/javase/6/docs/`, 2013.

[18] Luigi Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27:31–41, 1997.

[19] Takao Kawamura, Shinichi Motomura, Kengo Kagemoto, and Kazunori Sugahara. Meeting arrangement system based on mobile agent technology. In *Proceedings of the 2nd International Conference on Web Information Systems and Technologies*, pages 117–120, 4 2006.

[20] Takao Kawamura, Yusuke Hamada, Kazunori Sugahara, Kengo Kagemoto, and Shinichi Motomura. Multi-agent-based approach for meeting scheduling system. In *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 79–84, 2007.

[21] Yariv Aridor and Danny B. Lange. Agent design patterns: Elements of agent application design. In *Proceedings of the 2nd International Conference on Autonomous Agents*, pages 108–115, 1998.

[22] Mutsunori Banbara, Naoyuki Tamura, and Katsumi Inoue. Prolog cafe: A prolog to java translator system. In *Proceedings of the 16th International Conference on Applications of Declarative Programming and Knowledge Management*, pages 1–11, 2005.

[23] Noto M., Numazawa M., and Kurihara M. Empirical evaluation of traffic performance of inter-agent communication systems with mobile agents. *IEEJ Transactions on Electronics, Information and Systems*, 124(3):904–911, 2004.

[24] Takahashi T. and Mizuta H. Efficient agent-based simulation framework for multi-node supercomputers. In *Proceedings of the 38th conference on Winter simulation*, pages 919–925, 2006.

[25] Miyata N. and Ishida T. Placing agents in massively multi-agent systems. *The IEICE transactions on information and systems*, J90-D(4):1023–1030, 2007.

[26] Teck-How Chia and Srikanth Kannapan. Strategically mobile agents. In *Proceedings of the 1st International Workshop on Mobile Agents*, pages 149–161, 1997.

[27] Peter Braun and Wilhelm Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers Inc., 2005.

[28] M. Driscoll. *Web-Based Training: Creating e-Learning Experiences*. John Wiley & Sons, New York, USA, second edition, 2002.

[29] V. Uskov, editor. *Web-based Education*. ACTA Press, 2010.

[30] Apple Inc. iTunes Store. available at `http://www.apple.com/itunes/`, 2013.

[31] Netflix, Inc. Netflix. available at `http://www.netflix.com/`, 2013.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[33] S. Motomura, T. Kawamura, and K. Sugahara. Logic-based mobile agent framework with a concept of "field". *IPSJ Journal*, 47(4):1230–1238, 2006.

[34] Oracle and/or its affiliates. java - the Java application launcher. available at `http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/java.html`, 2013.

[35] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a p2p networking infrastructure based on RDF. In *Proceedings of the 11th International Conference on World Wide Web*, pages 604–615, 2002.

[36] J. Verstrynge. *Practical JXTA II*. Lulu Enterprises Inc., 2010.

[37] M. Li, H. Zhu, and Y. Zhu. Plant: a distributed architecture for personalized e-learning. In *Proceedings of the 6th International Conference on Advances in Web Based Learning*, pages 20–30, 2007.

[38] I. A. Zualkernan. Hydra: A light-weight, scorm-based p2p e-learning architecture. In *Proceedings of the 5th IEEE International Conference on Advanced Learning Technologies*, pages 484–486, 2005.

[39] V. Carchiolo, A. Longheu, G. Mangioni, and V. Nicosia. Adaptive e-learning: An architecture based on PROSA p2p network. In *Proceedings of the 21st International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems: New Frontiers in Applied Artificial Intelligence*, pages 777–786, 2008.

[40] G. Wang, Y. Yuan, Y. Sun, J. Xin, and Y. Zhang. Peerlearning: A content-based e-learning material sharing system based on p2p network. *World Wide Web*, 13(3):275–305, 2010.

[41] S. Tagashira, S. Shirakawa, and S. Fujita. Proxy-based index caching for content-addressable networks. *IEICE Transactions on Information and Systems*, E89-D(2):555–562, 2006.

[42] C. Huang, F. Xu, X. Xu, and X. Zheng. Towards an agent-based robust collaborative virtual environment for e-learning in the service grid. In

Proceedings of the 9th Pacific Rim International Conference on Agent Computing and Multi-Agent Systems, pages 702–707, 2006.

