Cooperative Task Processing by Separating and Fusing Multi-Mobile-agents

Yasuhiro Tsuchida Masahito Yamamoto Hidenori Kawamura Azuma Ohuchi

Laboratory of Harmonious System Engineering,
Research Group of Complex Systems Engineering, Graduate School of Engineering,
Hokkaido University, Sapporo, Japan, Kita 13, Nishi 8, Kita-ku, Sapporo, 060-8628, Japan
Phone +81-11-716-2111(Ext.6498), Fax +81-11-706-7834
E-mail: tuti,masahito,kawamura,ohuchi@complex.eng.hokudai.ac.jp

Abstract: We develop the Multi-Mobile-agents system for realizing effective cooperative task processing in the network environment. In this system, agents are separated / fused by the Place and migrated to another computer. A Place can assign agents to other places by agents' migration to be flat the time to execute agents' action. In this paper, the effectiveness of this system is shown by experimental results applying an agent given simple task.

1. Introduction

With rapid growth of network technology in recent years, the technology that helps users' work in network environment also develops.

The Mobile-agent technology is one of them and it is used for collecting information from vast Internet space. A Mobile-agent is a program which is not restrained from a system in which the program launched. A Mobile-agent is able to migrate from a Place to another Place and managed by software called 'Place'.

The Mobile-agent system which consists of Multi-Mobile-agents is expected to realize the distributed computing, though there are difficult problem to assign divided tasks to agents and to gather their solutions. In this paper, we propose the functions of 'Separation' and 'Fusion' as the framework to assign the tasks and to gather solutions. Moreover, we construct a Mobile-agent system that consists of the Places which can make agents separating, fusing and moving to another light load Place.

The effectiveness of this system is shown by experimental results applying an agent given simple task.

2. Mobile-agent system

The Mobile-agent system is a system which consists of some server software called 'Place' and Mobile-agents. The Place manages execution of Mobile-agents and communication of data (agent). The Mobile-agent carries out its given task with migrating from a Place to another Place.

Fig.1 shows general Mobile-agent system. The system is expected to apply to various kinds of fields. The Electric Market Place by Telescript[1] is one of the realized Mobile-agent systems. The Telescript is the script language like the C++ to describe the Mobile-agent in the Electric Market Place.

The Mobile-agent description language based on Java is expected to construct more general Mobile-agent system. Aglet[2] and Kafka[3] are the typical script languages by Java. Although their languages get advantageous point by using Java, they are influenced by Java's restrictions (i.e. an agent is not able to migrate to another place with preserving register value and state of thread, so an agent after migrating can not resume its process with same states of pre-migration.).

In our research, Mobile-agent system (**Fig.2**) is realized based on Java. That is by some reason (an agent of this system is not necessary to preserve its register value and state of thread in migration, Place and agents don't depend on their Operating System, and so on...). The following explains the system.

