Total Performance by Local Agent Selection Strategies in Multi-Agent Systems

Toshiharu Sugawara[1] Satoshi Kurihara[2] Toshio Hirotsu[3] Kensuke Fukuda[5]
Shinya Sato[4] and Osamu Akashi[4]

[1]NTT Communication Science Laboratories
2–4 Keihanna City,Soraku-gun
Kyoto 619-0237, Japan
sugawara@t.ecl.net

[2]Inst. of Scientific and Industrial Research
Osaka University
kurihara@ist.osaka-u.ac.jp

[3]Dept. of Information and Computer Sciences
Toyohashi University of Technology
hirotsu@entia.org

[4]NTT Network Innovation Laboratories
3–9–11 Midori-cho, Musashio
Tokyo 180-8585 Japan
{sato,akashi}@t.ecl.net

[5]National Information Institute
2–1–2 Hitotsubashi, Chiyoda-ku
Tokyo 101-8430, Japan
kensuke@nii.ac.jp

ABSTRACT

In order to achieve efficient progress in activities such as e-commerce and e-transactions in an open environment like the Internet, an agent must choose appropriate partner agents for collaboration. However, agents have no global information about the whole multi-agent system (MAS) and the state of the Internet; therefore, they must select the appropriate partners based on local knowledge and local observations. In this paper, using a multi-agent simulation, we discuss how total MAS performances are affected by local decisions when agents select partners to collaborate with. We also investigate how MAS performances change and how network structures between agents shift according to the progress of agents' local learning and observations. We then discuss the relationship between task load and agent network structure. This relates to estabilishing the optimum time when agents should learn about appropriate partners in an actual environment.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial IntelligenceMultiagent systems

General Terms

Theory

Keywords

Collaboration, Coordination, Organization, Multi-Agent Simulation, Load-Balancing

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

1. INTRODUCTION

Each agent in a multi-agent systems (MAS) must select the appropriate collaborative partner agent when assigning a task. The choice of an appropriate agent depends on the types of task and the agent's abilities and specialties. However, if there are still multiple candidate agents, the most efficient agent is preferable. Furthermore if deadlines have been set, the issue of performance/efficiency becomes dominant. For e-commerce on the Internet, for example, the selection of efficient agents (in this case, Web and database servers) provides smooth transactions for all customers. This also means that the computational resources provided by the e-commerce company are well-utilized.

The agents selected for collaborations are often memorized by a local agent for a certain period. This stored agent structure constitutes a kind of agent organization. Such an organization can avoid repeated searches for collaborating agents, retains the quality of the results, and enables agents to accurately estimate the finish time of given tasks for real-time requirements[3]. This suggests that the process by which agent teams are organized and the way appropriate partner agents are selected are important issues in designing MAS applications.

However, in an open environment like the Internet, complete information is not available; it is not possible to grasp the global states of the Internet and the complete states of the entire MAS working there. The efficiency of task execution by other agents depends upon their processing speed and load, and the communication bandwidth, traffic, and latency between agents. Hence, agents have to select appropriate (that is, efficient) partners based only on locally available information, which often contains uncertainties. This kind of local decision-making has already been used in Internet operations[2, 8, 9]. For example, Tenbin DNS[9] is an alternative DNS which, if it knows a number of servers that can provide the same content or functions, returns the IP address of the appropriate server for load-balancing and

for selecting the most efficient servers. Tenbin DNS selects servers based on local observations such as RTT (round trip time) using 'ping' or the average throughput between local hosts and target servers. A similar methodology is also used for Web server selection[2] and for route selection in multi-homed network environments[5, 6].

Here, another issue involved with scalability arises: when all agents select partner agents and form organizations based on local data, is the overall performance of the total MAS maximal or reasonable? ('Total' means not the performance of individual agents but the average performance of all agents.). This issue is important because the result is influenced by factors including task load, the agent's (CPU and processing) performance, communication costs, the agent's distribution across the Internet and the strategies of the agent's partner selections. For example, a high-performance agent a is identified as the best partner by many agents, and thus, is likely to be overloaded. This concentration worsens its observed performance data, so many agents will switch to other lower performance agents. Later, a will again be observed as a high-performance agent so they will switch to it again. Of course, these situations may depend on communication costs and the agent's *partner selection strategies (PSS)*. A number of studies have investigated how much the performance of each agent can be improved[2, 9]. From other global view points, for example, Gaston and DesJardins [4] propose an agent organizational network, and investigate what features are required to effectively form teams. To our knowledge however, there has been little research about how agents' local strategies and task loads influence the total performance of MAS and the structure of networks according to learning by individual agents.

The purpose of this paper is to show, using multi-agent simulation, how the total performance of a MAS improves or degrades when all agents statically, adaptively, or collaboratively select partner agents based on their local information. We also illustrate how the network of inter-agent structures (this network is simply called the agent network or hereafter, AN) changes and converges, and how the total performance changes according to the observed data, where an AN node corresponds to an certain agent and the edge between nodes a_1 and a_2 means that a_1 recognizes that a_2 is the appropriate agent to collaborate with. In this paper, we will first show that agents' PSSs affect the total performance and the structure of the agent network and that simple adaptation and collaboration with neighboring agents can effectively improve the total performance. Furthermore, the AN structures are heavily dependent on the frequency of the tasks occurring as well as the PSS. This result suggests that for the best performance the appropriate PSSs depend on the overall task loads.

