Grid Service Movements in ARMS

Junwei Cao

C&C Research Laboratories, NEC Europe Ltd., Germany
cao@ccrl-nece.de

Abstract. ARMS is a grid management system that utilizes an agent-based methodology for distributed grid service discovery. In this work, additional mechanisms are implemented in the ARMS system that enable agents to cooperate with each other and adapt to the movement of grid services. When a grid service is moved from one agent to another, some un-registration and re-registration processes are invoked along the agent hierarchy. While not all of the agents in the system will be aware of the movement during a certain period of time, agents can still be able to cooperate with each other to discover the moved grid service as requested. While additional workload is involved temporarily, simulation results included in this work show that reasonable performance for service advertisement and discovery can be achieved as well as avoiding system bottlenecks and discovery failures.

1 Introduction

Grid computing [8] is becoming a mainstream technology for large-scale resource sharing across multiple organizations. Grid service monitoring and discovery is one of the essential functionalities that must be provided by the grid software infrastructure [9].

In our previous work described in [3], an agent-based resource management system for grid computing, ARMS, is developed where software agents are considered to be the main high level abstractions for grid resource and service integration. Each agent can be registered with multiple grid services and agents are organized into a hierarchy. Agents can also cooperate with each other and service information is advertised and discovered along the agent hierarchy using different strategies.

The dynamism of grid environments is the main challenge that must be addressed in grid management systems. If a grid service is moved from one place to another, previous service information has to be updated. In this work, particular processes are defined in the ARMS for grid service un-registration and re-registration when movements occur. Performance issues are discussed and corresponding simulation results are included to illustrate that the mechanism used in the ARMS is effective to achieve reasonable performance as well as avoiding system bottlenecks and discovery failures.

Most of current grid tools, including Globus [7], Legion [5], GFarm [15], UNICORE [13], Condor 12] and NetSolve [4], provide service or resource monitoring, discovery or matchmaking mechanisms. Recently released GT3 utilizes web services standards and describes grid service data in the WSDL [6]. Discovery and notification protocols are defined to query or update the service data. While a different object-oriented abstraction is adopted in Legion, similar mechanisms are also provided in the Legion resource management via so-called connections and enactors. The grid datafarm is a data grid infrastructure that utilizes an LDAP [16] based meta-server for file information management, which is similar to the previous Globus MDS implementation. In a UNICORE server, databases are used to collect detailed resource information that is used to translate abstract job descriptions. While Condor is originally motivated for cluster management and scheduling, the classified advertisement matchmaking framework can be also applied for grid management. The NetSolve agent operates both as a database and as a resource broker which is quite similar to the ARMS agent.

Most of above works focus on the implementation of data models and communication protocols. Performance issues related to monitoring and discovery have not been the key consideration of these works. In our work, we focus more on performance optimization of agent behaviors for service discovery. While there are several other related works on agent-based grid computing [10, 11, 14], the emphases of these works are different. In this work, performance issues especially when grid service movements are involved are investigated in a quantitative way according to some pre-defined metrics.

The rest of the paper is organized as follows: An overview introduction of ARMS is provided in Section 2; Mechanisms for service advertisement and discovery when service movements occur are presented in Section 3; Simulation results are included in Section 4; and the paper concludes in Section 5.

2 ARMS

Agents comprise the main components in the ARMS system; the agents are organized into a hierarchy and are designed to be homogenous [3]. Each agent is viewed as a representative of multiple grid services at a higher level of grid management. The service information of each grid service can be advertised within the agent hierarchy (in any direction) and agents can cooperate with each other to discover available services [2].

Each agent utilizes Agent Capability Tables (ACTs) to record other grid service information. An agent can choose to maintain different ACTs corresponding to the different sources of service information: T_ACT is used to record information of an agent's own registered grid services; L_ACT to record service information received from lower agents in the hierarchy; and G_ACT to record information from the upper agent in the hierarchy.

There are basically two ways to maintain the contents of ACTs in an agent: data-pull and data-push, each of which has two approaches: periodic and event-driven. An agent can ask other agents for their service information either periodically or when a request arrives. An agent can also submit its service information to other agents periodically or when the service information is updated. The frequency of the periodical service information update is also configurable. These strategies can be applied to all kinds of ACTs.

