Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 31 (2005) 529–543
PARALLEL COMPUTING
www.elsevier.com/locate/parco

A time-to-live based reservation algorithm on fully decentralized resource discovery in Grid computing

Sanya Tangpongprasit *, Takahiro Katagiri, Kenji Kise, Hiroki Honda, Toshitsugu Yuba *

Graduate School of Information Systems, The University of Electro-Communications 1-5-1 Choufu-gaoka, Choufu-shi, Tokyo 182-8585, Japan

Received 20 February 2004; received in revised form 15 March 2005; accepted 26 March 2005
Available online 9 June 2005

Abstract

We present an alternative algorithm of fully decentralized resource discovery in Grid computing, which enables the sharing, selection, and aggregation of a wide variety of geographically distributed computational resources. Our algorithm is based on a simply unicast request transmission that can be easily implemented. The addition of a reservation algorithm is enable resource discovery mechanism to find more available matching resources. The deadline for resource discovery time is decided with time-to-live value. With our algorithm, the only one resource is automatically decided for any request if multiple available resources are found on forward path of resource discovery, resulting in no need to ask user to manually select the resource from a large list of available matching resources.

We evaluated the performance of our algorithms by comparing with first-found-first-served algorithm. The experiment results show that the percentages of request that can be supported by both algorithms are not different. However, it can improve the performance of either resource utilization or turnaround time, depending on how to select the resource. The

* Corresponding authors.
E-mail addresses: sanya@ntt.co.th (S. Tangpongprasit), katagiri@is.uec.ac.jp (T. Katagiri), kis@is.uec.ac.jp (K. Kise), honda@is.uec.ac.jp (H. Honda), yuba@is.uec.ac.jp (T. Yuba).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2005.03.005

algorithm that finds the available matching resource whose attributes are closest to the required attribute can improve the resource utilization, whereas another one that finds the available matching resource which has the highest performance can improve the turn-around time.

However, it is found that the performance of our algorithm relies on the density of resource in the network. Our algorithm seems to perform well only in the environment with enough resources, comparing with the density of requests in the network.

Keywords: Grid computing; Fully decentralized; Resource discovery; Resource reservation; TTL

1. Introduction

Grid computing is a distributed computing model where easy access to large geographical shared computing resources provided to large virtual organization; group of resources which are geographically apart while appearing to others to be a single [1]. These shared computing resources include computers, storage space, sensors, software application, and data. All are connected through the Internet and a middleware software layer provides basic services for security, monitoring, accessing information about components, etc. The idea of Grid computing is to analogous to electric power network (grid) where electric generators are geographically distributed, but the users are able to access electric power without bothering about the source of energy and its location [2]. Users in Grid computing system are allowed to harness the idle remote resources by using job scheduling.

Grid computing is one of many services which are provided in Grid technology. It provides secure services for executing application jobs on distributed computational resources individually or collectively. Some examples of Grid computing are NASA IPG [3], the World Wide Grid [4], and the NSF TeraGrid [5].

The basic service in Grid computing is resource discovery. With given a description of desired resources, a resource discovery mechanism finds the matching resources and returns the information of those resources to users. In large-scale Grid environments, resource discovery is made challenging by a potentially large number of resources and users and considerable heterogeneity in resource types and user requests. Resource discovery is further complicated by the dynamic variation of the number of shared resources in the system, shared resource characteristics such as availability and CPU load which vary with the time.

Today, Grid environments rely mainly on centralized architectures [6,7] which resource discovery is all done at only one central server. This method can perform the resource management easily. However, the centralized resource discovery architecture is not suitable for large-scale system. Grid computing in the future is expected to be large in scale and have lack of centralization as the same as today peer-to-peer file sharing systems [8].

Recently, there are alternative resource discovery mechanisms studied by many researchers. These works pay attention to a decentralized architecture instead of a centralized one. In such network, the central database or server has been removed

and all nodes act together to perform the resource management in the large-scale system. There is no central server where the information of all resources in the system is located. Resource management with a fully decentralized resource discovery mechanism is much more challenging. These related works rely on both of flat [9] and hierarchical [10] resource discovery architectures. A hierarchical architecture is quite complicated, whereas a flat architecture is easier to implement, and there is no doubt to its scalability. Hence, a flat decentralized architecture will be studied in this paper.