This paper is organized as follows: In the next section, the objective and settings of our multi-agent simulation are explained. Then, as a reference for subsequent experiments we will show the results of the first simple experiment. Next, we try to improve the number of dropped tasks and response time by changing some performance parameters. We also examine the effects of fluctuation in selecting the best server. Finally, we will discuss some implications for the control and design of MAS derived from these experiments.

2. AGENT SIMULATION

2.1 Objective

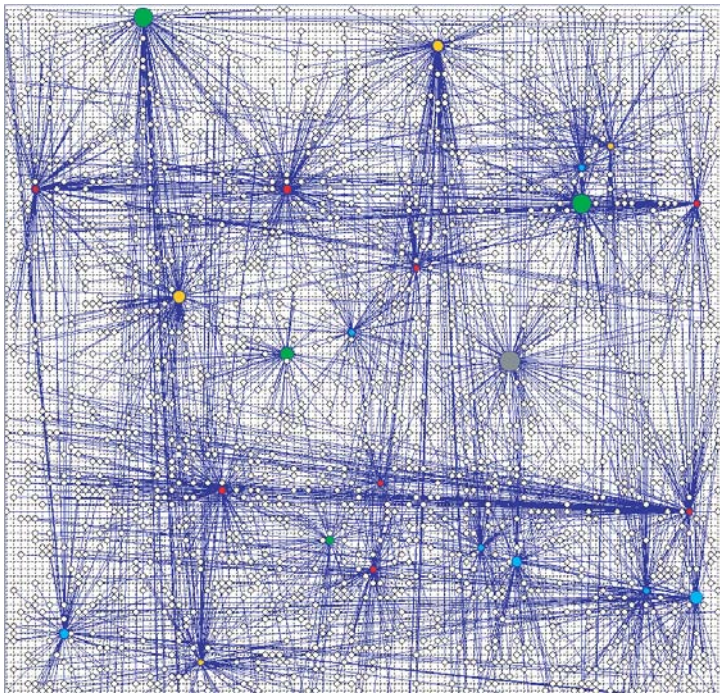
The objective of our simulation is to understand the structural changes of the agent network and to investigate the improvements and limitations of the total performance of the whole MAS when all agents make rational decisions about partner selection. Our simulation model consists of a set of agents $A = \{a_i\}$, and server agents $S = \{s_j\}(\subset A)$, which are able to execute a specific task T. When agent a_i has T, it assigns it to one of the servers that it knows, $S_i(\subset S)$. A PSS corresponds to the method whereby a_i selects s_j from S_i. To clearly understand the relationship between total performance and PSSs, we make the other parameters simple. For this purpose, we assume that all tasks are only of a single type. It is important to note that a server in this paper means an agent that can execute a specific task. Because we only consider a single type of task, the set of servers is static. In general, among the agents in a MAS, multiple tasks request each other, and different agents become servers for other tasks. We do not consider a simple client-server model but rather more flexible multi-agent models in which each agent selects appropriate partner agents depending on the task types from a local viewpoint.

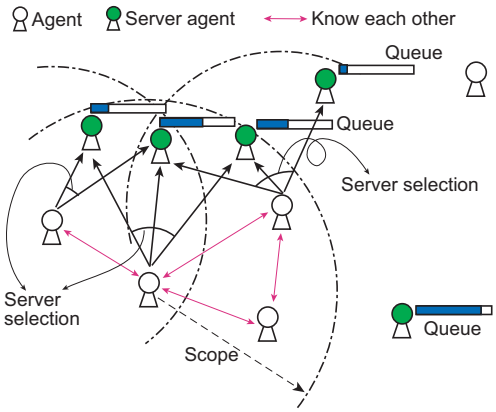
On the actual Internet, one of the observable parameters concerning agents' performance is the response time (rt), which is the length of time between the agent requesting a task and the return of the result. We assume that smaller rt is better. Another parameter for evaluating agents is the number of dropped tasks, which indicates that the requested task is dropped (silently discarded) or refused (with notice), which also is better if it is smaller. Other parameters such as type of CPU and the number of queuing tasks in servers can also been used to evaluate agents, but these parameter values are not usually locally available, so they are not used for the evaluation of servers by agents in our experiments.

Hence, using the observed data, each agent learns which server's rt is expected to be the smallest. In our experiments, agent a_i calculates the average values of the observed response time of the known server s_j (let this value be h_i^j) and (usually) selects the best server, $arg\ \min_{s_j \in S_i} h_i^j$.

To evaluate the total performance of a MAS, we adopt the average value of the observed response time in all agents. This value is denoted by RT. Thus, our experiments demonstrate how RT changes when all agents try to make rt smaller. Another parameter for understanding the total MAS is the drop number, D, which is the total number of dropped tasks. Note that a smaller drop number is better but a drop is observed only when the MAS is overloaded. Therefore, the use of this value is restrictive.