Apart from service advertisement, another important process among agents is service discovery. Discovering available services is also a cooperative activity. Within each agent, its own registered grid services (recorded in the T_ACT) are evaluated first. If the requirement can be met locally, the discovery ends successfully. Otherwise service information in L_ACT and G_ACT is evaluated in turn and the request dispatched to the agent, which is able to provide the best (or the first) requirement/service match. If no service can meet the requirement, the request is submitted to the upper agent. When the head of the hierarchy is reached and the available service is still not found, the discovery terminates unsuccessfully. This process is essential for the agent system to adapt to the service movement and avoid discovery failures as introduced in the next section.

The ARMS architecture and mechanisms described above allow possible system scalability. Most requests are processed in a local domain and need not to be submitted to a wider area. Both advertisement and discovery requests are processed between neighboring agents and the system has no central structure, which otherwise might act as a potential bottleneck.

Several metrics are defined to evaluate the performance of agent behaviors quantitatively [1]. Two of them are also used in this work: discovery speed and system efficiency. The average agent service discovery speed (v) during a certain period is calculated via the total number of requests (r) divided by the total number of connections made for the discovery (d). The average efficiency of the system (e) is considered as the ratio of the total number of requests (r) during a certain period to the total number of connections made for both the discovery (d) and the advertisement (a). A performance modeling and simulation tool is also developed to aid the performance evaluation process and simulation results can be shown in multiple views, e.g. the step-by-step view and the accumulative view.

While our previous performance studies on the ARMS focused on the impact of dynamic service information update on the system performance, the impact of adding, removing or relocating grid services on the agent performance is focused and evaluated in this work.

3 Service Movements

When a grid service is added, removed or relocated dynamically in the ARMS, additional service advertisement and discovery processes are required that have an effect on the system performance. Since the relocation can be considered as a combination of removing and adding, only the service relocation will be considered below. In this section, how service movements are processed in the ARMS is described via an example system and corresponding formal representations.

Two protocols are defined in the ARMS when a grid service is moved: the un-registration from its current agent and the re-registration to another agent. When a service requests to un-register from its agent, the agent will remove the service entry in its T_ACT; and when a service requests to register to an agent, the agent will add a new entry to its T_ACT. While the operation is simple and easy from the point of the service view, corresponding agents have to process additional advertisements to update the service information along the agent hierarchy. And these have important impact to the future service discovery processes, which is illustrated using an example system below.

The example shown in Fig. 1 is a simple agent system with two levels, one broker *B* with several agents below. Each agent maintains a T_ACT, a L_ACT, and a G_ACT. The broker only has a T_ACT and a L_ACT. Consider a typical process: the user *U* sends a request, *s*, through agent A_1, and the service can be provided by *S*. But the service *S* is just moved from agent A_2 to A_3.

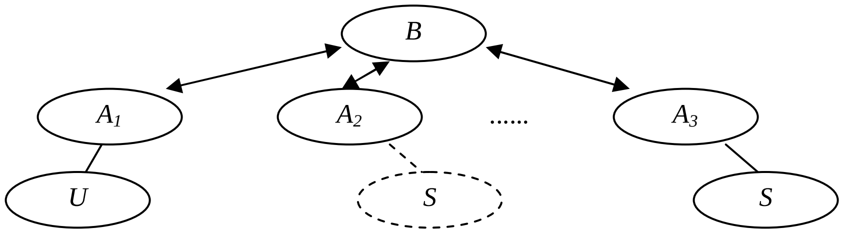


Fig. 1. An Example System

Each T_ACT and L_ACT is maintained by an event-driven data-push method, and the G_ACTs of these agents are updated using a periodic data-pull method. In this situation, when the service is moved, the related T_ACTs and L_ACTs are all updated immediately, but when the request is sent out, the G_ACTs of these agents have not been updated. How will the service discovery proceed?

The formal representation of the problem is summarized in Table 1, which includes the definitions of agents, evaluations, and processes. This is the basis for the rule-based reasoning of system dynamic processes.

Table 1. Formal Representations

Agents	A_i, *(i=1,……,n)*, one of the agents *s*, a given service request
Evaluations	*t(s)*, evaluation result of *s* in T_ACT *l(s)*, evaluation result of *s* in L_ACT *g(s)*, evaluation result of *s* in G_ACT $t(s), l(s), g(s) \in \{A_i\ (i=1,\ldots\ldots,n), null\}$ *null* means no service information is available for the request *s*
Processes	$A_i(s)$, A_i processes the request *s*

We represent the process for an agent to require a service in a logical way. The rules show the routes for a request from the original agent to reach the target agent though the service can be moved dynamically. Several basic rules are used, which formalize the service discovery process described in the last section.

