DATA TRAFFIC REDUCTION FOR MOBILE AGENT MIGRATION

Masayuki Higashino, Kenichi Takahashi, Takao Kawamura and Kazunori Sugahara
Graduate School of Engineering, Tottori University, Tottori, Japan

Keywords: Mobile agent, Data traffic reduction, Cache.

Abstract: In this paper, we propose a method of reducing data traffic on mobile agent migration. Mobile agents are able to simplify network programming with autonomous process migration as compared with inter-process communication. However, mobile agents increase data traffic by transfer of program codes when migration. Therefore, many researchers have proposed for reducing data traffic on mobile agent system with agent behaviour. These methods complicate the algorithm of agent behaviour. On the other hand, we focus on a mechanism of mobile agent migration, and our method is fully independent from agent behaviour. Our method reduces program code traffic with cache on mobile agent runtime environment. We have implemented our technique on a mobile agent framework, and experimented on a practical mobile agent system. As a result, the system's performance has improved to 52%.

1 INTRODUCTION

In recent years, various devices such as personal computers, mobile phones, car navigation systems, televisions and etc., are connected to the Internet. Additionally we are able to use wired (e.g., xDSL, FTTx) or wireless (e.g., Wi-Fi, WiMAX) connections to the Internet easily by improvement of networking infrastructures. Therefore, becoming a ubiquitous network society is a reality. Advances in technology such as RFID, sensor and etc., computers embedded in every environment are able to work together to autonomous. Thus, the user will be able to receive services appropriate to the situation.

In order to develop a system that is constructed with a wide variety of autonomous computers will require a complex network programming. In a traditional network programming based on IPC (Inter-process communication) such as Socket and RPC (Remote Procedure Call), it is necessary to develop a separate program for senders and receivers. If there are changes to programs of senders or receivers then it is necessary to change the programs on both sides. It makes development of the system difficult.

Therefore, a mobile agent technology has attracted attention. Mobile agents simplify network programming using autonomous process migrations instead of inter-process communications. A Mobile agent running on agent runtime environments (ARE) built on computers, and it is able to migrate between AREs. It is possible to develop network programs without being aware of communications APIs (Application Programming Interface) and communications protocols by using the single concept of mobile agent migration. A mobile agent is composed of a runtime state and program codes and any other data. A mobile agent transfers these elements together when migrate between AREs. Therefore, a mobile agent is able to continually process same tasks at before and after migration. Thus, aggregated network programs in mobile agents make it possible to easily develop distributed systems.

However, mobile agents degrade the performance of systems by increases of data traffics. Because mobile agents transfer not only data but also a runtime state and program codes. For this reason, many agent behaviour algorithms have been studied to reduce the amount of traffic behaviour (Chia and Kannapan, 1997; Jurasovic et al., 2006; Lee and Kim, 2008; Miyata and Ishida, 2008). However, these algorithms are complex. Thus, it lowers the easiness of developments.

In this paper, we propose a method to reduce traffic by improving the agent transfer mechanisms with a caching, not the agent behaviour algorithms. The agent is composed of a runtime state, program codes set and arbitrary data. The runtime state and members of program code set and arbitrary data changes with the agent activities. In contrast, the each program code is immutable data. The set of program codes is

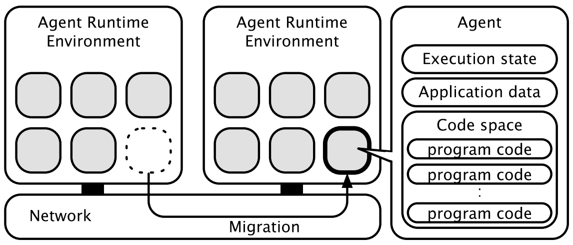


Figure 1: A mobile agent system overview.

constructed by immutable program codes. Therefore, program codes are able to cache into AREs.

2 DESIGN

2.1 Mobile Agent System

Figure 1 shows an overview of a mobile agent system. A mobile agent is able to running on AREs. The agent is able to migrate to another AREs via networks. The mobile agents consist of an execution state, application data, and program codes. The execution state contains variables such as call stack pointers, program counters and more. It is ever changing while the mobile agent is executing. The application data contains various resources that depended on mobile agent systems. The code space contains program codes. The program codes contain tasks of the mobile agent. The mobile agent read and executes these program codes dynamically. Such a mobile agent system is based on Fuggetta's executing units model (Fuggetta et al., 1998).

2.2 Structure of Mobile Agent

The construct of the agent contains cacheable spaces and un-cacheable spaces. The execution state cannot be cached because they change from time to time along with the behaviour of the agent. The application data contains the data for a variety of implementation dependent mobile agent system, the possibility that the cache implementation dependent. Thus, in this paper, we do not consider the application data cache. In reality, the program code space is a set of program codes such as subroutines and classes. These program codes have a name. However, if we try to manage program codes with just the name of the program code, when the contents of the program code has been rewritten, because it loads and executes a program code is not intended to be the agent cannot ensure cache coherency.

2.3 Identification of Program Code

Therefore, in order to know that the content of the program code is changed, the agent runtime environment manages program codes using not only their names, but also their hash values as unique identifiers.

