A Multi-Agent System based on Specific-Profile for Efficient Distributed Computing

Youn-gyou Kook, Woon-yong Kim, Young-keun Choi
Dept. of Computing Science, Kwangwoon University
Wolgye-dong, Nowon-gu, Seoul, Korea

Abstract: - We propose the multi agent system based on profiling specifics for the distributed system, SPMaS. The job scheduling and the path finding on the distributed computing are important factors for the reliable system and running time reduction. The factors which affect the entire performance of scheduling and path finding are node's capability(CPU, memory), network state(speed, traffic frequency) and usage patterns(average available resource, connection lasting time, job request frequency, job execution frequency and error frequency). SPMaS analyses the characters of node by using agent and composes node history. The node history helps system allocate the best-suitable job for each node during job scheduling. In addition, managing groups based on the node history makes effective handling for the load balance and defect of transmission.

Key-Words: - Multi Agent System, Mobile Agent, Specific Profile, Distributed Computing

1 Introduction

The numbers of web users are increasing dramatically with advance of web technology. The user requirements for consuming the variety information are expanded in the both aspects, quality and amount, so the great computing power is necessary as the amount of computation grows. However the server-oriented internet environment lacks the utilization of the client resource [1]. To utilize it, the distributed computing methods using the mobile agent are researched at many fields. The mobile agent as a technique for distributed computing moves autonomously among nodes and does its job by itself, so it reduces communication cost, and supports asynchronous communication. These advantages actualize we can use the resource of many clients, same as the local resource.

In the distributed computing system using the mobile agent, the agent's efficient transmission and node management are very important factors to consider [2]. In the job scheduling, the best suitable job allocation can reduce the entire execution time. Also, when agent is transmitted to the more than one node, the entire execution time can be changed as the transmission order [2][3]. The general distributed systems(AMIS, MOS) use the serialization transmission like master/slave, so the entire execution time increases as the number of nodes grows [4][5]. This problem can be resolved by the parallel transmission like the binary tree. However many factors like network capability, traffic and the defect in the hardware and software affect the distributed processing, so unexpected results are produced [6][7]. The reason is that they do not consider every node's real time statement(network speed, traffic, defects, system status and usage pattern).

We propose SPMaS which allocates job optimally and reduces entire execution duration with agent transmission for the distributed computing in job scheduling, node composition and node management. The agent monitoring the specific characters of each node in this system analyzes each node history. This shows the effective and confident characteristics of a node by presenting continuous status of node not temporary status of node. This system processes job scheduling based on node history in order to allocate an optimized job on each node. In addition, it composes and manages the group of nodes based on node history, so the network problem during transmission of agents and load balancing in node management are resolved properly.

This paper is organized as follows. In chapter 2, we review related works about distributed computing system and its specification. In chapter 3, we present an architecture of SPMaS and describe job scheduling algorithm and resource management one, and in chapter 4, we show the performance evaluation. Finally we make a conclusion in chapter 5.

2 Related Works

Several public systems for distributed computing appeared [Pea]. Most of these systems however are either focused on some specific problems, or they require dedicated hardware, or are proprietary, offering little flexibility to developers. Entropia [8] is a commercial version of distributed computing over the Internet. They offer a robust technique and assistance with expertise in a seamless integration into existing network environments and in a deployment of custom applications. Condor [9] is a high throughput computing environment and can manage very large collections of distributive owned workstations.

The environment is based on a layered architecture that enables it to provide a powerful and flexible suite of resource management services to sequential and parallel applications. A Beowulf cluster [10] is built out of commodity hardware components, running a free-software operating system like Linux or FreeBSD, interconnected by a private high-speed network. It is a dedicated cluster for running high-performance computing tasks. The nodes in the cluster don't sit on people's the desks, they are dedicated to running cluster jobs.