Rule 1: $A_i(s) \Rightarrow A_i \rightarrow (t(s), l(s), g(s))_{Ai}$

The service discovery process in an agent is the process of looking up its T_ACT, L_ACT and G_ACT.

Rule 2: $(A_{this}, *, *)_{this} \Rightarrow ServiceFound$

If an agent is aware that the required service can be provided by one of its own registered grid services, the service discovery is successful.

Rule 3: $(null, A_{lower}, *)_{this} \Rightarrow A_{lower}(s)$

If the required service information cannot be found in the T_ACT but in the L_ACT, the request will be dispatched to the lower agent.

Rule 4: $(null, null, A_{another})_{this} \Rightarrow A_{another}(s)$

If the required service information cannot be found in the T_ACT or L_ACT but in the G_ACT, the request will be dispatched to the corresponding agent.

Rule 5: $(null, null, null)_{this} \Rightarrow A_{upper}(s)$

If an agent exhausts the ACTs, and does not obtain the required service information, it will submit the request to its upper agent.

Rule 6: $(null, null)_{broker} \Rightarrow NoService$

If a broker (head of an agent hierarchy) exhausts the ACTs (G_ACT is not maintained in a broker), the service discovery ends unsuccessful.

These rules can be organized together to reason about the route of the service discovery process in the example system. The equations are shown below. For each step, the evaluation results of all of the ACTs to the request *s* replace the correspondent parts, $(t(s), l(s), g(s))_{Ai}$, in the process automatically. The number at the end of each line indicates the rule used for the transformation.

$$\begin{aligned}
A_1(s) &\Rightarrow A_1 \rightarrow (null, null, A_2)_{A_1} && (1)\\
&\Rightarrow A_1 \rightarrow A_2(s) && (4)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow (null, null, null)_{A_2} && (1)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow A(s) && (5)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow (null, A_3, *)_B && (1)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3(s) && (3)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3 \rightarrow (A_3, *, *)_{A_3} && (1)\\
&\Rightarrow A_1 \rightarrow A_2 \rightarrow B \rightarrow A_3 \rightarrow ServiceFound && (2)
\end{aligned}$$

Three connections are needed for the A_1 to find the required service in A_3. In the G_ACT of A_1 the service is still recorded to be within the capability of A_2. A_2 still has to take part in the routing process. The routing process can be simplified if the G_ACT of A_1 can be updated some time later.

It is obvious that service movements lead to some additional system workload for service advertisement and discovery. However, two advantages introduced by the ARMS approach include:

- *Scalability*. There is no central information point in the ARMS system, which otherwise might become a system bottleneck. Grid service information is distributed and replicated in all of the agents in the system. There is no one-to-all broadcast mechanism for service information update among agents; instead, if service movements occur, service information will be updated only along the agent hierarchy between neighboring agents. Other agent may be informed about the service movement gradually when the system evolves (in the case above, through the periodic data-pull of G_ACTs). This mechanism also contributes to improve the scalability of the system.

- *Fault tolerance*. It can be seen from the above example process that even if the service information is out-of-date in some of the agents, the discovery process would not fail as long as the service still exists in the system though moved. In this situation, more service discovery steps among agents occur. Since wrong discovery results are avoided, it is considered to be a reasonable trade-off in the ARMS.

Let's consider two extreme situations when service movements occur in the ARMS: one is that every time a grid service is moved, all the other agents are informed, which leads to much more service advertisement workload but eases discovery processes; another is that once a grid service is moved, no additional service advertisement is processed, which leads to future complex discovery processes. Both of these are not optimal and the ARMS aims to balance between them to achieve better performance.

The system can have more than two levels and the services may be changed many times. The system behaviors for service advertisement and discovery may become much more complex. As mentioned in last section, the modeling and simulation tool is developed to aid the performance evaluation process of the ARMS system. As introduced in the following sections, some statistical results are achieved that enable the agent performance to be investigated quantitatively.