[43] X. Tu, H. Jin, X. Liao, and J. Cao. Nearcast: A locality-aware p2p live streaming approach for distance education. *ACM Transactions on Internet Technology (TOIT)*, 8(2):7:1–7:23, 2008.

[44] Isamu Kaneko. *The Technology of Winny*. ASCII MEDIA WORKS Inc., 2005.

[45] Patrick Kirk. Gnutella - a protocol for a revolution. available at `http://rfc-gnutella.sourceforge.net`, 2013.

[46] Ali R. Hurson, Evens Jean, Machigar Ongtang, Xing Gao, Yu Jiao, and Thomas E. Potok. Recent advances in mobile agent-oriented applications. In *Mobile Intelligence: Mobile Computing and Computational Intelligence*, pages 106–139, 2010.

[47] Abdelkader Outtagarts. Mobile agent-based applications : a survey. *International Journal of Computer Science and Network Security*, 9(11):331–339, 2009.

[48] Doi Takuo, Tahara Yasuyuki, and Honiden Shinichi. IOM/T: An interaction description language for multi-agent systems. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 778–785, 2005.

[49] Ishida Toru. Q: A scenario description language for interactive agents. *IEEE Computer*, 35(11):42–47, 2002.

[50] Igor Čavrak, Armin Stranjak, and Mario Žagar. SDLMAS: A scenario modeling framework for multi-agent systems. *Journal of Universal Computer Science*, 15(4):898–925, 2009.

[51] Kenji Taguchi and Jin Song Dong. An overview of Mobile Object-Z. In *Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering (ICFEM 2002)*, pages 144–155, 2002.

[52] Kenji Taguchi and Jin Song Dong. Formally specifying and verifying mobile agents — model checking mobility: the MobiOZ approach. *International Journal of Agent-Oriented Software Engineering*, 2(4):449–474, 2008.

[53] ISO/IEC. *Information technology — Z formal specification notation —Syntax, type system and semantics*, 1 edition, 2002. ISO/IEC 13568:2002(E).

[54] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer London, 2008.

[55] Dianxiang Xu, Jianwen Yin, Yi Deng, and Junhua Ding. A formal architectural model for Logical Agent Mobility. *IEEE Transactions on Software Engineering*, 29(1):31–45, 2003.

[56] Junhua Ding, Dianxiang Xu, Xudong He, and Yi Deng. Modeling and analyzing a mobile agent-based clinical information system. *The International Journal of Intelligent Control and Systems*, 10(2):143–151, 2005.

[57] Flavio Oquendo. π-ADL: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile soft-

ware architectures. *ACM Software Engineering Notes (SEN)*, 29(3):1–14, 2004.

[58] Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2010: a toolbox for the construction and analysis of distributed processes. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS '11/ETAPS '11, pages 372–387, Berlin, Heidelberg, 2011. Springer-Verlag.

[59] Divine T. Ndumu, Hyacinth S. Nwana, Lyndon C. Lee, and Jaron C. Collis. Visualising and debugging distributed multi-agent systems. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, AGENTS '99, pages 326–333, New York, NY, USA, 1999. ACM.

[60] Dung N. Lam and K. Suzanne Barber. Debugging agent behavior in an implemented agent system. In *Proceedings of the 2nd International Conference on Programming Multi-Agent Systems*, ProMAS '04, pages 104–125, Berlin, Heidelberg, 2005. Springer-Verlag.

[61] Lin Padgham, Michael Winikoff, and David Poutakidis. Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173–190, 2005.

[62] Guillermo Vigueras and Juan A. Botia. Tracking causality by visualization of multi-agent interactions using causality graphs. In *Proceedings of the 5th International Conference on Programming Multi-Agent Systems*, ProMAS '07, pages 190–204, Berlin, Heidelberg, 2008. Springer-Verlag.

[63] Lawrence Cabac, Till Dörges, Michael Duvigneau, and Daniel Moldt. Requirements and tools for the debugging of multi-agent systems. In *Proceed-*

ings of the 7th German Conference on Multiagent System Technologies, MATES '09, pages 238–247, Berlin, Heidelberg, 2009. Springer-Verlag.

[64] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. *JADE PROGRAMMER'S GUIDE*. Telecom Italia S.p.A., 2010.

