EECS 425

Effective Mobile Agent Migration Mechanism on Load Distribution System

Masayuki Higashin
Tottori University

Kenichi Takahashi
Tottori University

Takao Kawamura
Tottori University

Kazunori Sugahara
Tottori University

Abstract
A mobile agent system is a distributed system that is constructed from software modules called "mobile agent" which are able to work autonomously and migrate between machines. A mobile agent is able to continue working on migrations thus which carry program codes. However, as the scale of software becomes larger, so the number and the size of program codes becomes greater, and thus the mobile agent system' s performance decreases because a mobile agent has to carry program codes on the migrations. In this paper, we propose a mobile agent migration mechanism based on program code caching and evaluate its validity.
Keyword: mobile agent, platform, cache.

1. Introduction

Nowadays, a wide variety of services with large-scale computer networks have been being provided thanks to global diffusion of the Internet. In order to realize these services, a method which any components existing in networks work together as one system is proposed, but it requires complexly programing techniques. Thus, a mobile agent has been being proposed as a technique that developers can design distributed systems by using an easy-to-understand model [1, 2]. We have been proposing a meeting scheduling system and a P2P e-Learning system based on a mobile agent, and showing effectiveness of these systems [3, 4].

However, in such systems based on mobile agent techniques, because a migration of a mobile agent has to transfer not only application data but also program codes and its runtime states, a migration of an agent increases network traffic than a socket communication or a RPC (Remote Procedure Call), and the performances of the systems are decreased [5].

Due to this, various researches that aimed to improve the performance of a mobile agent system have been conducted [6, 7, 8, 9, 10]. However, because many of these studies are dependent on an algorithm of interactions of agents, it complicates behavior of agents and development of systems. As a result, an advantage of a mobile agent that can design distributed systems by using an easy-to-understand model is lost. Additionally, because these algorithms are optimized for a specific system, it is limited a range of applications.

In this paper, we propose that reduces time required to migrate of agents by improving a mechanism of agent migrations. Our proposed method caches program codes into source and destination machines when agents migrate. When agents of same type migrate concurrently, or when agents of different type which have same program codes migrate concurrently, AREs prevent unnecessary transfer of program codes by using caches of these agents. When agents of same type migrate, or when agents of different type which have same program codes migrate, AREs prevent unnecessary transfer of program codes by using caches which are stored when these agents migrated in past times. Additionally, when these agents migrate concurrently, AREs cancel duplicative transfer of same program codes. Therefore, even where many agents migrate concurrently at the same time, the performance degradation of the cache is prevented.

The rest of the paper is organized as follows. Section 2 defines an internal data structure of a mobile agent and classifies this data structure into cacheable data or un-cacheable data. Section 3 proposes our agent migration protocol by using cache mechanisms. Section 4 evaluates this proposed method. Section 5 describes difference between our method and existing methods. Finally, section 6 concludes the paper.

2. Cache Structure

2.1 Data Structure of Mobile Agent

This paper proposes a method that reduces time required to migrate of agents by caching. First of all, we define an internal data structure of a mobile agent. Figure 1 shows architecture of a mobile agent system. In AREs (an agent runtime environments), multiple agents work concurrently and are able to migrate between AREs via networks. An agent is constructed from a runtime state area and an application area, and a program code area that contains program codes [11]. A runtime state area has information of states of the agent during executions of tasks such as call stacks, program counters, etc. An application area has any data that is specified by developers of the agent. A program code area has a set of program codes that is described behaviors of the agent. This constructions cover specifications of MASIF Specification by OMG [12] and Agent Management Specification by FIPA [13, 14].

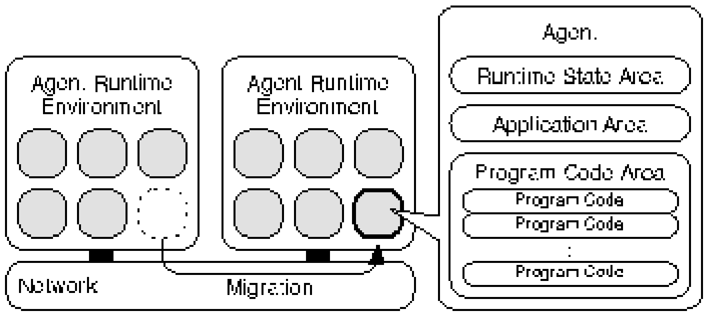

Figure 1. The structure of a mobile agent system.

2.2 Cacheable data in Mobile Agent

The mobile agent consists of a runtime state area, an application area, and a program codes area. These areas are classified into cacheable and un-cacheable data. Cacheable data must be static because data continuously changed is difficult to be reused. A runtime state area is un-cacheable because it changes continuously according to a behavior of a mobile agent. We cannot say application area is cacheable or un-cacheable because it depends on the implementation. It is cacheable if data rarely change, un-cacheable if data change continuously. A program code area is cacheable because program codes usually do not change. We aim at a general cache mechanism, and therefore we focus on only the program codes for cache.

2.3 Identification of Program Codes

In a general system development, in order to improve software reusability, program codes are divided into modules. Similarly in a mobile agent system development, large agents are divided functionally into small agents, and these some agents use common program codes. For example, when developers design agents, they reuse libraries for some agents. There are also code clones are generated by copy and paste. If these program codes can be shared between agents, the performance of cache can increase.