4 Performance Analysis

The modeling and simulation environment and the performance metrics mentioned in Section 2 are also used in this work. Additional service movement configurations are also added to the simulation environment. During each simulation step, service movements would be configured first if any exists, followed by general service advertisement and discovery simulation processes.

The multi-agent system model shown in Fig. 2 is more complex, containing 26 agents. The whole system is configured to have only one service named *Printer*. The service *Printer* is now registered to *till* and later, during the simulation, is moved to connect to *sun* (this is not shown in Fig. 2). All the other agents may or may not request the *Printer* service with a different frequency (Note that the details of requests are not given below).

This experiment is used to show the impact of service movements on the service discovery performance. All of the agents in the model use the same strategy for service advertisement and discovery. The T_ACT, L_ACT and G_ACT are used in each agent. T_ACTs and L_ACTs are maintained by event-driven service advertisement. G_ACTs are updated once every 30 steps using a periodical data-pull. The service movement mentioned above takes place at the 100th simulation step.

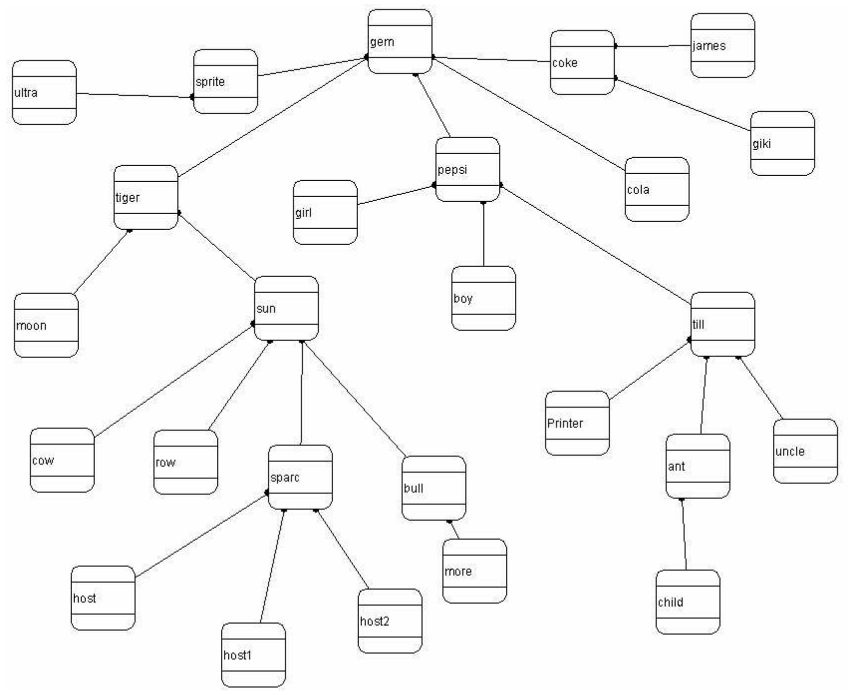
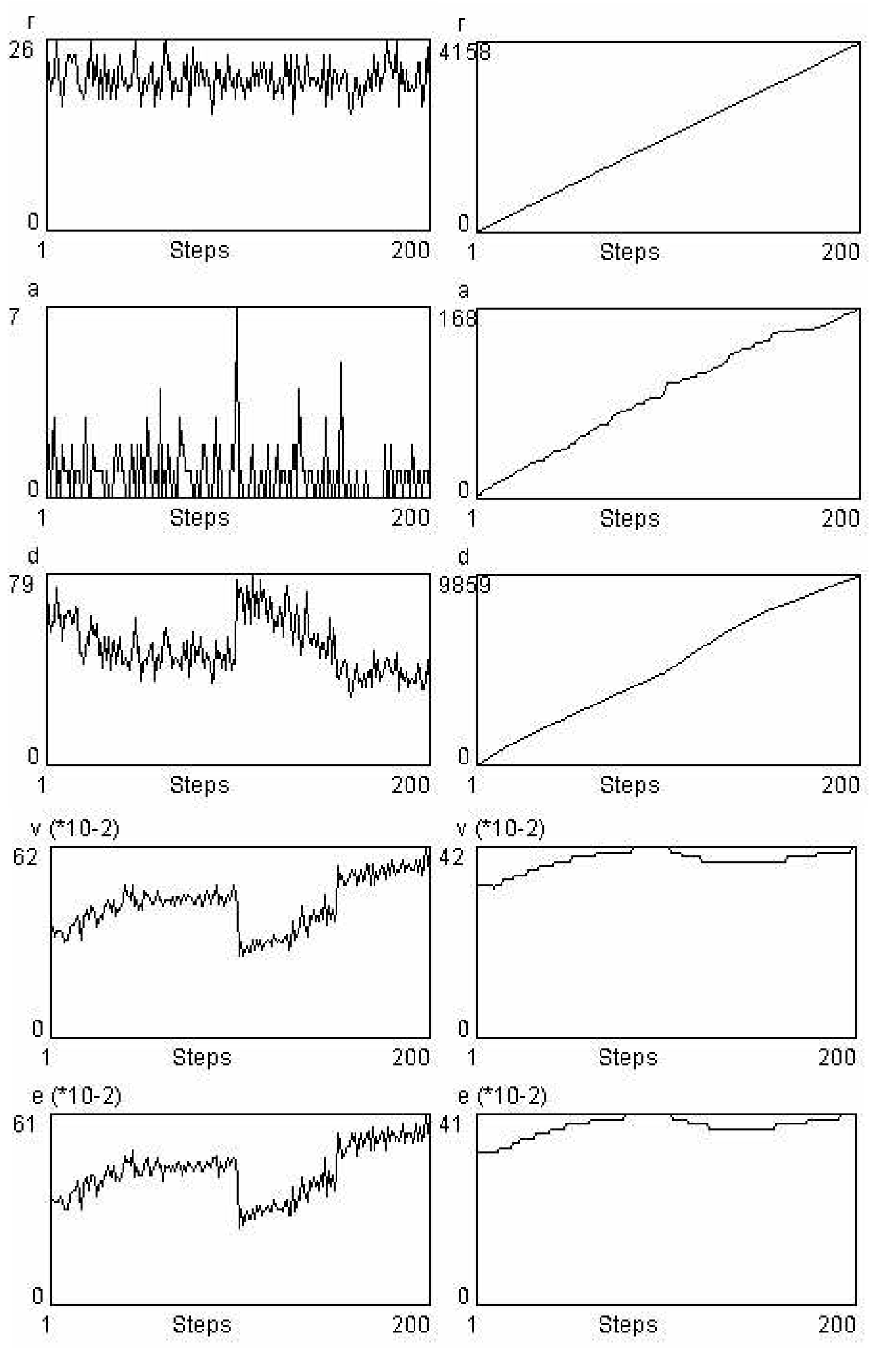