To investigate the structure of relationships between an agent and its selecting server, we introduce the *agent network (AN)* whose nodes are agents or servers and whose links indicate that an agent has determines that a server is the best according to its PSS and local data. For a positive integer γ (= 1 for the default value), called the *selection number*, agent a_i selects γ servers as appropriate servers from S_i, then assigns the given task to one of the selected servers. We look into the *degree* of the server's node in the AN because the degree denotes the level of concentration of the task assignment, that is, how many agents identify that server as appropriate for assigning tasks. The distribution of the servers' degrees expresses the balance or deviation of the overall (and potential) task load. We also investigate how the distribution changes as the agents' observation and





Above: Simulation Environment.
This figure shows the settings of our simulation environment.

Left: A Screen Snapshot of our Simulation.
A small outlined circle corresponds to an agent and a filled circle to a server agent. The sizes of the filled circle express their queue length, so the larger circles are busier. A line between agent and server expresses that when the agent is given a task, the agent assigns it to the connected server with some probability. This snapshot of the simulation is scaled down: gridsize is 100x100 with torus topology, the number of agents is 2000, and the number of servers is 24.

Figure 1: Simulation Environment and Screen Snapshot of our Simulation.

learning progress.

Our simulation can acquire other parameters for understanding the whole MAS performance, such as average values of (1) *communication cost* C (the average RTT between agents and servers), (2) the *queuing time* Q (the average duration that the incoming task is queued for before execution), and (3) *CPU time* E (the average time for executing a task). Although these parameter values are not directly observable on the Internet, they can help to estimate internal AN changes. We assume that $RT = C + Q + E$ holds.

2.2 Simulation Environment

Our simulation (Fig. 1) generates 10000 agents that are randomly placed on points of a 150 x 150 grid plane with a torus topology. Of these, 120 agents are randomly selected to be servers (so the remaining 9880 agents will assign tasks to these servers). We also introduce the Manhattan distance to this grid.

For every tick, tl tasks are generated and given to tl randomly selected agents, where tl is a positive integer. tl is called the *task load*. We can naturally extend these settings for a positive non-integer tl, which means that the average number of statistically generated tasks in every tick is tl. This is denoted by tl task/tick or simply tl T/t, where we assume that 1 tick is 10 msec. Each agent a_i given the task immediately selects a server $s_j \in S_i$ using its PSS and sends the task to s_j. Server s_j processes the committed task and returns the result to a_i. Agent a_i can observe the response time for every task and calculate the average response time h_i^j that will be used in the next PSS.

All servers are assumed to have their own CPU capabilities: each server can process a task in 100 to 500 msec. All of the servers have different capabilities that are randomly assigned *a priori*. Once assigned, these capabilities are invariant. When a task arrives at server s_j, it is immediately executed if s_j has no other tasks. The result is then returned. If s_j has other tasks, the received task is stored in its local queue and the queued tasks are processed in turn. An agent can store 20 tasks in its queue. If a_i already has 20 tasks in its queue, the new task is dropped or refused.

The communication cost (time required to send a task) is assumed to be proportional to the distance between the agent and server and ranges from 10 to 120 msec. Of course, this maximal cost is also a parameter of the simulation environment, although it is not altered in any experiment in this paper, so that we might easily understand the effect of PPS based on local observation. Each agent has its own scope depending on distance (less than 14 in our experiments), so it initially knows and can communicate with all agents and servers within the scope. For all agents to compare servers' response times and to select the best (or better) one, they have to know at least two servers. If an agent knows none or only one server, it asks all known agents for known servers; by doing this, agents can initially know two to fifteen servers.

The experimental results of all our simulations are the average value of three independent experiments derived from a random number using three different seeds. The figures of degree distribution shown in this paper are only for one of the three experiments, but their diagrams are almost identical to the those of the other cases. In these three experiments, the total sum of the capabilities all agents express is that they can theoretically process 4.7 to 5.0 tasks every tick; of course, the actual total capabilities are influenced by the communication cost, the deviation of task allocation, and server distribution in the grid plane.

3. PRELIMINARY EXPERIMENT (EXP. 1)

In the first experiment (Exp. 1), agent a_i selects γ (= 1) servers from S_i using the following PSS.

(P1) a_i selects the server $arg\ \min_{s_j \in S_i} h_i^j$ with a probability of 0.9[1]. If there are multiple servers that have the best h_i^j, one of them is randomly selected.

(P2) Otherwise, a_i selects the server with the probabilistic distribution $Pr(s_j)$,

$$Pr(s_j) = (h_i^j)^{-l} / \sum_{s_k \in S_i} (h_i^k)^{-l} \qquad (E1)^1,$$

where, $l = 3$ in this experiment[1]. In the beginning, the agents have no observed data of known servers. Hence, agent a_i initially sets $h_i^k = 0$ for the known server s_k so that it first selects s_k with no observed data. Note that (P2) introduces *fluctuation* to some extent into the PSS, since an agent may select a server whose h_i^j is not the smallest.