2.1 Architecture of Place

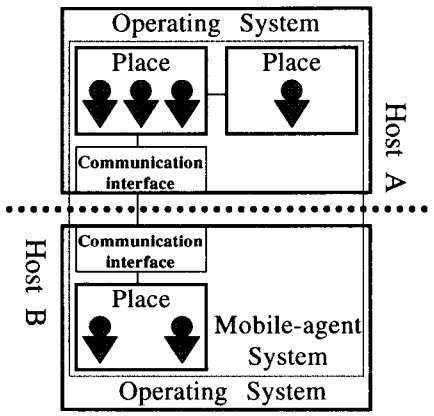


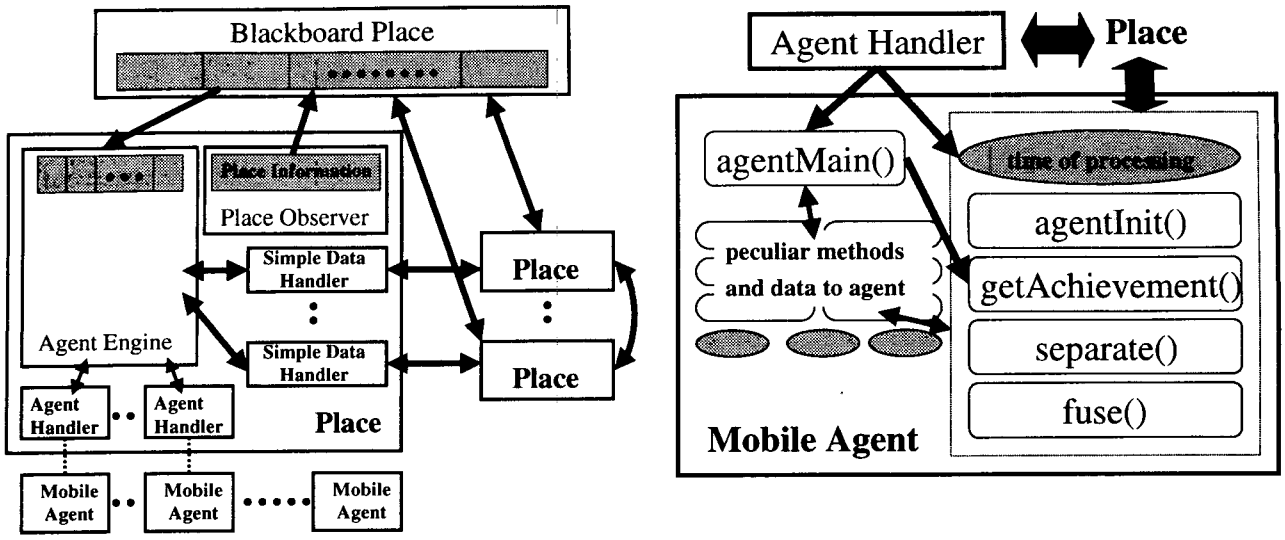
Fig.1 General Mobile-agent system

The Place is the server software in order to manage Mobile-agents and make them migrating. The Place consists of Agent Engine and Place Observer (shown in **Fig.2 (a)**). The rolls of them are as follows.

2.1.1 Agent Engine

The Agent engine is the main part of the Place and carries out following procedure sequentially in a turn.

1) **Calling agentMain() of all agents**
Agent Engine calls agentMain() of all agents. The Agent Engine is able to call the method in parallel by using Agent Handler which is introduced in order to assign threads to agents effectively in system implementation. The Agent Handler also measures the time to execute the method and writes it to *time of processing* which is one of the inner states of the agent. Agent Engine gets the *time of processing* from all agents to calculate *performance* which is one of the inner states of Place. The *performance* is the sum



(a) Architecture of Place

(b) Architecture of agent and relation with Place and Agent Handler

Fig.2 Architecture of realized Mobile-agent system

total of all *time of processing*.

2) **Separating, fusing and migrating agents**
Agent Engine makes agents separation, fusion and migration. The detail of these procedures is shown in section 3.

3) **Extracting and preserving received data**
Place sends and receives string messages and agents by Socket communication between another Places. The received data is stocked in Simple Data Handler. The Agent Engine picks out the data from every Simple Data Handlers and writes them to inner status of Place if necessary. If it receives agent, it undertakes to manage the agent.

2.1.2 Place Observer

There is a Blackboard Place (**Fig.2 (a)**) in this system. It is preserving *performance* value of all Places that are sent from Place Observer of each Place. The Place Observer is introduced for sending and receiving the *performance* values which are used for separation, fusion and migration by the Places. The sending and receiving of them is performed every fixed period. As it does frequently, the *performance* values of other Places are trustily, though the load of communication also increases.

2.2 Architecture of Agent

An agent on this system must implement following five methods.

1) **agentInit()**
The method agentInit() is immediately called by Place when an agent starts its process in the Place. The Initialization process when an agent visit new Place is described here.

2) **AgentMain()**
The method agentMain() is called by Place (The agent-Main() is called by Agent Handler in fact). The main procedure that an agent carries out is described here. The Place can call agentMain() of all agents' in 1 turn.

3) **getAchievement()**
The method returns the value which shows the achievement of its task. It expresses as the real value from 0.0 (means that its task isn't achieved at all) to 1.0 (means that its task is completed).

4) **separate()**
The method returns some separated agents (detail is shown in section 3).

5) **fuse()**
One agent fuses another agent who is given as argument value (detail is shown in section 3).

The architecture of an agent and relation with Place (Agent Handler) is shown in **Fig.2 (b)**.