Fig. 2. Agent Hierarchy

Fig. 3 shows the simulation results for 200 steps. The curves for discovery speed (*v*), and the system efficiency (*e*) in the step-by-step view show the effect of the service movement most clearly. The whole process can be divided into five phases, which are explained in detail below.

- *Learning phase.* In the first 40 steps, the G_ACTs of the agents are updated gradually, so the discovery speed and system efficiency increase. This can be viewed as an agent learning process.
- *Stable phase.* After about 40 steps, the curves are flat at a higher level. All G_ACTs of the agents have been updated and there are no service movements, so the system runs in a steady state mode with high service discovery speed and system efficiency.
- *Service movement.* The defined service movement happens at the 100th simulation step. When the service is moved it must advertise to remove its service information from the old agent and to add the new service information to the new agent. This causes an increase of the connections for service advertisements (*a*), since corresponding agents have to advertise the change along the hierarchy. Because not all of the agents are informed immediately about the movement as described in the last section, the service information in some of the agents becomes out-of-date, which results in more workload for the service discovery (*d*). So the average service discovery speed (*v*) and system efficiency (*e*) decrease suddenly.



(a) Step-by-step View (b) Accumulative View

Fig. 3. Simulation Results

- *New learning phase*. This phase is the same as the previous learning phase. The agents learn about the new information of the service *Printer* gradually via the G_ACT updating.
- *New stable phase*. The service movement finally results in a stable state mode with higher performance. This is because *sun* is the coordinator of a larger sub-hierarchy than *till*. When the service is moved, more requests become local instead of remote, which reduces the discovery workload of the system.