However, when agents use different versions of program codes, or when agents that are developed by different venders work together, there could be agents that use program codes that have a same name but a different behavior. In this case, a program code cannot be identified by the name because the cache coherence is lost. Therefore, the cache store identifies program codes by contents of those program codes. For example, in the case of JVM (Java Virtual Machine), in order to identify program codes, the cache store uses a hash value generated from Java byte codes as a identifier of program code. Thus, our proposed method realizes the cache coherence and can reuse program codes between different agents.

3 Agent Migration Mechanism

3.1 Agent Migration Session

In this paper, 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.

3.2 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 destination ARE, such as a meeting pattern [15], 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.

3.3 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.

4 Experiments

4.1 Experimental Environments

We implemented our migration mechanism on Maglog (Mobile AGent system based on proLOG) [16], 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 [17]. The ARE of Maglog runs on a Java Platform Standard Edition 6 Development Kit.

4.2 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 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.

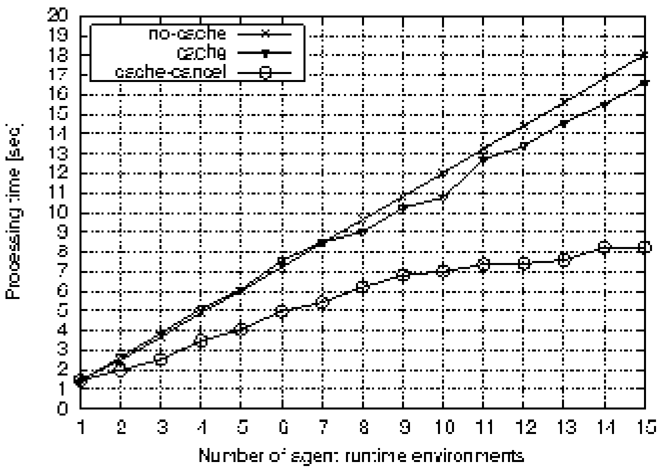

Figure 2. The time required for processing of a load distribution.

5 Related Works

Several researchers have proposed a method that improves performance of a mobile agent system. [6, 7, 8] have proposed that agents who communicate remotely migrate to same node in order to communicate locally. [10] has proposed that agents determine whether migrate by size of data communication traffic that is compared interactions with migrations.

These approaches reduce data communication traffic of a mobile agent system. However, the advantage that can design distributed systems by using an easy-to-understand model is lost because these depend on behaviors or interactions of agents.

MASIF Specification [12] 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. [18] 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.

6 Conclusion

This paper proposed that an agent migration mechanism to reduce migration time of agent. The proposed mechanism reduces transfers of program codes when agents migrate by using the cache technique. 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 a mobile agent framework, called 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%.

7 References

[1] Hurson A R, Jean E, Ongtang M, Gao X, Jiao Y, Potok T E. Recent advances in mobile agent-oriented applications. *Mobile Intelligence: Mobile Computing and Computational Intelligence*, 2012, pp.106-139.

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

[3] Kawamura T, Hamada Y, Sugahara K, Kagemoto K, Motomura S. Multi-agent-based approach for meeting scheduling system. *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, 2007, pp.79-84.

[4] Motomura S, Nakatani R, Kawamura T, Sugahara K. Distributed e-Learning

System Using P2P Technology. *Proceedings of the 2nd International Conference on Web Information Systems and Technologies*, 2006, pp.250-255.

[5] Maeda N, Nakajima S. A Quantitative Evaluation Method of Mobile Agent for Distributed System Development. *IPSJ Journal*, Information Processing Society of Japan, 2002, 43(7): 2330-2339.

[6] Noto M, Numazawa M, Kurihara M. Empirical Evaluation of Traffic Performance of Inter-Agent Communication Systems with Mobile Agents. *IEEJ Transactions on Electronics, Information and Systems*, The Institute of Electrical Engineers of Japan, 2004, 124(3): 904-911.

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

[8] Miyata N, Ishida T. Placing Agents in Massively Multi-Agent Systems. *The IEICE transactions on information and systems*, The Institute of Electronics, Information and Communication Engineers, 2007, 90(4) :1023-1030

[9] Lee Y, Kim K. Optimal migration path searching using path adjustment and reassignment for mobile agent. *Proceedings of the 4th International Conference on Networked Computing and Advanced Information Management*, vol.2, 2008, pp.564-569.

[10] Chia T-H, Kannapan S. Strategically mobile agents. *Proceedings of the 1st International Workshop on Mobile Agents*, 1997, pp.149-161.

[11] Fuggetta A, Picco G P, Vigna G. Understanding code mobility. *IEEE Trans. on Software Engineering*, vol.24, 1998, pp.342–361.

[12] Object Management Group, Inc.. Mobile agent system interoperability facilities specification. 1997.

[13] Foundation for Intelligent Physical Agents. Fipa agent management specification (sc00023k). 2004.

[14] Foundation for Intelligent Physical Agents. Fipa abstract architecture specification (sc00001l). 2002.

[15] Aridor Y, Lange D B. Agent design patterns: Elements of agent application design. *Proceedings of the 2nd International Conference on Autonomous Agents*, 1998, pp.108-115.

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

[17] Rizzo L. Dummynet: A simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, vol.27, 1997, pp.31-41.

[18] Braun P, Rossak W. Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit. Morgan Kaufmann Publishers Inc., 2005.