In the cases of tl=1T/t and tl=4T/t, we measure the performance data, RT, C, Q and E, every 20,000 ticks (20K ticks). Fig. 2 shows how these values change over time. The data in Table 1 are the average values of these parameters during a 600K to 800K tick. Fig. 3 (ii) and (iii) illustrate the cumulative degree distribution (CDD) of AN at an 800K tick, and Fig. 3 (i) is the CDD of the initial state. Thus, the CDD gradually changes from (i) to (ii) (the case of 1T/T) or from (i) to (iii) (the case of 4T/t) in Fig. 3. The last column of Table 1 shows the performance data when only (P1) is adopted (so there is no fluctuation in server selection).

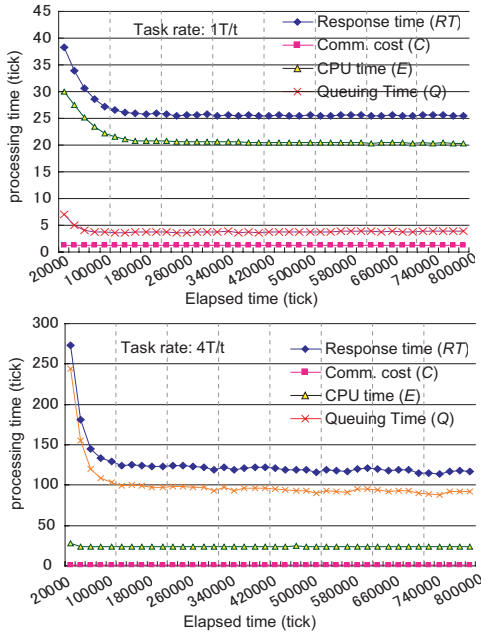


Figure 2: Total performance values (Exp. 1).

[1]We define probability 0.9 and $l = 3$ so that the server selection adequately fluctuates. Although there are a number of other ways to introduce fluctuation, we used (E1) for the first experiment. These values and expression are defined based on some experiments. We will show experimental results elsewhere when these values are changed.

Table 1: Observed performance values in Exp. 1.

Param.	0K (1T/t)	0K (4T/t)	800K (1T/t)	800K (4T/t)	4T/t (no fluct.)
RT	38.29	272.89	25.52	117.11	125.70
C	1.233	1.269	1.295	1.309	1.312
E	30.05	27.41	20.38	24.27	24.01
Q	7.01	244.20	3.85	91.52	100.38
D	0	11180.0	0	622.8	1780.1

The unit of value is a tick except D (number of tasks). 0K means the 'initial state' whose values are observed from 0 to 20K. The value of the columns at 800K are the average values of the parameters observed at every 20K ticks from 600K ticks to 800K ticks.

As indicated in Fig. 2, in this experiment the servers' performance became stable at around 120K ticks because the agents only know 2 to 15 servers, so all of them finished learning their performance. Agents select only local servers (since C hardly changed over time), so the global performance plateaus at an earlier stage. Though the theoretical capability of all servers is higher than 4T/t, approximately 600 tasks are dropped after learning. This implies that the servers' capabilities are underutilized.

The performance data are different depending on whether or not there is fluctuation as shown in Table 1. The fluctuation introduced in these experiments given by (P2) can improve average response time (RT) by 8% when tl = 4T/t. However when tl = 1T/t, this kind of improvement cannot be observed. The details are described later. Note that this experiment is similar to the one of the fixed-load case in [7], although there are many different settings; for example, the number of agents is quite large and communication costs are introduced in ours. The conflict vector in [7] corresponds to the degree of AN in ours; our experiment has more agents so it can show the structure of AN after learning.

In this experiment, the final total performance could have been further improved if all agents had known more servers. However, learning efficiency would have drastically dropped. For example, if all agents knew all 120 servers, they would require at least 1186600 (= $120 * 9880$) tasks to get performance data for all the servers. So in the case of 1T/t, it would take about 1200K ticks. In actuality, however, it would take more than 2400K ticks, because tasks are randomly assigned to agents. In practice, it is not feasible to know and attempt to use all servers on the Internet if the agents are in a massive MAS. This means that a more effective strategy for better knowing servers from a non-local viewpoint is required.

4. PERFORMANCE IMPROVEMENT AND STRUCTURAL CHANGES

We can consider a number of performance measures, but we first focus on the total drop number D in a MAS and try to reduce it. Using local information, we examine the following strategies to reduce D.

(S1) **Best two selection:** Set $\gamma = 2$. All agents select the best two servers every 500 ticks. When a task is given, the agent selects one of two servers using probability (E1) and sends it the task.