[65] Rem Collier. Debugging agents in Agent Factory. In *Proceedings of the 4th International Conference on Programming Multi-Agent Systems*, ProMAS '06, pages 229–248, Berlin, Heidelberg, 2007. Springer-Verlag.

[66] F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, SAC '02, pages 101–106, New York, NY, USA, 2002. ACM.

[67] Simon Lynch and Keerthi Rajendran. Breaking into industry: Tool support for multiagent systems. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 136:1–136:3, New York, NY, USA, 2007. ACM.

[68] N. Fukuta, T. Ito, and T. Shintani. Milog: A mobile agent framework for implementing intelligent information agents with logic programming. In *Pacific Rim International Workshop on Intelligent Information Agents*, 2000.

[69] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. Request for Comments (RFC) 4122, 2005.

[70] Takao Kawamura, Shinichi Motomura, and Kazunori Sugahara. Implementation of a logic-based multi agent framework on java environment. In *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 486–491, 2005.

List of Publications

Journals

[1] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara. Effective Mobile Agent Migration based on Code Caching. *The Transactions of the Institute of Electronics, Information and Communication Engineers D*, J96-D(7):1576-1584, 2013 (in Japanese).

[2] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara. Evaluation of Program Code Caching for Mobile Agent Migrations. *Computer Technology and Application*, 4(7):356-363, 2013.

[3] **Masayuki Higashino**, Tadafumi Hayakawa, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara. Management of Streaming Multimedia Content using Mobile Agent Technology on Pure P2P-based Distributed e-Learning System. *International Journal of Grid and Utility Computing*, 2013 (in press).

International Conferences and Workshops

[1] **Masayuki Higashino**, Shin Osaki, Shinya Otagaki, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Debugging Mobile Agent Systems. In *Proceedings of the 15th International Conference on Information Integration and Web-based Applications and Services (iiWAS 2013)*, pages 667-669, Vienna, Austria, 2013.

[2] **Masayuki Higashino**, Tadafumi Hayakawa, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Management of Streaming Multimedia Content using Mobile Agent Technology on Pure P2P-based Distributed e-Learning System. In *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA 2013)*, pages 1041-1047, Barcelona, Spain, 2013.

[3] **Masayuki Higashino**, Tadafumi Hayakawa, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Multimedia Streaming with Caching on Pure P2P-based Distributed e-Learning System using Mobile Agent Technologies. In *Proceedings of the 2nd International Conference on Electrical Engineering and Computer Sciences (ICEECS 2013)*, pages 661-669, Tokyo, Japan, 2013.

[4] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Effective Mobile Agent Migration Mechanism on Load Distribution System. In *Proceedings of the 2nd International Conference on Electrical Engineering and Computer Sciences (ICEECS 2013)*, pages 653-660, Tokyo, Japan, 2013.

[5] Tadafumi Hayakawa, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Management of Multimedia Data

for Streaming on a Distributed e-Learning System. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2012)*, pages 1282-1285, Fukuoka, Japan, 2012.

[6] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Mobile Agent Migration Based on Code Caching. In *Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2012)*, pages 651-656, Fukuoka, Japan, 2012.

[7] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Data Traffic Reduction for Mobile Agent Migration. In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence (ICAART 2012)*, volume 2, pages 351-354, Vilamoura, Portugal, 2012.

[8] **Masayuki Higashino**, Toshihiko Sasama, Takao Kawamura, and Kazunori Sugahara: A Method of Transparent Swapping Control for Mobile Agents. In *Proceedings of SICE Annual Conference 2010 (SICE 2010)*, pages 575-576, Taipei, Taiwan, 2010.

Domestic Conferences and Workshops

[1] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Reduction of Traffic for Mobile Agent Migration using GAP. In *Proceedings of Joint Agent Workshops and Symposium (JAWS 2013)*, pages 11-13, Wakayama, Japan, 2013 (in Japanese).

[2] **Masayuki Higashino**, Toshihiko Sasama, Takao Kawamura, and Kazunori Sugahara: Speeding Up Mobile Agent Migration using Caching. In *Proceedings of the 9th Forum on Information Technology (FIT 2010)*, volume 4, pages 91-95, RM-010, Kyushu, Japan, 2010 (in Japanese).