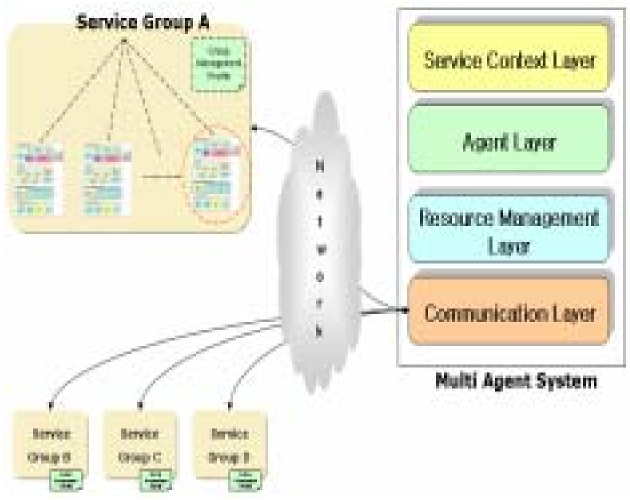


Fig. 1 Network diagram of SPMaS

3 SPMaS System

The network of SPMaS we proposed consists of multiple service groups. Each service group is classified by specific profile and node history, process distributed transaction. Figure 1 shows the network diagram of SPMaS we proposed.

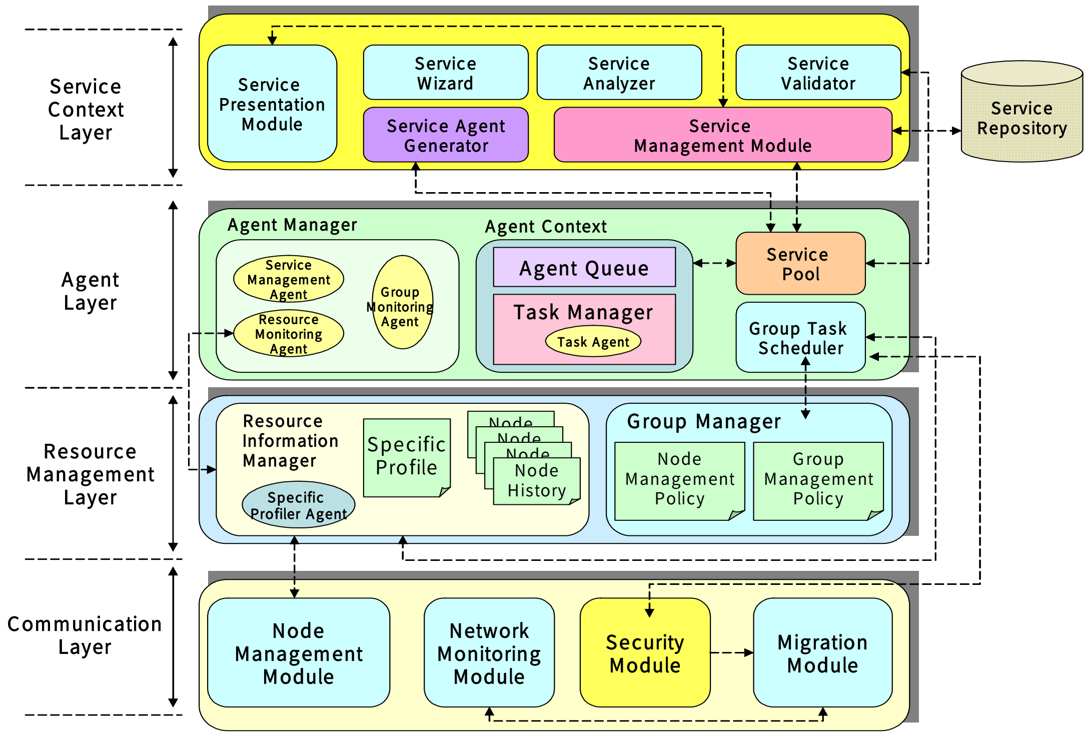
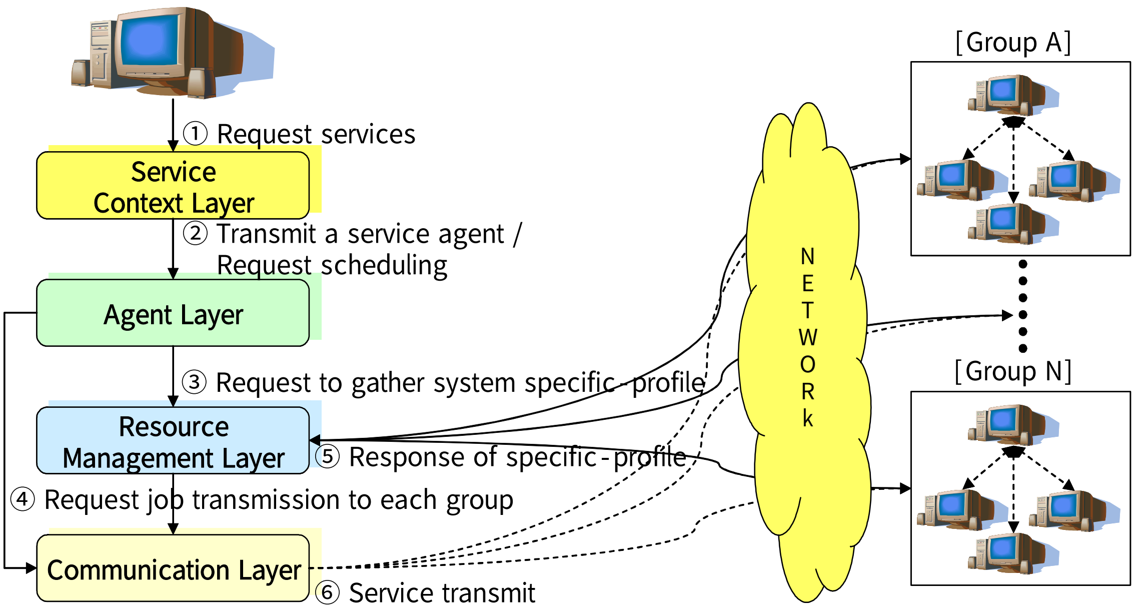