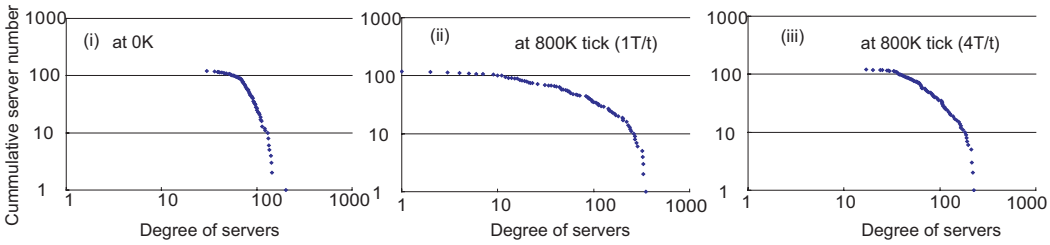


Figure 3: Cumulative degree distribution of servers.
Each point (x, y) indicates that there are y servers whose degrees are more than or equal x.

Table 2: Observed performance values in Exp. 2.

	(S1)	(S2)	(S3)	(S4)
RT	134.08	118.22	153.21	117.00
C	1.305	1.309	1.306	1.307
E	24.81	24.28	24.89	24.36
Q	107.96	92.63	127. 02	91.33
D	729.1	594.5	180.3	420.1

The unit of value is a tick except D. All data are the average values of the corresponding parameters observed every 20K ticks from 600K ticks to 800K ticks. $tl = 4$ T/t.

(S2) **Retransmission:** If their queues are full, the servers do not drop but rather refuse the requested task. The requesting agent resends the task to the server again. Agents will repeat this process five times, but if all five requests are refused, the agent gives up (drops) the task.

(S3) **Alternate selection:** Agents adopt both (S1) and (S2). The main difference is that agents have the top two servers, so after a task is refused, it may be sent to another server.

(S4) **Penalty:** This strategy is the same as (S2) except for introducing the penalty of refusal which is calculated as follows: Add double the RTT to the observed response time. This worsens the rt of the refusing server, so the agent may select another server the next time.

Table 2 shows the results of the second experiment, Exp. 2, whose tl is 4T/t. Comparing Tables 1 and 2, all performance values show only a small change in (S1) and (S2), and especially RT and D get worse in (S1). Even if an agent knows the second best server in (S1), it is probable that the server is the best for a number of other agents, so the drop rate does not decrease. Additionally, the agent requests the second best server, so the value of E (and of Q) also gets worse. Strategy (S2) is not a useful tactic for reducing D; although, after a task is refused, an agent requests it of the same server, there is a high probability that it will be refused again because with 4T/t the server's queue is likely to be still full. Note that if the agent can wait for a longer time D may become smaller, but there is a trade-off between the probability of refusal and the waiting time.

However, strategy (S3) can effectively reduce D. Alternately requesting a task of two servers simply halves the probability of refusal. Instead, RT gets larger; this implies that more tasks are stored in agents' queues, so Q also increases. Finally, strategy (S4) can only improve RT and D a little bit. We expect that this kind of penalty works well when servers are placed non-uniformly and the agents in the area where servers are sparsely settled observe more refusals so they then try to access more distant servers. The figures of cumulative degree distribution for (S1) – (S4) are not shown in this paper, but their characteristics are the same as those in Fig. 3 (iii).

4.1 Collaboration with neighbor agents

The experiments in the previous section aimed at reducing D, but the total response time RT and queuing time Q increased. These strategies kept all agents waiting for a longer time. In this section, we aim to reduce RT and Q.

The performance of the simple server selection strategies used in Exps. 1 and 2 depends on the size of the known servers. Agents that know more servers can choose more capable ones. However, the more servers the agent know, the more inefficient their learning performances are. So, a more effective and efficient strategy for agents to acquire good servers is required.

A practical strategy in the network environment is the recommendation of good servers by neighboring agents. Each agent learns the expected servers' capabilities from their local viewpoint so they can recommend to other agents the best or better server based on their observations. The basic idea is that it is possible for the better servers of neighboring agents to also be good servers for other agents. This strategy works as *collaborative filtering* using server evaluations.

To achieve this collaboration, we add the following PSS, (P3) between strategies (P1) and (P2).

(P3) With a probability of 0.01, an agent a_i randomly selects a known agent, and asks it to recommend a 'good server' s_n. If a_i already knows s_n, this recommendation is ignored. If not, a_i adds s_n to its set of known agents (so $h_i^n = 0$).

An agent that is asked to recommend a server selects a 'good server' using probability distribution (E1). The experimental results using (P1)–(P3) for 1T/t and 4T/t are shown in Table 3 and Fig. 5 (this experiment is referred to as Exp. 3). The cumulative degree distributions of Exp. 3 are illustrated in Fig. 4.

Table 3 shows that both RT and D are improved. This implies that server recommendation is an effective strategy for total MAS performance. Compared with Table 1, Exp. 3's C becomes larger – this indicates that agents assign their tasks to more distant but more efficient servers (E becomes

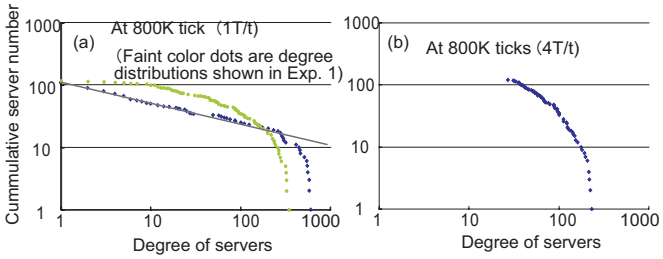


Figure 4: Cumulative degree distribution at 800K (Exp. 3).