Until now, on this flat, fully decentralized architecture, most resource discovery mechanisms still let users manually select the resource from the large list of matching resources. Undoubtedly, users will prefer the available resource with the highest performance as it could be. From the system point of view, this disables other suitable users who should use the resource from accessing it. Hence, the mechanism should not allow users to select the resources manually. If the only one matching resource should be decided automatically and arbitrarily, it is convenient for network administrator to control system performance.

In this paper, we introduce and evaluate a new resource discovery mechanism on a flat, decentralized resource discovery architecture in Grid. The feature of this algorithm is to find an appropriate matching resource using time-to-live (TTL) value as the deadline of resource discovery time.

The remainder of this paper is organized as follows. In Section 2, we briefly review related work, and in Section 3 we describe decentralized resource discovery mechanisms in detail. In Section 4 we present a simulated Grid used in this work, and experimental results are shown in Section 5. Finally, we close the paper with conclusion and future work.

2. Related work

There have been relatively few papers published on the problem of large-scale resource discovery in Grid. To our knowledge, work by Iamnitchi and Foster [9] on decentralized resource discovery comes closest to our research. In their paper, they proposed a flat, decentralized, self-configuring architecture, where resources are located on network nodes. A user connects to a local node and the resource discovery is performed by forwarding the request node by node. The node either responds with the matching resource or forwards the request to another node. The request is forwarded until a resource is found or the initial time-to-live (TTL) value in the request message is decreased to zero.

A node can forward a request using one of four request forwarding algorithms which consist of:

(1) ‘random’
(2) ‘experience-based + random’
(3) ‘best-neighbor’
(4) ‘experience-based + best-neighbor’

For the result, 'experience-based + random' algorithm gives the best performance among four algorithms. All algorithms are based on first-found-first-served (FFFS) algorithm. In FFFS, the request is sent back immediately, if it finds any resource whose attributes match to the type described in the request.

In the real world of Grid computing, requests are often described as a set of desired attributes, not only the type of required resource (e.g., "a Linux machine with speed more than 500 MHz and 128 MB of available memory" [11]). We think that FFFS is insufficient for the resource discovery mechanism in Grid environments.

In this paper, we present a new algorithm differing from FFFS, which is based on 'experience-based + random'. With the addition of the reservation algorithm, more available matching resources can be found by using TTL value in the user's request message. Our mechanism automatically decides which matching resource should be informed back to the user.

Our framework relies on unicast request transmission. Multicast request transmission from users can perform resource discovery, although it is quite complicate to decide an algorithm. The transmission of a unicast request, however, can implement easily. Besides, it does not spread out the packets of requests on the network like multicast transmission which is often implemented in small networks.

In the next section we show how this resource discovery mechanism works with the reservation algorithm.

3. Resource discovery mechanism

3.1. Resource discovery model

In Grid environments there are a great deal of resources shared by all members in virtual organization. Resource discovery service acts as an intermediary between users and resource providers. Its function is to look for the resources whose attributes match to the required description in the user request in large network environment.

We assume that all members in a virtual organization have at least one server in order to store and provide access to local resource information. We call these servers 'nodes'. The information at each node contains information about its local shared resources and information about experience it learned from neighbor nodes. There are many resource attributes in Grid computing, e.g., type, speed, memory size, local load, and etc. 'Matching resources' mean the resources whose all attributes are equal to or more than all required attributes specified in the request message. Our resource discovery algorithm is presented in the following section.

3.2. Framework of resource discovery

The framework is divided into two main paths: the forward path and the backward path as shown in Fig. 1.