3. Separation / Fusion

The Separation and Fusion is framework to assign tasks to agents and to gather solutions from agents. They are implemented as the methods separate() and fuse(). Although the implementation of them depends on the task and realizing agent, separate() must returns some agents who deal divided task, an agent who is called the fuse() must adopt an agent requests to fuse.

Fig.3 shows an example of separation / fusion. In this example, the task of agent is to calculate summation from 1 to n. separation corresponds division of calculation domain and fusion corresponds summation of solutions that a fusing agent has (detail is in **Fig.3**). The methods separate() and fuse() are called by Place if the Place determines they are necessary. Namely, the agents implement only way of separation and fusion, their operations are carried out by the Place appropriately. The timing to separate, fuse and migrate is determined as following rules.

3.1 Timing to separate

When there is a gap of *performance* value among the Places, separation occurs in this system. Namely the agents in heavy load Place are separated and sent to light load Places to make the load of Place by agents being flat

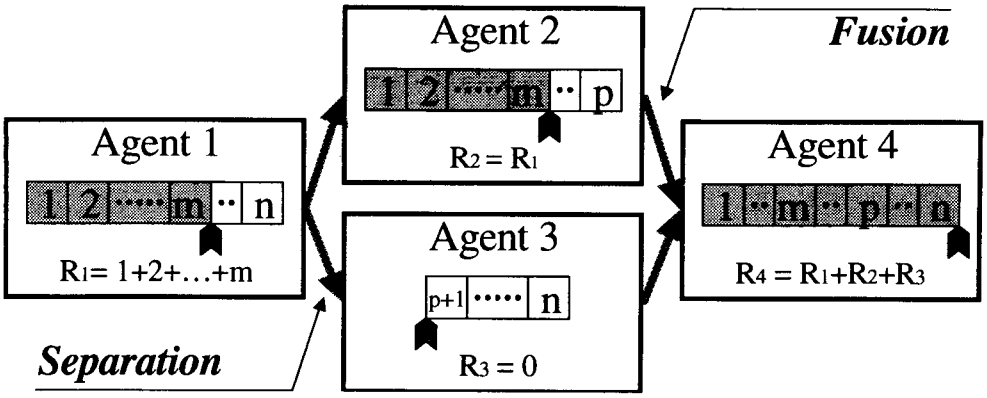


Fig.3 The Example of Separation and Fusion

The agent is constructed to calculate summation from 1 to n. Agent 1 has finished summation form 1 to m and the result of the calculation is R_1. By the separation of Agent 1, two agents are made. One is Agent 2 who has the result R_1 and given the task (summation from m+1 to p). Another is Agent 3 who has the given task (summation form p+1 to n). They perform calculation and find their result (R_2 and R_3). Finally they fuse to Agent 4 who has conclusive result R_4.

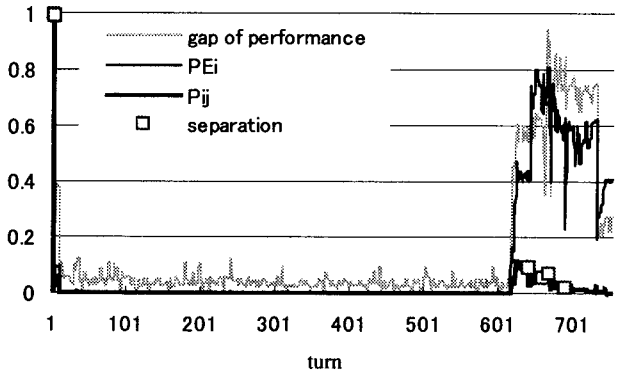


Fig.4 Relation among *PEi*, *Pij* and a gap

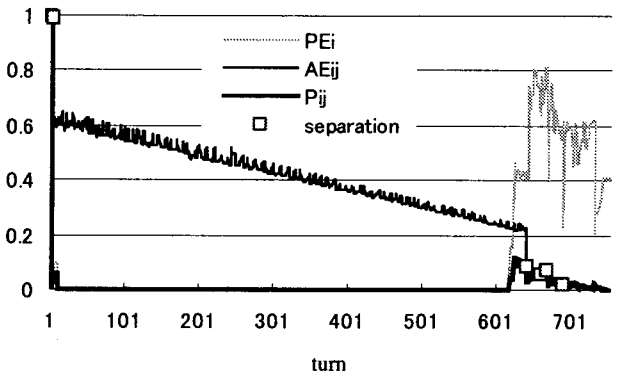


Fig.5 Relation among *AEij*, *Pij* and *PEi*

To reduce the gap means that the load of executing agents' task is distributed averagely among Places. The following is an equation to calculate the probability in which an agent *j* in Place *i* separates (*Pij*).