This is a small example model with only one service movement. The system model is not a large-scale one and the service in the system is static during most of the simulation time. However, this simple case study gives an intuitive impression that system dynamics has a great impact on the service discovery performance. The modeling and simulation techniques developed in this work enable such kinds of problems to be investigated quantitatively.

5 Conclusions

ARMS is an agent-based grid management system that focuses more on the performance issues of grid service discovery in a large-scale dynamic grid computing environment. In this work, additional mechanisms are implemented in the ARMS system that enable agents to cooperate with each other and adapt to the movement of grid services. Two main features of the ARMS approach are scalability and fault tolerance. Some formal representation based reasoning and simulation techniques are developed to investigate the performance of agent behaviors in the ARMS system both qualitatively and quantitatively.

The adaptation to service movements enables the ARMS to be applied for more grid applications. For example, the ARMS agent-based management can be used in a data grid environment to provide a large-scale distributed data indexing service. Since in a data grid, it is unavoidable to transfer or replicate large amount of data in a large scope frequently, the ARMS methodology can be applied for dynamic data indexing and searching. Ongoing works on applications of the ARMS system also include agent-based grid workload balancing and agent-based grid workflow management and scheduling.

Acknowledgements

The authors would like to express his gratitude to members of the High Performance Systems Group at the University of Warwick, including Prof. Graham R. Nudd, Dr. Darren J. Kerbyson, Dr. Stephen A. Jarvis, Mr. Daniel P. Spooner and Mr. James D. Turner, for their previous contribution to the ARMS development.

References

1. J. Cao, D. J. Kerbyson, and G. R. Nudd, “Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing”, in Proc. of 1st IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, pp. 311-318, 2001.
2. J. Cao, D. J. Kerbyson and G. R. Nudd, “Use of Agent-based Service Discovery for Resource Management in Metacomputing Environment”, in Proc. of 7th Int. Euro-Par Conf., Manchester, UK, Lecture Notes on Computer Science 2150, pp. 882-886, 2001.
3. J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, “ARMS: an Agent-based Resource Management System for Grid Computing”, Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 135-148, 2002.
4. H. Casanova, and J. Dongarra, “Applying NetSolve’s Network-Enabled Server”, IEEE Computational Science & Engineering, Vol. 5, No. 3, pp. 57-67, 1998.
5. S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, “Resource Management in Legion”, Future Generation Computer Systems, Vol. 15, No. 5, pp. 583-594, 1999.
6. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, “Web Services Description Language (WSDL) 1.1”, W3C Note, 2001. http://www.w3c.org/TR/wsdl.
7. I. Foster, and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit”, Int. J. Supercomputer Applications, Vol. 11, No. 2, pp. 115-128, 1997.
8. I. Foster, and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.
9. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, “Grid Services for Distributed System Integration”, IEEE Computer, Vol. 35, No. 6, pp. 37-46, 2002.
10. L. Moreau, “Agents for the Grid: a Comparison for Web Services (Part 1: the Transport Layer)”, in Proc. of 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Berlin, Germany, pp. 220-228, 2002.
11. O. F. Rana, D. Bunford-Jones, D. W. Walker, M. Addis, M. Surridge, and K. Hawick, “Resource Discovery for Dynamic Clusters in Computational Grids”, in Proc. of 10th IEEE Heterogeneous Computing Workshop, San Francisco, CA, USA, 2001.
12 R. Raman, M. Livny, and M. Solomon, “Matchmaking: Distributed Resource Management for High Throughput Computing”, in Proc. of 7th IEEE Int. Symp. on High Performance Distributed Computing, Chicago, USA, 1998.
13. M. Romberg, “The UNICORE Grid Infrastructure”, Scientific Programming, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 149-157, 2002.
14. W. Shen, Y. Li, H. Ghenniwa, and C. Wang, “Adaptive Negotiation for Agent-Based Grid Computing”, in Proc. of AAMAS Workshop on Agentcities: Challenges in Open Agent Environments, Bologna, Italy, pp. 32-36, 2002.
15. O. Tatebe, Y. Morita, S. Matsuoka, N. Soda, and S. Sekiguchi, “Grid Datafarm Architecture for Petascale Data Intensive Computing”, in Proc. of 2nd IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Berlin, Germany, pp. 102-110, 2002.
16. W. Yeong, T. Howes, and S. Kille, “Lightweight Directory Access Protocol”, RFC 1777, IETF Draft Standard, 1995.