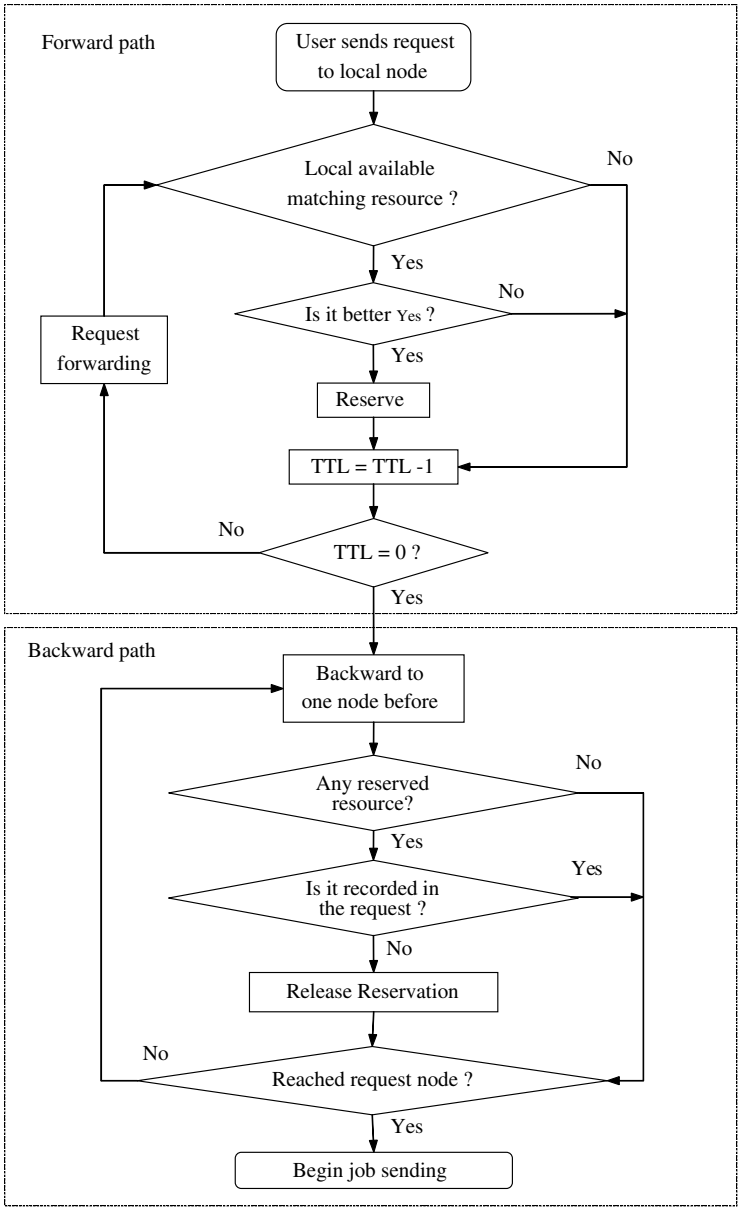

Fig. 1. Flow chart of TTL based resource discovery algorithm.

In the forward path, when users need to use the computing resource, they send their requests to their local node. Then, the node checks whether there is any local available matching resource whose attributes are better than one reserved before which is recorded in the request. If so, that resource is reserved. In the check and

reservation process, if there are more than one available matching resources in the same node, the node arbitrarily decides to reserve only one. The information of the last reserved resource is always added in the request packet. Then, the node forwards the request to one of its neighbors. The node decides to which node to forward the request using the 'experience-based + random' algorithm; i.e., the request is forwarded to the node that answered the same type of request previously. If there is no any record for that type of request, the request is forwarded to a randomly chosen neighbor node. This process is continued until TTL decreases to zero.

In the backward path, the reply message is sent back from the destination node along the forward path to release all unnecessary reserved resources until it reaches the source node which generated that request. Then it will be sent to its local user. The node records the experience by determining this reply message on the path between the target node, where the decided available resource is located, and the source node. After the resource discovery mechanism finishes (the reply message reaches the node initiating the request), the details of the job (the computational information in Grid environments) are sent directly from the user to the target node. Then, the target resource starts the job execution.

The execution time of each job can be approximately computed by the following equation:

$$\text{Execution time} = \frac{\text{Job size(CPU clocks)}}{\text{Resource performance(speed, Hz)}}$$

We assume that when the resource is reserved or occupied by the execution of any job, its status becomes unavailable. The resource cannot be reserved by another job until it becomes available again. However, the node where a reserved resource is located is not seized during the resource reservation. Node crossing may occur during multiple requests. Period of node occupation is limited during checking of an available matching resource at the node. It is short and then node usage does not significantly increase. For the clearer view of this algorithm, we show the example: The user requires resource whose OS = Linux, performance (speed) = 1 GHz, and memory size = 256 MB, where we assume that 'closest attribute' consideration is used. First, if at the first node, two matching resources (resource A: Linux, 1.2 GHz, 256 MB and resource B: Linux, 1.5 GHz, 256 MB), resource A is reserved since its performance is closer to the required description than resource B and the information of resource A is recorded in request packet. The request is kept forwarding and reserves more resource if it finds the idle resource C whose OS = Linux, performance (speed) = 1 GHz, and memory size = 256 MB. Now the information of resource C is recorded in the request packet instead of that of resource A. In the backward path, the reservation at resource A is released. After the user receives the reply packet back, the job is immediately sent to the node where resource C is located.