$$P_{ij} = AE_{ij} \cdot PE_i \qquad \ldots(1)$$

$$AE_{ij} = \frac{TP_{ij}}{\sum_l TP_{il}}(1 - AC_{ij}) \qquad \ldots(2)$$

$$PE_i = \min\left(\frac{1}{N}\frac{PF_i}{\sum_m^N PF_m}\frac{1}{\overline{PF}^2}\sum_m^N (\overline{PF} - PF_m)^2, 1\right) \qquad \ldots(3)$$

The *Pij* is influenced by an agent *j* as *AEij* and Place *i* as *PEi*. In these equations, the *PFi* and *Ai* mean *performance* value and the number of agents in Place *i*, the *TPij* and *ACij* mean an agent *j*'s *time to processing* and returned value of getAchievement() in Place *i*. The *PEi* in equation (2) becomes larger value as the gap of *PFi* among Places more increases. The agents who are made by separation migrate to the lightest Place to distribute the load of Place. By these terms, separation (and migration sequentially) occurs in high probability at the Place of which the load is larger than around Places. To distribute the agents who made by separation make a gap of load among Places reducing.

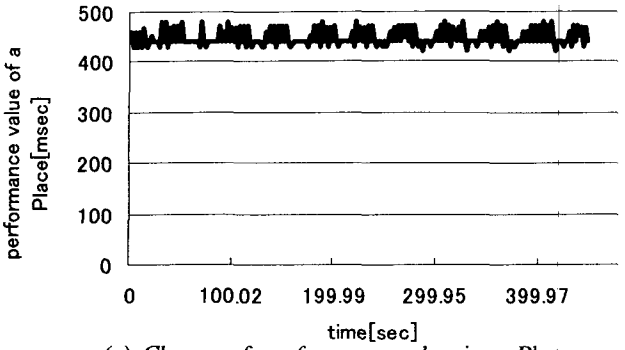
3.2 Timing to fusion

The probability of fusion is determined by the returned value of getAchievement(). Namely, the more agents finished their task the higher the probability of fusion increases. An agent who found solution of its task (getAchievement() == 1.0) fuses certainly. Consequently the Place that the number of agents decreases by fusion becomes light and the agents in heavy Place are separate and migrated to this Place.

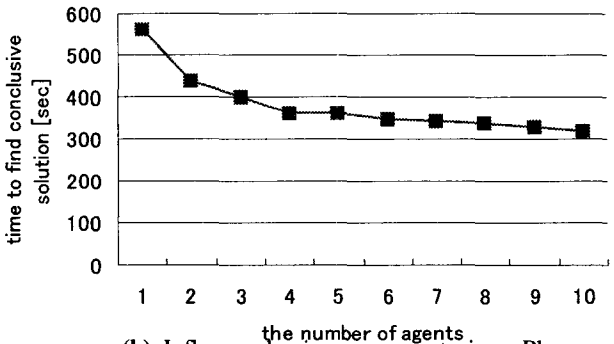
4. Experiment

We verify its effectiveness from the experimental results. An agent following **Fig.3** is realized to this experiment. The number of Places is from 1 to 5. The agent is launched at one of these Places to calculate summation from 1 to 100000000. First, we show the nature of the elements of the equation to separate. In **Fig.4** and **Fig.5**, y-axis denotes the probability of separation *Pij*, a gap of *performance* value and the elements of the equation (*AEij* and *PEi*). These figures show the relation between the elements of equation and *Pij*.