Thus, the agent runtime environment ensures program code cache coherence between agents. In addition, without polluting the namespace between different agents, that share program codes have same names but different contents, the agent runtime environment is able to save memory usage.

2.4 Mobile Agent Migration with Cache

Figure 2 shows a sequence diagram of agent migration with program code caches. The source ARE transfers an agent execution state, application data, and a set of program code identifiers to the destination ARE. The destination ARE determines whether program codes is cached in a local cache space using program code identifiers. If uncached program codes exist then the destination ARE requests transfer of uncached program codes from the source ARE. Thus, the caching makes reduction of data traffic on agent migration.

3 IMPLEMENTS

We implemented our proposal method on Maglog (Motomura et al., 2006). Maglog is our proposal mobile agent framework. Its ARE is implemented in Java and runs on Java Runtime Environment (JRE). The agents are java objects that can run concurrently using threads.

The program code identifier is generated using a hash function (SHA-1) to Java bytecode. 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 are extremely small.

The agent execution state, application data, program code identifier, and program code are converted to the byte array by Java Object Serialization, and transferred between AREs using HTTP/1.1.

4 EXPERIMENTS

4.1 Overhead of Cache Mechanism

In the case of an agent transfer using the cache mechanism, it requires the transfer of program code iden-

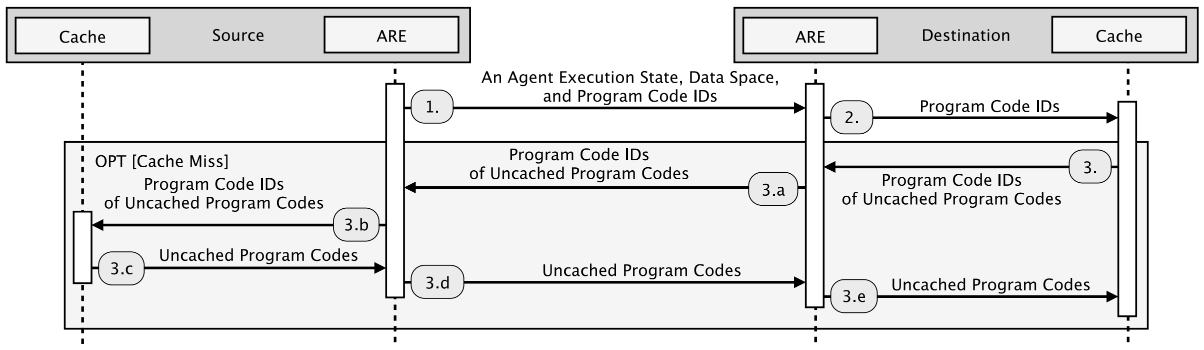


Figure 2: A sequence diagram of an agent migration with program code caches.

tifiers and the reference to cache space. Therefore, The agent migration with cache is slower than without cache on the first an ARE. In other words, an overhead of the cache mechanism become clear at the agent migration to the undiscovered ARE.

In order to confirm the overhead of cache mechanism, the agent migration time on a one-way migration was measured. The results ware obtained using two computers of a Intel Core 2 Duo processor 2.66 GHz and 4 GB RAM. These two computers are connected via Ethernet of 1000 BASE-T. An ARE of Maglog runs on a JRE 1.6.0_17 on Mac OS X 10.6.2. A mobile agent consists of 1020 program codes and their total size is 842 kB.

Figure 4(a) shows the results of measurements. In the first agent migration, it shows the overhead of caching. On the other hand, after second agent migration, the migration time is decrease by the caching effect.

4.2 Effect of Cache on Agent Migration Patterns

In order to confirm the effect of proposed method on various agent migration patterns as shown in the Figure 3, the agent migration time was measured by shuttle, round, star, and random pattern. The agents, the computers, and the network speed used to measure are the same as section 4.1.

Figure 4(b), Figure 4(c), and Figure 4(d) shows results of agent migration time measurement on agent migration patterns in the case of shuttle, round and star. In the migration patterns of all, the migration with cache is slower than the migration without cache at the first migration. However, after that, the migration with cache is faster than the migration without cache.

Figure 5 shows results of migration time measurement and regression curves on the random migration pattern. As a migration times increases, the regression curves of migration with cache approach the regression curves of migration without cache.

4.3 Effect of Cache on Practical System

We have experimented with a meeting scheduling system (Kawamura et al., 2006) in order to evaluate the effectiveness of the proposed method in practical systems.