Fig. 2 Architecture of SPMaS



① Request services
Service Context Layer
② Transmit a service agent / Request scheduling
Agent Layer
③ Request to gather system specific-profile
Resource Management Layer
⑤ Response of specific-profile
④ Request job transmission to each group
Communication Layer
⑥ Service transmit
N E T W O R k
[Group A]
[Group N]

Fig. 3 Process of resource management and service request

3.1 SPMaS Architecture

SPMaS's architecture is shown at Figure 2. The service context layer creates the requested service from the user dynamically, and the agent layer provides the environment of execution for the service and group scheduling. The resource management layer manages each group, node's dynamic situation, and lastly the communication layer is in charge of supporting communication among groups and transmitting agents. When the participated node requests service, the user creates the desired service dynamically through the service context layer.

The requested service becomes mobile agent's code, and group task scheduler schedules every agent's transmission to the each service group. The best suitable amount of job is allocated based on each service group's history which is managed by the resource management layer. In the step of job transmission, Group task scheduler transmits jobs to each service group, not the entire nodes. The group re-divides jobs to the subordinate nodes based on the node history. It will reduce network traffic and load which can be a great problem if jobs are transmitted to the entire nodes. Figure 3 describes the process of resource management and service request.

The new node transmits specific profile to the basic service group after setting basic specifications. The new node is allocated to the proper service group based on the location, interests and system information expressed in the specific profile. The node allocated to the any node is added to the proper group's priority list based on the specific profile, and the group management policy which has the modified list is transmitted to the basic service group and it is synchronized.

The priority list of node is the alternative node list which will replace group managing node if it is in trouble. The list which is maintained independently on each group is managed by group monitoring agent, and is modified by checking each node's dynamic states periodically.

3.2 Service Context Layer

The user can create and search services dynamically through the service context layer. If the first node cannot find the required service at the service repository through the service presentation module, the service will be created. The service is created by the service wizard, and created service is transformed to the designated type by service analyzer, and the Service-Validator validates it. If the service is created properly, the service agent generator creates mobile agent code which can be executed on all the nodes.