3.3. Four resource discovery algorithms

To evaluate this algorithm, we compare the performance with the FFFS algorithm. Instead of letting users select the resource manually from a large list of

resources, we present four alternative resource discovery algorithms which automatically try to find only one resource for each user request.

- ALG1. *TTL + closest attribute*: The request is forwarded until TTL value decreases to zero. The resource reservation algorithm is used to find the resources whose attributes are closest to the description in the request.
- ALG2. *TTL + highest performance*: The same as ALG1, but it looks for the resource which has the highest performance.
- ALG3. *FFFS + closest attribute*: The request is sent backward immediately if it finds the first available matching resource. In the case that there are multiple available matching resources found in the same node, the user is answered with the resource whose attributes are closest to the description in the request.
- ALG4. *FFFS + highest performance*: The same as ALG3, but in the case that there are multiple available matching resources found in the same node, the user is answered with the resource which has the highest performance.

The following section describes the simulated Grid used to evaluate these algorithms.

4. A simulated Grid

4.1. Network topology

In order to study large-scale Grid environments, we evaluate our resource discovery algorithms with a simulation which can be designed as a fully decentralized, flat large-scale architecture. We generated the starting network topology using Tiers network generator [12]. We assume that all the nodes are the members of virtual organization and connected through time. In the topology generation process, we also pay attention to the point that the network should be similar to the real network, avoiding unrealistically configurations, such as a star topology that is found only in small networks. In this study, we perform the experiments with an assumption similar to Iamnitchi and Foster's [9] with a 1000-node network.

Links connected in the network are all duplex links with transmission queue. If any packet tries to transmit via the busy link, it is added in a link queue, and initiates the transmission as soon as the link becomes idle. It is also assumed that there is no any link failure.

4.2. Resource distribution

The resource distribution in our experiment is closely related to real environments as possible. The nodes that share a large number of resources are fewer than the nodes that share only one or two resources. Hence, the distribution of resources on node is decided by geometric distribution, where the average number of resources is the constant set to five.

Table 1
Distribution functions of system parameters

Parameter	Distribution function
Number of resources on node	Geometric
Resource type (OS)	Uniform
Resource performance	Discrete poisson
Required resource type (OS)	Uniform
Required performance	Discrete poisson
Job size	Negative exponential
Request generation time	Poisson arrival time

In order to evaluate all algorithms, we give the following assumption for simplicity; first, resource attributes consist of only one type of resource and performance, second, each resource has adequate memory for the required memory size described in every request.

4.3. User requests

We assume that all users require only single resource, and matching resources are the ones whose attributes are equal or more than the required description in the request. Every node generates its own requests through simulation time. The random time of request generation is the Poisson process, the most common distribution of spike generation. Information in the request consists of TTL, source node address, traveling path of request message, description of required resource and lastly reserved resource.

All of the distribution functions used in the simulation study is shown in Table 1.

5. Experimental result

5.1. Evaluation parameters

We study the performance of previously mentioned four algorithms within the following three parameters.

(1) *System capacity*: The percentage of the requests which found matching resources.
(2) *Turn-around time*: The period counted from when the user's node initiates the request transmission until the result of execution is sent back to the user. This turn-around time includes resource discovery time, job execution time, and all transmission time. Job execution is, however, much larger than others. Most of turn-around time is job execution time.
(3) *Resource utilization*: The percentage of time that all resources in the system are occupied by the job execution.

Table 2
Average value of system parameters

Parameter	Average value
TTL	0–40
Request generation time	1–30 min
Number of resources on node	5
Resource type (OS)	10
Resource performance	1 GHz
Required performance	1 GHz
Job size	1 Tera CPU clocks

The values of parameters used for the following experiments are shown in Table 2.