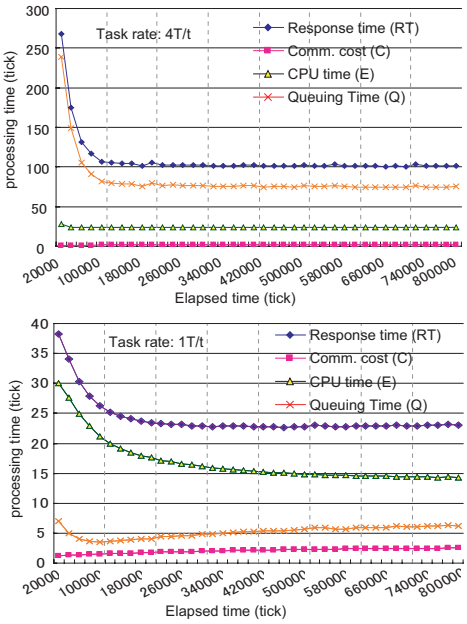


Figure 5: Total performance values (Exp. 3).

slightly smaller). The biggest difference between Exp. 1 and 3 is that the cumulative degree distribution for 1T/t follows the power distribution in Exp. 3, though it follows an exponential distribution in Exp. 1. It seems that the recommendation of good servers works like the preferential attachment in the Barabasi-Albert model[1]. However, there are some differences. For example, in our experiments, (1) all agents re-evaluate the recommended servers from their viewpoints, (2) the server becomes more inefficient if more agents assign their tasks to it, and (3) the communication costs prevent agents from accessing distant servers. So we argue that the server recommendation by agent collaboration (that is, by collaborative filtering) with local evaluation may be another mechanism for expressing the power low in the servers' degree distribution when the workload is low. Note that the cut-off in Fig. 4 (a) appears because of the limited numbers of agents in our simulation setting.

Table 3: Observed performance values in Exp. 3.

	1T/t	4T/t	4T/t with (S3)
RT	22.99	101.33	98.91
C	2.449	2.420	2.420
E	14.43	24.00	24.05
Q	6.11	74.91	72.45
D	0	309.6	0.17

The unit of value is a tick except D (number of tasks). All data are the average values of the corresponding parameters observed every 20K ticks from 600K ticks to 800K ticks.

At 800K ticks, the number of the agents' known servers ranges from 3 to 20 and increases by about 5.5 on average. This number is much smaller than 120, which is the total number of servers in this simulation. Therefore, we can say that the collaborative filtering described here is an effective learning method in our simulation setting.

Additional observations of Fig. 5 and Table 3 are:

- When tl is 1T/t, collaborative filtering produces a task concentration at higher capability servers and thus the number of servers whose degrees are small also increases. This concentration results in highly efficient total performance because the task load is low, which means that Q is still small.

- When tl is 4T/t, there is little difference between the degree distributions of Exp.1 and Exp. 3. However, Tables 1 and 3 express a difference in RT values. Comparing E values, agents did not find high capability servers, but we can understand from the C values that agents allocate their tasks to distant servers. A decrease of Q and D indicates that agents achieve load-balancing across wider regions.

Fig. 5 (b) demonstrates that RT becomes stable at an earlier time, but E, Q, and C still change around 800K ticks when $tl = 1$. This implies that the AN structure constantly changes; the degree distribution gradually changes to express power laws through this structural change.

Finally we want to point out that the collaborative filtering and strategy (S2) in Exp. 2 improves performance for different reasons if we compare Tables 2 and 3. Therefore, we additionally experimented with total performance and strategy by combining them. These results are shown in the final column in Table 3, which expresses the best performance in our experiments.

4.2 Fluctuations and Performance

The probabilistic distribution (E1) in Exp. 1 introduces a small fluctuation into the server selection decision because there is the small possibility of selecting a known server that is not the best. The last column in Table 1 shows the performance data if this fluctuation is excluded (that is, only (P1) is used) when tl is 4T/t. This data show that the fluctuation can reduce RT by 7.3%, Q by 9.7%, and D by 186%. We think that these differences are considerable.

However, this improvement by fluctuation appears only when tl is near four. The data (a) in the two graphs of Fig. 6 illustrate the distributions of $RT' - RT$ and $D' - D$in Exp. 1 over various tl ranging 0.5 to 5.2 T/t, where RT' and D' are the average of the whole response time and the number of total dropped tasks when no fluctuation was introduced (that is, (P2) was not used). The left graph in Fig. 6 shows that when $tl \leq 2.5$ the difference of RT and