3.3 Agent Layer

The Agent Layer has the agent manager for managing agent, service pool for storing created services, group task scheduler for dynamic scheduling, and agent context for executing services. The agent manager has the job agent for distributed processing, and network monitoring agent for managing node's network state, service management agent for managing service and group monitoring agent for managing each service group. The group task scheduler allocates the best suitable jobs through referring the service group's specific profile, history and group management policy.

3.4 Resource Management Layer

The resource management layer manages the service group and information for node management. It has the group management policy for node management, node management policy, specific profile and node history for managing node's performance factors.

The specific profile contains each node's resource information, network status and group management information. The communication layer creates and manages the node management policy, and group management policy contains performance of each service group and contained node's brief information. Node history is in charge of managing the changes of node specifications. Node history classifies six attributes as below.

Node_history = {SRF, SPF, AUR, NR, AST, EFR}

Definition 1. SRF (Service Request Frequency)
SRF is the number of times to request for processing the service at the node.
Definition 2. SPF (Service Processing Frequency)
SPF is the number of times to process the service at the node.
Definition 3. AUR (Average Usage Resource)
AUR is average value of the available resource that is existed permanently at the node, so we can check the possibility of service.
Definition 4. NR (Network Resource)
NR is the network resource that node is currently using, so we can check the status of available network resource when the service is requested.
Definition 5. AST (Average Service Time)
AST is the average connection duration at the distributed system, and it is used for checking the average time in one day.
Definition 6. EFR (Error Frequency Rate)
EFR is the frequency of software and hardware error occurrence, and we can check the error rate at the node.

Definition 1 and 2 decide the priority of processing service when the multiple services are requested. And definition 3, 4 and 5 decide the amount of available resource to allocate the job. Definition 6 decide whether allocating the job or not. As SRF, SPF is high frequency and AUR, NR and AST is high value, we gives to node the amount of job allocation.

```
Node status is ON {
   Start timer for current session ;
     Create nodeServiceInfoTable, nodeResourceInfoTable ;
     While( Node status not OFF ) {
        If( N_ACT instanceof SERVICE ) {
            Service currentService  = getServiceType() ;
            nodeServiceInfoTable.set
                  ( beforeService, currentService);
        } else if( N_ACT instanceof RESOURCE ) {
           Resource currentResource = getResourceType() ;
            nodeResourceInfoTable.set
                  ( beforeResource, currentResource) ;
        } else {
            increase
                  (ERROR_TYPE, ERROR_FREQUERENCY) ;
        }
        Stop timer ;
        readHistories() ;
        updateHistories() ;
    }
```

Fig. 4 Profiling specific of nodes

Each node history stores the information about node resource's change and specific profile of nodes. Figure 4 shows the algorithm for node's profiling specific 6 types of performance factors. Figure 5 shows the way to determine each node's weight based on the node histories.

3.5 Communication Layer

Communication Layer is in charges of network communication with other nodes. Communication Layer has Node Management Agent(NMA) to face the unpredictable network effect on the node management. NMA extracts user's characteristics from the node histories, and makes the priority list and group node election list based on it. Therefore, it reduces fault occurrence more than the existed uniformed selection method.

```
C_Weight = Weight of Node
Histories[] : Array of history
H_Threshold[] = Threshold of Histories
C_Rate = Change Rate of History
Nodes :
  Loop number of  Node :
    Loop number of Histories :
      History cHistory = readHistoryFile( Histories[i]) ;
      C_Weight = cHistory.readC_Weight() ;
      Threshold = readThreshold( H_Threshold[i]) ;
      Loop number of History :
        C_Rate = cal(before(histories[i]), after(histories[i]));
        If( C_Rate > Threshold) {
            C_Weight = C_Weight – C_Rate*Threshold;
        } else {
            C_Weight = C_Weight + C_Rate*Threshold;
        }
      Loop END :
      Write(C_Weight);
    Loop END :
  Loop END :
END :
```

Fig. 5 Determining each node's weight based on the node history

In addition, it considers the entire nodes when it selects subsidiary node, so it can deal with the trouble properly when the both group and subsidiary nodes are defected. The next algorithm shows the way to decide group and subsidiary nodes.