5.2. Relationship to TTL value

This section shows three performances of all four algorithms when TTL value varies and the average generation time is set to 15. Note that the average job size is given 1 Tera CPU clocks, which is much larger than turn-around time.

From Fig. 2, when TTL value is small (less than 10), system capacity extremely increases and goes into the stable value when TTL value is large enough (more than 10) because of a uniform request generation in all network areas.

In large TTL period, ALG1, ALG3, and ALG4 have similar number of requests that found matching resources (around 85%). Although ALG1 tried to keep the high performance resources idle, the resource reservation algorithm affects in an indifferent value from ALG3 and ALG4. To find the resource with the highest performance, ALG2 makes one who has more need unable to use that resource, and then gives the smallest value (77.4%).

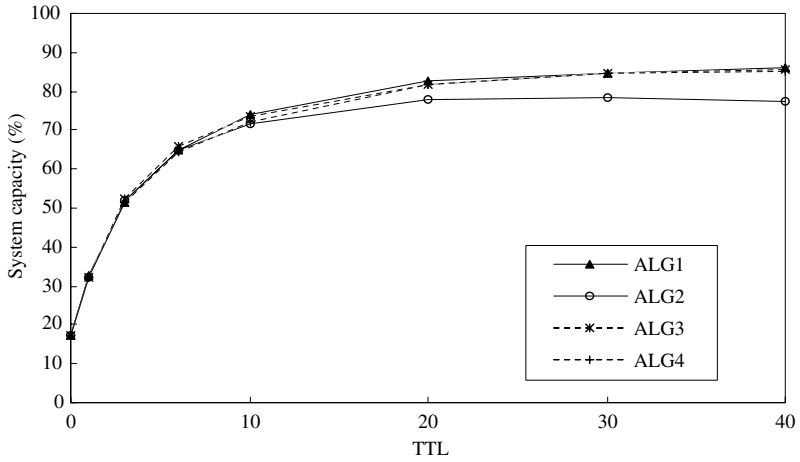

Fig. 2. System capacity vs. TTL.

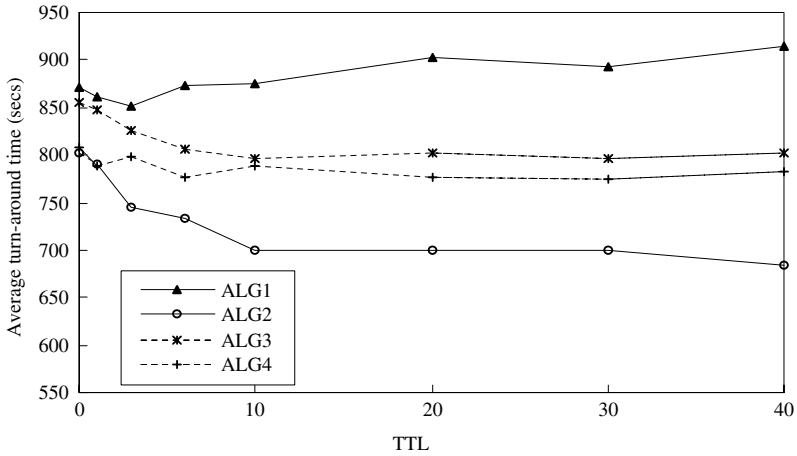

Fig. 3. Average turn-around time vs. TTL.

According to Fig. 3, ALG2 can finish resource discovery and user's job execution in the shortest time, and the next is ALG4, ALG3, and ALG1 as can be predicted from the characteristics of each algorithm. ALG1 executes the user job at the resource whose attributes are closest to the description in the request, resulting in the longest execution time, whereas ALG2 with the highest performance has the shortest execution time, and gives the shortest turn-around time (10% improvement compared with TTL = 0). When TTL value increases, the turn-around time of ALG1 tends to be longer, but that of ALG2 tends to be shorter because it has more chance to find the resource with higher performance. The higher the chosen resource performance is the shorter the turn-around time is. When TTL is more than ten, the increasing of TTL value does not improve much performance of turn-around time for all algorithms.