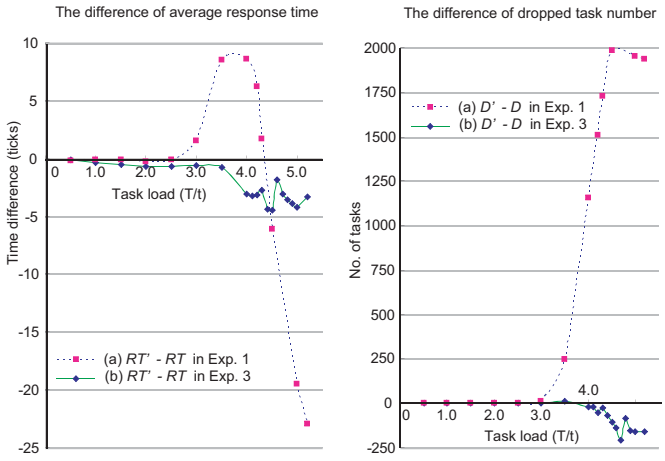


Figure 6: Performance differences (Exp. 4).

These graphs show the difference in the average response times and the number of dropped tasks when fluctuation is introduced and when it is not. Both RT and D are better if they are smaller, so if $RT' - RT$ $(D' - D)$ is positive, the fluctuation positively affects the average of the total response time (the number of dropped tasks).

RT' is almost zero (or more precisely, slightly negative). Then, fluctuation (by P2) can improve the average response time between 2.5 and 4.4, but worsens it when $tl \geq 4.4$. However, D is always better if fluctuation is introduced. We must note again that the theoretical total performance of all servers in our experiments varies from 4.7 to 5.0 and the average value is 4.86.

These figures show that the agents' best selection is not accurate when $2.5 \leq tl \leq 4.4$ in this experiment although agents select, in both cases, the best server within a restricted scope, which is the set of initially given servers. This implies that there is more uncertainty in server selection, in this range of tl. When the total load is low, the agent's PSS without collaborative filtering is relatively accurate, because all servers' queues have enough vacancies so that their observed capabilities are mainly affected by the servers' capabilities. A similar situation also appears when $tl \geq 4.4$, because when all servers' queues are almost full the observed data simply reflects their capabilities. When $2.5 \leq tl \leq 4.4$ however, the observed data are mixed with the capabilities and the queue lengths, which float over time. Therefore, the server observed to be the best may not be the best with higher probability in the next timing of the task assignment. Note that regardless of whether or not fluctuation is introduced, an agent selects servers only from the set of the initially known servers.

This uncertainty is eliminated to some extent when agents employ PSS with collaborative filtering. The graphs (b) in Fig. 6 show $RT' - RT$ and $D' - D$ in Exp. 3. As opposed to the previous case, the fluctuation worsens the total performance and the number of dropped tasks when $tl \geq 4$; this intuitively feels natural. This result implies that the collaborative filtering offers agents a more accurate and scalable decision-making process for selecting appropriate partner agents.

Table 4: Change task rate at 800K tick (Exp. 5).

	1→ 4T/t w/o CF	4→ 1T/t w/o. CF	1→ 4T/t w/ CF	4→ 1T/t w/ CF
Th	180.71	33.99	213.14	32.60
C	1.287	1.312	2.460	2.514
E	22.33	24.30	17.42	24.27
Q	158.09	8.38	193.25	5.81
D	6989	0	23790.7	0

(CF stands for collaborative filtering)

5. DISCUSSION

In real applications, selecting the appropriate agents for task allocation is quite important. We discuss this issue by taking into account our experimental results.

Effect of prior learning

Load-balancing should be achieved when the total load of the MAS is heavy. However, although agents' prior learning to identify the best way to assign specific tasks when the global MAS state is normal (that is, when the load is not heavy), it has little or no effect on performance improvement. One of our findings in both Exp. 1 and 3 was that the AN structures are quite different depending on the task load tl. Especially, Fig. 4 (Exp. 3) shows that the features and their degree distributions are different in the cases of 4T/t and 1T/t; one is exponential and another is a power-law. This implies that the agents' local decision about which server is the best is also affected by the total task load.

In Exp. 5, we switched the task load 1T/t to 4T/t at the 800K-th tick and vice versa; these results are shown in Table 4. Comparing this table and Table 1, the performance values are almost back to the values found in early stages in all cases. So, we can conclude that agents should learn the best partner agent when the MAS is totally overloaded.

Compatibility with conventional programs

Our experiments show that observation and learning by each agent, especially collaborative filtering, play an important role in achieving efficient and effective partner selection. Actually, some systems described in Section 1 such as TenbinDNS and IPv6 route selection decide their preferences based on these kinds of observations without collaborative filtering. This learning-based load-balancing is more effective than static, random, and round-robin-based methods.

Furthermore, strategy (S3) in Exp. 2 can realize more efficient load-balancing with collaborative filtering as shown in Table 3. Although strategy (S3) in Exp. 2 seems to be a natural extension of (S1) and (S2), it is not easy to deploy this function for a number of reasons. First, in real applications, modifying legacy and existing programs is avoided; for example, the previously mentioned approach of load balancing by DNSs and proxies was chosen for this reason. Normal software such as a Web browser makes a query to the local DNS to resolve the IP address from the server's name. The result is then cached for a while (because this function is provided by the OS). Therefore, agents do not change their accessing servers.