4 Performance Evaluation

SPMaS is developed in JDK 1.4.5, JNI and Visual C++ 6.0. Figure 6 shows the initial dialog, it shows the basic user information which is current status, type, operating system, VM(virtual machine), location, CPU and memory information.

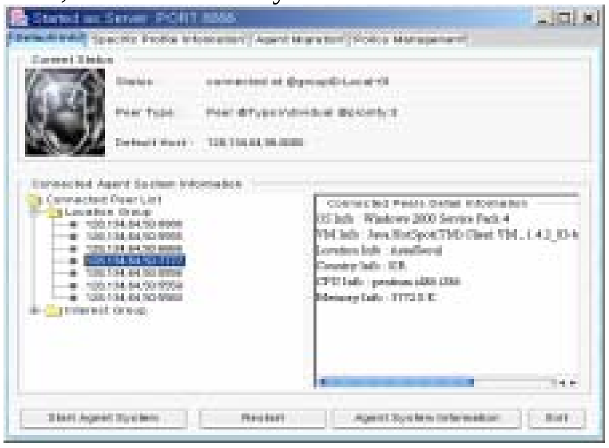


Fig. 6 SPMaS's basic user information

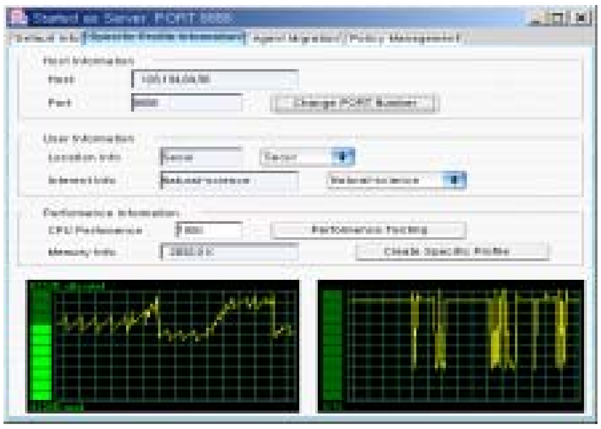


Fig. 7 SPMaS's resource manager

Figure 7 shows system's basic resource which is managed by Resource Manager. Resource Manger monitors usage, fluctuation rate of CPU and memory. Figure 8 is the screen of loading agent file and data file to schedule the requested service. The node which performs scheduling allocates jobs based on the basic service's information, node's specification profile and history.

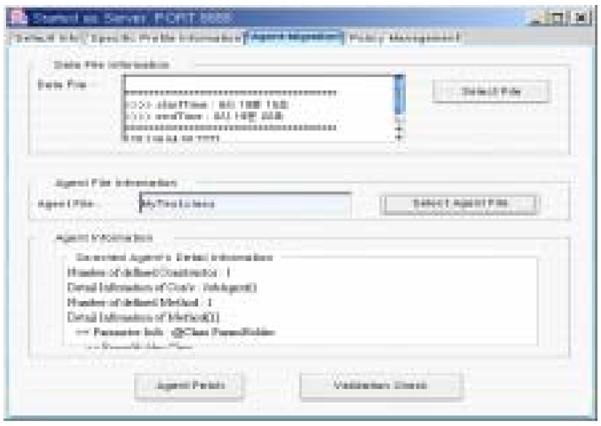


Fig. 8 Result of service request and scheduling

Figure 9 shows the screen of result collecting after requested service is finished. During collecting the result, the execution process, result and error are shown on the real time base. To measure SPMaS's performance, we test it on the environment shown follow as:

Fig. 9 Result of processing distributed transactions

1) Execution environment
   - Windows 2000 / XP : 15 PCs
   - Linux RedHat 9.0 : 5 PCs
2) Test Data
   - The analyze of log data (approximately 100MB)
   - Mersenne prime number computation
3) Number of systems to test
   - 1 PC, 2PCs, 4PCs, 10 PCs, 20PCs

In this test data, we test two types, because we need to analysis the job with heavy network load and light one. Each test was run with 2,4,10,20 PCs based on the first single PC execution. The entire result of analysis is shown at Figure 10.