Considering resource utilization in Fig. 4, the sequence of algorithms is the same as when considering average turn-around time; that is ALG1, ALG3, ALG4, and ALG2, respectively, where ALG1 results in the maximum resource utilization and ALG2 results in the minimum resource utilization. This parameter has a direct relationship with average turn-around time that includes resource discovery time, job transmission and execution time. The resource discovery time and job transmission time is very small compared with the execution time in this study, so change in the execution time has a direct effect on turn-around time. Avoiding the use of high performance resources in ALG1 makes chosen resources occupied for a long job execution time and causes the highest resource utilization. On the other hand, ALG2 has the shortest execution time, resulting in the smallest resource utilization. The other algorithms, ALG3 and ALG4, have resource utilization performance between that of ALG1 and ALG2, where ALG3 has slightly higher resource utilization than ALG4.

It can be obvious from Fig. 2 that increasing of TTL value that is more than 10 can improve the performances of all algorithms only a little. Considering at

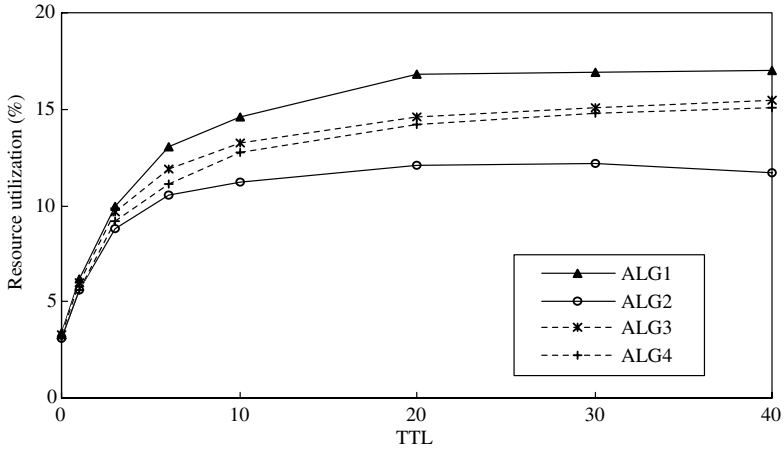

Fig. 4. Resource utilization vs. TTL.

TTL = 10, system capacity of all algorithms are not different, but the turnaround time and the resource utilization differs in each algorithm. ALG2 has the shortest execution time, but ALG1 has the best performance on resource utilization.

5.3. Relationship to average request generation time

In this section we set TTL value at 10, which is the most appropriate value as mentioned in previous section, and study the relationship between system capacity, turn-around time, and resource utilization to average request generation time.

From Figs. 5 and 7, it can be obvious that all algorithms have the same tendency. When the average request generation time increases (request generation rate

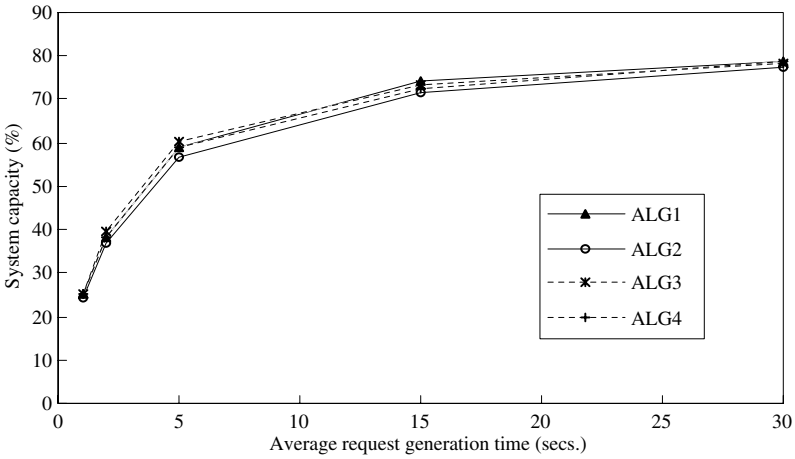

Fig. 5. System capacity vs. average request generation time.