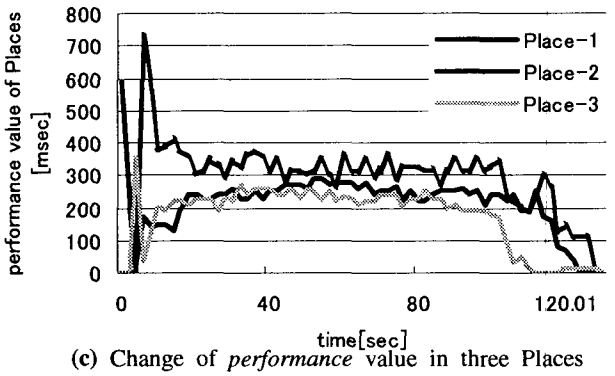
(a) in **Fig.6** shows a change of the *performance* in case single Place was running. The figure shows that the *performance* is approximately constant (swinging is brought from machine condition or Java influence). The (b) in **Fig.6** shows the relation between the number of agents and *performance* in a Place. The number of agents



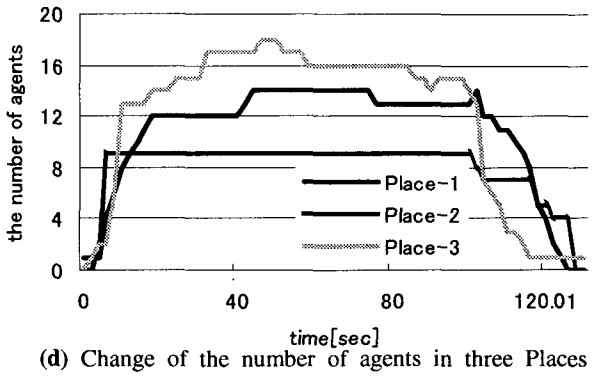
(a) Change of *performance* value in a Place



(b) Influence by increase agents in a Place



(c) Change of *performance* value in three Places



(d) Change of the number of agents in three Places

Fig.6 Experimental results

makes improvement of *performance* in single Place. The improvement is brought by effectiveness of Agent Handlers (in this experiment, three Agent Handlers ran in parallel, however the improvement where the number of agents is more than 3 may be influenced by Java.

The (c) and (d) in **Fig.6** show a change the *performance* value and the number of agents in case the number of Places is three. In the each *performance* of Place is close to the others. It shows the load of Places is distributed by the agents' separation. According to the figure (d), we can see increase the number of agents immediately after an agent starts. When Places which didn't launched an agent started, the *performance* of these Places is 0. Therefore a gap among Places becomes large so that separation occurs frequently. Some agents are created by separation in the Place which launched agent first and are distributed them to other Places. And it keeps on occurring at all Places until a gap becomes small. The number of agents changes a little until the agents who found their task appear. The gap is again large by decrease the number of agents by fusion and separation occurs.

Fig.7 shows the relation between the number of Places and the time to find conclusive solution. As the number of Places increases, the time to do decreases. This figure shows the effectiveness to perform the task by separating / fusing Mobile-agent in large number of the Places.

These experimental results show that we can finish a task more quickly as the number of Places more increases.

5. Conclusion

We realized the Mobile-agent system by the separating / fusing Mobile-agents and Place. Moreover we showed the effectiveness by the experimental results. What we constructs the system without Blackboard Place and sophisticate the rule of separation and fusion are our future works.

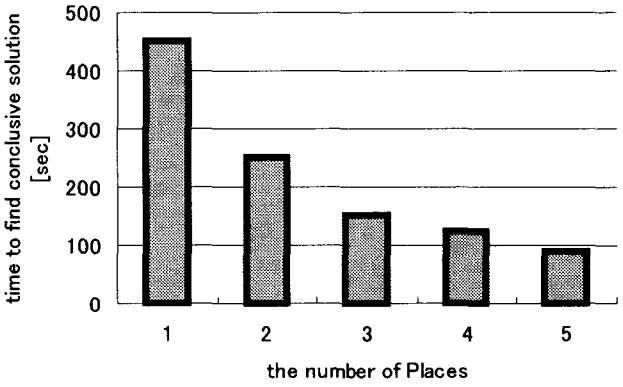


Fig.7 Relation among the number of Places and the time to find conclusive solution

References

[1]White, J. E.: Telescript Technology: The Foundation of the Electronic Marcketplace, White paper, General Magic Inc., http://genmagic.com/Telescript/, 1996

[2]Lange, D. B and Chang, D. T.: IBM Aglets Wor kbench-Programming Mobile Agents in Java. IBM Corp., White paper, http://www.ibm.co.jp//trl/aglets, 1996

[3]Takeshi Nishigaya: Design of Multi-Agent Progra mming Libraries for Java, Fujitsu Laboratories Ltd http://www.fujitsu.co.jp/hypertext/free/kafka/paper/