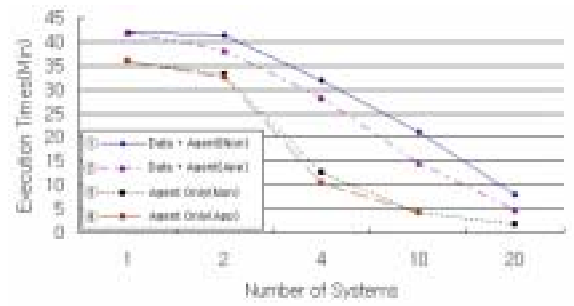


Fig. 10 Entire processing time of distributed transactions

In Figure 10, ①, ② are the log analysis which carries data, and ③, ④ are the measurement of prime number. In the log analysis, ② shows the comparison of the test with consideration of system's specification and ① shows to be test without consideration. In the result of ①, ②, the execution time of two systems is even worse than the single system, because computing is executed based on the network transmission. It means network capability is very important factor if we use the large data, so the network capability has to be concerned. Additionally, we test our system with designated conditions for the variety validation.

Firstly, we classify the status of node's resource as 3 types.

1) Availability 1%~30% : resource is low in the node
2) Availability 31%~60% : resource is middle in the node
3) Availability 61%~100% : resource is high in the node

When we execute the job with over 30% of availability, the performance is similar or little lowered. However, lower than 30% of availability means the distributing overhead was too big to ignore. Figure 11 and 12 show the execution time as availability of node resource. Figure 11 describes over 30% of availability is stable, because measurement of prime numbers does not require much CPU possession rate. We predict that if the job needs much more available resource, the result will be changed.

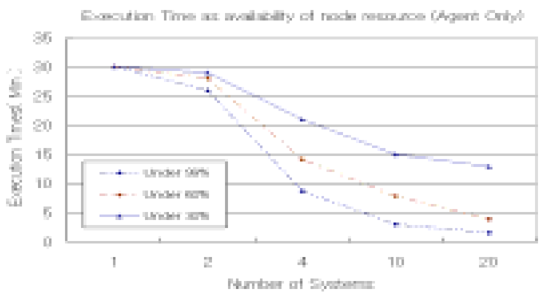


Fig. 11 Execution time that only agent processing distributed transactions without estabilishing thresold to use node's resource

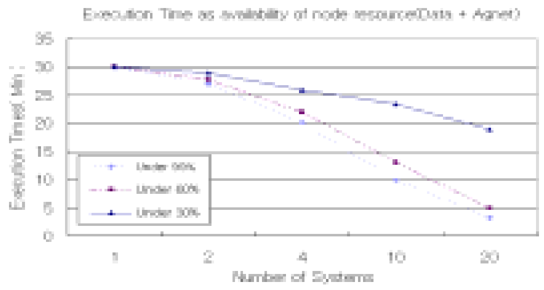


Fig. 12 Execution time that agent processing distributed transactions with data without estabilishing thresold to use node's resource

Figure 12 shows the execution time of log analysis with resource availability condition. The log analysis requires more resource than the prime number, so the execution time was increased.