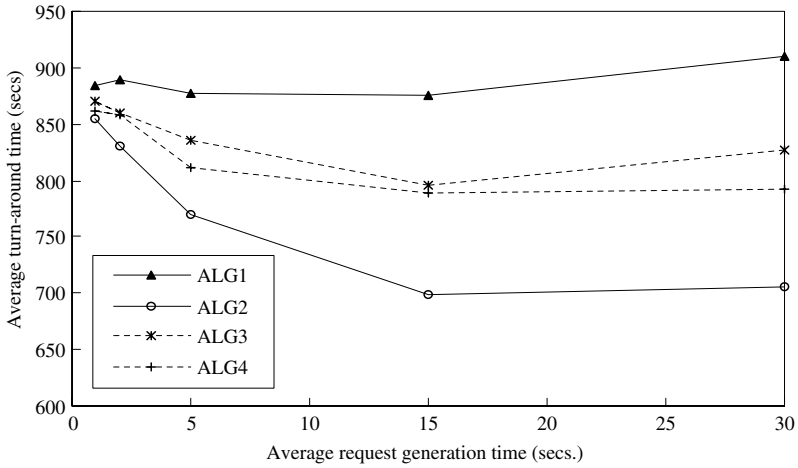

Fig. 6. Average turn-around time vs. average request generation time.

decreases), system capacity increases and the resource utilization value decreases. This is because the increment of request generation time leads to decrease the number of requests traveling in the network, resulting in less resource access competition. Hence, there is more probability for each request to find available matching resources. With few requests traveling in the network, the occupied time of all resources becomes decreased and results in less resource utilization.

The turn-around time of all algorithms has different tends as shown in Fig. 6. All algorithms have the similar turn-around time in high traffic request condition. ALG1 has the similar turn-around time (around 880 s) throughout the range of request

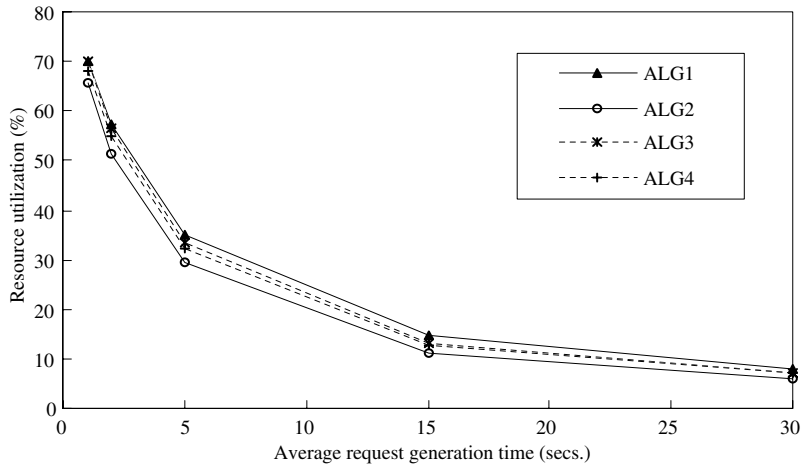

Fig. 7. Resource utilization vs. average request generation time.

generation time. The turn-around time of ALG2 decreases when average request generation time increases in the first-half period. In the last-half period, the value of ALG2 does not much vary because although the request traffic is lower, the performance to find more matching resources is limited by fixed TTL value.

6. Conclusion and future work

In this paper, we have proposed new resource discovery algorithms with TTL-based reservation and unicast request forwarding algorithms on a flat, fully decentralized architecture in Grid computing. In order to evaluate our algorithms, we created a simple large-scale network of Grid computing and decided all the necessary parameters and their distribution patterns by considering real environments. Our reservation algorithm is implemented on the forward path of request to find more available matching resources, then on the backward path it releases all reserved resources except one that will be used to execute the job specified by the user. Our results show that our reservation algorithm improves the performance of resource discovery on the first-found-first-served (FFFS) algorithm which is implemented in Iamnitchi and Foster's work [9]. ALG1 (TTL + closest attribute) attempts to make use of resources in the system as much as possible and keep high performance resource idle and result in higher resource utilization, whereas ALG2 (TTL + highest performance) attempts to find the resource with the highest performance and can improve turn-around time. Our algorithm needs more hops to obtain a better solution than FFFS approach. However, the search space does not increase in geometrical progression during its finding process, in which case, network traffic will tremendously increase. Our algorithm forwards a request to one way by unicast request transmission in step-by-step manner until TTL. This means the finding process does not cause much traffic in the network.

However, there are tradeoffs between resource utilization and the turn-around time. When we decide which algorithm to adopt, it is important to think whether we would like to support the need of the network administrator-to maximize resource utilization (ALG1) or the user's-to minimize the turnaround time (ALG2).