The second reason is that agents have to understand the context of the (distributed) processes running on them in order to flexibly change their accessing partners. For exam-

ple, when buying goods using an e-commerce application, if a Web browser were to reconnects to another Web server that provides the same functions during the writing of an address or credit card number, it could cause some inconsistencies. More importantly, it could create an opportunity for user information to be stolen or illegally accessed. Thus, agents cannot change the server even if the server becomes overloaded during an interaction. We assume that (S1) and (S2) can be implemented in actual environments, but that it can only slightly improve the total MAS performance. Therefore, such a strategy (S3) should be implemented in future agent programs.

Rational decision and fluctuation

In general, an agent is required to make rational decisions according to its local knowledge which may contain uncertainty. However, this decision should be made with the consideration of accurate knowledge. Our experimental results regarding the effects of fluctuations indicate that the best selection does not always lead to the best performance. When all agents do not adopt collaborative filtering, it is better that agents sometimes select the second or third best server. However, when they adopt collaborative filtering, their rational decisions usually lead to a better performance.

Our experiments and discussion suggest that collaborative filtering can provide a more accurate view for rational decision-making. Collaboration is indispensable because it can eliminate some of the uncertainties contained in local knowledge. Rational decisions are required in multi-agent design but they must be based on accurate information about non-local information, otherwise the rationality is uncertain.

For more accurate rational decision-making, information about network and other agents' states is important. As shown in Exp. 3, collaborative filtering can achieve effective PSS. This suggests that MAS design should take into account the structure and (dynamic) performance of the Internet. Looking again at the example of DNS-based load-balancing, collaborations with other DNSes in the same autonomous system (AS) or neighboring AS's must be useful. However, Exp. 5 implies that this kind of learning through collaborative filtering in mismatched situations makes the performance worse. In this case, it is better that agents know network traffic and other agents' loads. To form appropriate MAS organizations for problem solving, we believe that MAS infrastructure that has the view of Inter- and Intra-AS structures (that is, routing) and (predicted) traffic, is necessary for future MAS design[10, 11].

6. CONCLUSION

In this paper, we investigated whether agents' local strategies for selecting partner agents that are committed a specific task affect the total performance of an entire MAS. In particular, we focused on the structure of agent networks that vary according to the task load. Then, we discussed a number of findings from the experimental results: (1) learning partners by observation must be derived from the observed data gathered in the same/similar situation, (2) for more precise learning, each agent has to take into account the states of network and other agents, and (3) rational decisions by agents do not provide the best performance when the task load is at or near the critical point for total MAS capabilities.

Only a few years ago, the communications bandwidth of the Internet was narrow. Hence, to provide effective and efficient access to the (Web) servers, it was important to place them at the sites near customers or at major network ISPs. However, recent communication technologies can provide a 1M to 100M line to each home, and the communication bottleneck tends to move to the server side, because they must process a large number of requested tasks. In such an environment, it is necessary for an agent to select high-performance partners rather than closer partners. In designing a MAS, we must consider how each agent finds other agents to assign tasks. Each agent's decision, based on local information, can be improved by careful local observation and learning. This can realize more efficient MAS operation. However, our experiments show that the simple local observation and learning may not be scalable when all agents do the same thing. Collaboration, such as collaborative filtering, plays an important role in achieving more scalable load-balancing.

7. REFERENCES

[1] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[2] V. Cardellini, M. Colajanni, and P. S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, 1999.

[3] K. M. Carley and L. Gasser. Computational organization theory. In G. Weiss, editor, *Multiagent Systems*, pages 229–330. MIT press, 1999.

[4] M. E. Gaston and M. desJardins. Agent-organized networks for dynamic team formation. In *Proceedings of AAMAS2005*, pages 230–237, 2005.

[5] C. Launois, O. Bonaventure, and M. Lobelle. The NAROS approach for IPv6 multihoming with traffic engineering. In *Proc. of Int. Workshop on Quality of Future Internet Services (QoFIS2003)*, 2003.

[6] S. Lee, Z.-L. Zhang, and S. Nelakuditi. Exploiting as hierarchy for scalable route selection in multi-homed stub networks. In *Proc. of Internet Management Conference (IMC'04)*, pages 294–299, 2004.

[7] A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

[8] R. Schemer. lbnamed: A load-balancing nameserver written in perl. In *Proc. of Large Installation System Administration Conference*, pages 294–299, 1995.

[9] T. Shimokawa, N. Yoshida, and K. Ushijima. Flexible server selection using DNS. In *Proc. of Int. Workshop on Internet 2000 (held at DCS2000)*, pages A76–A81, 2000.

[10] T. Sugawara, O. Akashi, S. Kurihara, and S. Sato. An Organization-Related Information Maintenance Component. In *Proceedings of ICMAS2000*, pages 443–444, July 2000.

[11] A. Terauchi, K. Fukuda, O. Akashi, M. Maruyama, T. Hirotsu, T. Sugawara, and S. Kurihara. ARTISTE: Agent Organization Management System for Multi-Agent Systems. In *Proc. of PRIMA2005*, 2005.