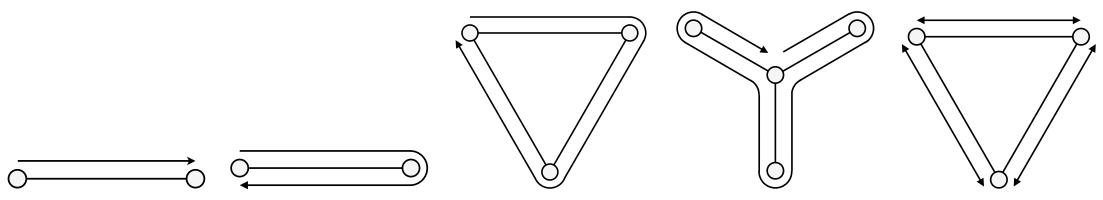
This meeting scheduling system is developed with Maglog. When a user intends to call a meeting, he only inputs information about the meeting. On behalf of the inviter, mobile agents move around each invited user's computer to ask whether he is able to join the meeting and negotiate with him if necessary. In this system, a large number of agents migrate over the network. Thus, if users want to call many meeting, it causes increasing the number of agents. As result, the system performance degradation occurs.

The results shown have been obtained using 11 computers of Intel Pentium 4 processors 3.0 GHz with 1 GB RAM and connected via 1000 BASE-T Ethernet. The ARE runs on JRE 1.5.0 on Turbolinux 10 (Kernel 2.6.0). The experiment measured the processing time required to arrange meeting dates. However, processing of user input required parts have substituted by dummy program.

In the result, In the first trial, there is no difference between with cache and without cache. However, in the second trial, the migration without cache required 25 s. On the other hand, the migration with cache required 12 s. From these results, the performance of meeting scheduling system has improved to 52%.

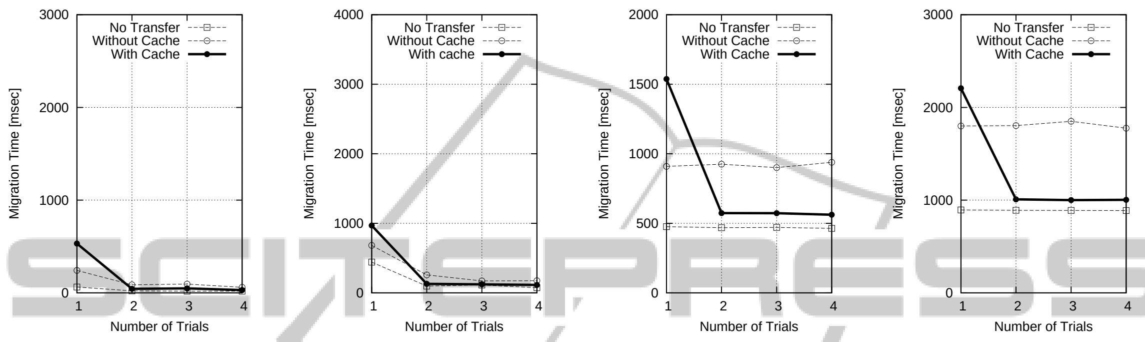
5 CONCLUSIONS

In this paper, we propose a method of reducing data traffic on mobile agent migration. We focus on a mechanism of mobile agent migration, and our



(a) One-way pattern. (b) Shuttle pattern. (c) Round pattern. (d) Star pattern. (e) Random pattern.

Figure 3: Patterns of agent migration.



(a) One-way pattern. (b) Shuttle pattern. (c) Round pattern. (d) Star pattern.

Figure 4: Results of agent migration pattern.

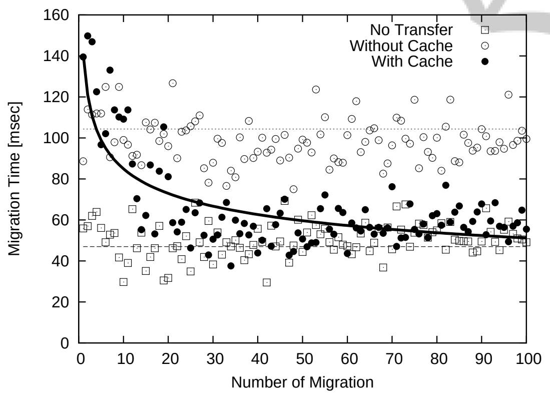


Figure 5: Results of random migration pattern.

method is fully independent from agent behaviours. Our method reduce program code traffic with cache on mobile agent runtime environment.

We have implemented our technique on a mobile agent framework, and experimented on a practical mobile agent system. As a result, the performance of system has improved to 52%.

REFERENCES

Chia, T.-H. and Kannapan, S. (1997). Strategically mobile agents. In *Proceedings of the First International Workshop on Mobile Agents*, pages 149–161, London, UK.

Fuggetta, A., Picco, G. P., and Vigna, G. (1998). Understanding code mobility. *IEEE Transactions on Software Engineering*, 24:342–361.

Jurasovic, K., Jezic, G., and Kusek, M. (2006). A performance analysis of multi-agent systems. *International Transactions on Systems Science and Applications*, 1(4):335–342.

Kawamura, T., Motomura, S., Kagemoto, K., and Sugahara, K. (2006). Meeting arrangement system based on mobile agent technology. In *Proceedings of the 2nd International Conference on Web Information Systems and Technologies*, pages 117–120.

Lee, Y. and Kim, K. (2008). Optimal migration path searching using path adjustment and reassignment for mobile agent. In *Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management*, pages 564–569.

Miyata, N. and Ishida, T. (2008). Community-based load balancing for massively multi-agent systems. In *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 28–42. Springer.

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