Non-Refereed Domestic Conferences and Workshops

[1] Shin Osaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Approach for Debugging of Mobile Agent System. In *Proceedings of the 15th IEEE Hiroshima Student Symposium (HISS 2013)*, Koyama Campus of Tottori University, 2013 (in Japanese).

[2] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Method to Reduce Data Traffic on Flash Migration of Mobile Agents. In *Proceedings of the 64th Chugoku-Section Joint Convention Record of the Institutes of Electrical and related Engineers (RENTAI 2013)*, pages 246-247, Tsushima Campus of Okayama University, 2013 (in Japanese).

[3] Shin Osaki, Shinya Otagaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Proposal of Debug Environment for Mobile Agents. In *Proceedings of Joint Agent Workshops and Symposium (JAWS 2013)*, pages 134-138, Hotel Laforet Nanki-Shirahama, 2013 (in Japanese).

[4] Shin Osaki, Shinya Otagaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Study of Remote Debugger

for Mobile Agent Systems. In *Proceedings of the 12th Forum on Information Technology (FIT 2013)*, pages 215-218, Koyama Campus of Tottori University, 2013 (in Japanese).

[5] Shinya Otagaki, Shin Osaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Proposal of a Debugging Method focused on Partial Interactions among Agents for Mobile agent Systems. In *Proceedings of the 12th Forum on Information Technology (FIT 2013)*, pages 211-213, Koyama Campus of Tottori University, 2013 (in Japanese).

[6] Shinya Otagaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Proposal of Mobile Agent System Debugger. In *Proceedings of the 14th IEEE Hiroshima Student Symposium (HISS 2012)*, pages 438-441, Okayama Prefectural University, 2012 (in Japanese).

[7] Shinya Otagaki, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Proposal of a Break Function of Debugger for Mobile Agent Systems. In *Proceedings of the 63th Chugoku-Section Joint Convention Record of the Institutes of Electrical and related Engineers (RENTAI 2012)*, pages 380-381, Shimane University, 2012 (in Japanese).

[8] Tadafumi Hayakawa, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Operation and Evaluation of a Distributed e-Learning System. In *Proceedings of the 63th Chugoku-Section Joint Convention Record of the Institutes of Electrical and related Engineers (RENTAI 2012)*, pages 382-383, Shimane University, 2012 (in Japanese).

[9] Tadafumi Hayakawa, **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: Multimedia Streaming Based on P2P System Using Mobile Agents. In *Proceedings of the 13th IEEE Hiroshima Student Symposium (HISS 2011)*, pages 159-161, East Hiroshima Campus of Hiroshima University, 2011 (in Japanese).

[10] **Masayuki Higashino**, Kenichi Takahashi, Takao Kawamura, and Kazunori Sugahara: A Proposal of an Agent Migration Protocol using WebSocket. In *Proceedings of the 62th Chugoku-Section Joint Convention Record of the Institutes of Electrical and related Engineers (RENTAI 2011)*, pages 146-147, Hiroshima Institute of Technology, 2011 (in Japanese).

[11] Tadafumi Hayakawa, **Masayuki Higashino**, Takao Kawamura, and Kazunori Sugahara: A Proposal of a Streaming Method of Fragmented Video Files on Distributed e-Learning Systems. In *Proceedings of the 62th Chugoku-Section Joint Convention Record of the Institutes of Electrical and related Engineers (RENTAI 2011)*, page 148, Hiroshima Institute of Technology, 2011 (in Japanese).

[12] **Masayuki Higashino**, Shinichi Motomura, Toshihiko Sasama, Takao Kawamura, and Kazunori Sugahara: A Code Transfer Method for Traffic Reduction in Migration of Mobile Agent. In *Proceedings of the 11th IEEE Hiroshima Student Symposium (HISS 2009)*, pages 357-359, Yamaguchi University, 2009 (in Japanese).

[13] **Masayuki Higashino**, Shinichi Motomura, Toshihiko Sasama, Takao Kawamura, and Kazunori Sugahara: A Proposal of a Fast Migration Method for Mobile Agents using Code Caching. In *Proceedings of the 60th Chugoku-Section Joint Convention Record of the Institutes of Elec-*

trical and related Engineers (RENTAI 2009), page 191, Hiroshima City University, 2009 (in Japanese).