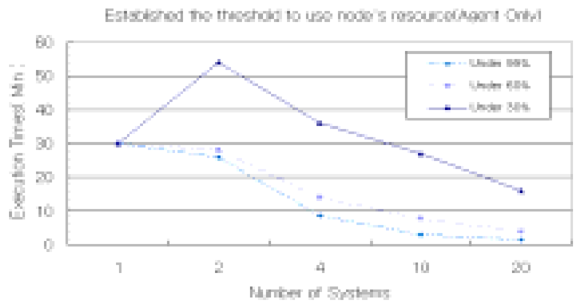


Fig. 13 Execution time that only agent processing distributed transactions on estabilishing thresold to use node's resource

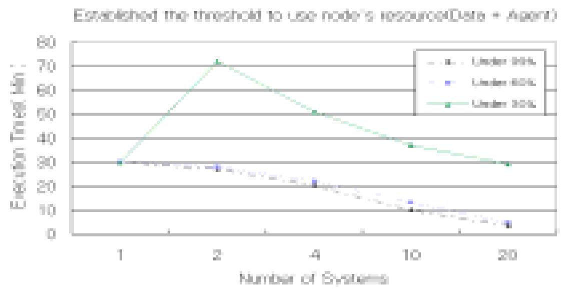


Fig. 14 Execution time that only agent processing distributed transactions with data on estabilishing thresold to use node's resource

In figure 13 and 14, we established the threshold to use node's resource. As established threshold, we cannot use the nodes which have less than 30% of resource availability. In Figure 13, we add the result with 30% threshold and 30% of resource availability. Because of threshold, when the execution time of nodes in less than 30% of availability is increased. In this case, we computed our job with 20 PCs, but the execution time was 18 minutes, so the advance of distributed computing disappeared. Therefore, we need at least 30% of resource availability for distributed computing. Figure 14 shows the lowed performance like Figure 13 when the availability of node is less than 30%. Specially, the prime number computation is more lowered than log analysis, because it needs more resource Figure 14 shows that the result of computing with 20 PCs is similar with the single computation. Based on these results, we should select the nodes which have more than the minimum available resource for the distributed computing.

5 Conclusion

The distributed computing area is focused as the computing power of clients connected WWW(World Wide Web) is increased rapidly. However, the general distributed computing system does not consider each client's dynamic state and performance during job scheduling. This leads the increment of execution time, so the entire performance is lowered. Therefore, this paper considers each client's dynamic state and performance, so the entire execution time is reduced. This system supports each client can create desired service dynamically, and applies the client's status and specification which are learned by the monitoring agent and specification learning agent, so it can reduce the entire execution time.

As the future works, we need to improve the security module for the assurance of safety in the distributed system and the service creation module for more accuracy and reliable service creation.

Reference

1. H. M. Deitel, P. J. Deitel, J. P. Gadzid, K. Lomeli, S. E. Santry, S. Zhang, Java Web Services for Experienced Programmers, Prentice Hall, 2003
2. Stefan Pleisch & Andre Schiper, Fault-Tolerant Mobile Agent Execution, IEEE Transactions on Computers, Vol.52, No.2, pp209-232, 2003
3. W. Theilmann and K. Rothermel, Optimizing the Dissemination of Mobile Agents for Distributed Information Filtering, IEEE Concurrency, pp.53-61, 2000
4. Hong-Jin Park & Jae-Hyun Lee, AMIS:Agent Migration Information System, Proc of IEEE TENCON '02, 2002
5. M. Dikaiakos and G. Samsras, A Performance Analysis Framework for Mobile Agent Sysems, Proc. First Ann. Workshop Infrastructure for Scalable Multi-Agent Systems, Proc. Fourth Int'l Conf. Autonomous Agents 2000, June 2000
6. K. Takashio, G. Seoda, and H. Tokuda, A Mobile Agent Framework for the Follow-Me Applications in Ubiquitous Computing Environment, Proc. Int'l Workshop Smart Appliances and Wearable Computing(IWSAWC '01), pp.202-207, 2001
7. Misikangas. P, Raatikainen. K, Agent migration between incompatible agent platforms, Distributed Computing Systems, 2000
8. Entropia, http://setiathome.ssl.berkeley.edu
9. Michael Litzkow, Miron Livny, and Matt Mutka, Condor – a hunter of idle workstations. In Proc. of the 8th International Conference of Distributed Computing Systems, pp104-111,1988
10. Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V.Packer, Beowulf: A parallel workstation for scientific computation. In Proc. of the 1995 International Conference on Parallel Processing(ICPP), pp11-14, 1995