Our algorithm shows satisfied performance in the Grid computing environments with enough resources, comparing with the density of requests in the network. It can select the most appropriate resource from many matching resources found in forward path of resource discovery. On the other hand, in the high request traffic environment, the performance is not much different from FFFS algorithm.

We can apply this algorithm by deciding the appropriate TTL value; i.e., not too small and too large value. Too small TTL value results in bad performance and too large one can improve the performance only a little. The appropriate TTL value depends on all the network environments; e.g., density of the resources in the network, resource distribution, and users request traffic.

We have done the simulation on a simple assumption and with few conditions. In order to study the resource discovery in the Grid computing system, the extension of our work can be done on several methods written below.

(1) *The effect of network size*: The number of nodes can be increased in order to study the effect of size of the system to the appropriate TTL value of the Grid computing system in the large-scale environment which may has size in order of ten thousands of nodes.
(2) *Cost and budget of job execution*: Our algorithms in the work consider mainly on the attributes of matching resources. In the practical way, we cannot neglect the cost and the budget of job execution. In Buyya's work [13], these factors are studied in the centralized resource scheduling architecture. The cost of job execution is set by shared resource's owner, whereas the budget is decided by job's owner. Certainly, the negotiation between job's owner and shared resource's owner becomes to play the important. The final decision must satisfy the needs of both sides. In a fully decentralized resource discovery architecture, which has no any central server, how to perform the resource discovery efficiently and make the best resource scheduling is very challenging.
(3) *Advanced reservation*: In this paper, the resource can be reserved only when they are available. If users can reserve some matching resources in advance although those resources are not in idle status, the number of discarded requests will be decreased. Moreover, the resources can continue execution as soon as the previous job execution finished, resulting in higher resource utilization.
(4) *Parallel job execution*: To conduct the high performance computing in Grid environments, jobs from users are divided to many parts and distributed to execute separately like parallel processing in order to lessen the job execution time. In a fully decentralized architecture, it becomes much more complicate to achieve this goal. How to predict the performance of the resources that are geographically distributed in both location and information will become a challenging problem in this case.

References

[1] I. Foster, C. Kesselman, S. Teucke, The anatomy of the Grid: enabling scalable virtual organizations, International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.
[2] Available from: <http://www.gridcomputing.com/>.
[3] W. Johnston, D. Gannon, B. Nitzberg, Grids as production computing environments: The engineering aspects of NASA's information power grid, in: Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing, (Redondo Beach, CA, 1999).
[4] R. Buyya, The World-Wide Grid. Available from: <http://www.buyya.com/ecogrid/wwg/>.
[5] NSF Tera-Grid. Available from: <http://www.teraGrid.org/>.
[6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information service for distributed resource sharing, in: Proceedings of 10th IEEE International Symposium on High Performance Distributed Computing, IEEE Press, NY, 2001, pp. 181–184.
[7] M. Livny, R. Raman, High-throughput resource management, in: I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., CA, 1998 (Chapter 13).
[8] A. Iamnitchi, I. Foster, D. Nurmi, A peer-to-peer approach to resource discovery in Grid environments, in: Proceedings of High Performance Distributed Computing, IEEE, Edinbourgh, UK, 2002.

[9] A. Iamnitchi, I. Foster, On fully decentralized resource discovery in Grid environments, in: Proceedings of International Workshop on Grid Computing (Denver, Colorado, 2001).
[10] Z. Juhasz, A. Andics, S. Pota, Towards a robust and fault-tolerant multicast discovery architecture for global computing grids, in: Proceedings of 4th Austrian-Hungarian Workshop on Distributed and Parallel Systems, Linz, Austria, 2002, pp. 74–81.
[11] R. Raman, M. Livny, M. Solomon, Matchmaking: a extensible framework for distributed resource management, Cluster Computing: The Journal of Networks, Software Tools and Applications 2 (2) (1999) 129–138.
[12] M. Doar, A better model for generating test networks, in: Proceedings of IEEE Global Telecommunications Conference, GLOBECOM'96, London, UK, 1996, pp. 83–96.
[13] R. Buyya, J. Giddy, D. Abramson, An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications, in: Proceedings of 2nd International Workshop on Active Middleware Services, Kluwer Academic Press, Pittsburgh, USA, 2000.