International Journal of Grid and Distributed Computing
Vol.7, No.2 (2014), pp.21-32
http://dx.doi.org/10.14257/ ijgdc.2014.7.2.03

A New Method for Resource Discovery in a Grid Resource Service Chain

Masoud Barati[1] and Rahim Alizadeh[2]

*[1]Department of Computer science
Islamic Azad University- Kangavar branch
Kangavar, Iran, emsbarati@yahoo.com
[2]Department of Mathematics, Shahed University
P.O.Box:18151-159, Tehran, Iran, alizadeh@shahed.ac.ir*

Abstract

A new decentralized method is proposed to resource discovery in Grid computing systems. This agent-based method supports semantic description and is flexible for resource identification and allocation. It allows each of the individual resource agents to interact only with its neighbor agents semantically and form a resource chain for a specific task dynamically. We discuss about the success of probability of our method under different task loading rates and compare it with some other known Algorithms. The experiments show the algorithm could flexibly and dynamically discover resources.

Keywords: *Grid computing, Recourse agent, Recourse discovery*

1. Introduction

One of the distributed computing systems is Grid computing. Grid computing makes it possible for sharing in heterogeneous computing resources such as saving tools, processors, data and applications on an unprecedented scale, among an infinite number of geographically distributed groups [1]. Researchers want to seamlessly access and integrate Grid resources for their works. Grids offer a way of using the information technology resources optimally inside an organization [2]. They also provide a means for offering information technology as a utility for commercial and noncommercial clients, with those clients paying only for what they use, as with electricity or water.

Besides some important topics such as sharing, allocation and integration of recourses, the issue of resource discovery has a special importance in Grid computing systems. About recourse discovery in these systems with such characteristics, different methods such as Peer-to-Peer Approache in [3, 4], Routing Transferring Model-Based Approach in [5], Parameter-Based Approaches in [6, 7], Quality of Service (QoS) Approaches in [8, 9] and Ontology Description-Based Approaches in [10-13] have been identified. The challenge is to devise highly distributed discovery techniques that are fault tolerant and highly scalable. Among the suggested methods, those methods without remaining due to the negotiation and decentralized recourse discovery have a main importance. In these methods each recourse has an agent that is called *resource agent*. To complete a specific task, resources agents interact with their neighbors based on local knowledge and form a chain of resources dynamically.

In Ontology Description-Based Approach, Instead of exact syntax matching, ontology-based matchmaker performs semantic matching using terms defined in those ontologies. Harth *et al.*, proposed an ontology based matchmaker service for dynamic resource discovery and description [11]. The presented method used separate ontologies to declaratively describe resources and job requests. They adopted the Globus Toolkit for the Grid service development and exploited the monitoring and discovery service in the Grid infrastructure to dynamically discover and update resource information. But, it is still not sure if it works well in the large scale distributed network. Authors in [12] proposed semantic grid architecture for discovering resources semantically. It was built on top of Gridbus broker [13], but its improvement mainly consists in the semantic description of the Grid resources rather than concerning global scale search and discovery. Gorgojo *et al.*, proposed a semantic approach for the learning web service discovery to support collaborative learning activities within a particular educational setting [14]. Liangxiu and Dave introduced a flexible approach to discover resources semantically by using a decentralized fashion [10]. In this approach resource agents are considered as resource carriers and based on local knowledge and individual agents manage resources chain connectivity and dynamically form resources chain to complete tasks or jobs.

In this article using the suggested method in [10] and adding some other information to each of the resource agents, we represent a new method that at first constructs an appropriate resources chain and then removing removable superfluous nodes, reach an optimal resource chain. Also we discuss about success of probability of our method under different task loading rates and compare it with some other known methods.

To structure the grid, we use a small-world overlay graph. Small world graphs have been found to be important in many different domains. Our overlay construction is the dynamic version of a well-known algorithm for generating small world graphs which are a family of graphs with several nice properties, most notably low diameter and low average degree. We refer the interested reader to [15, 16] for details.

The rest of this paper is constructed as follow. In Section 2 some definitions and primary concepts will be established. Our proposed algorithm is presented in Section 3. The experiment results are demonstrated in section 4 and finally In Section 5, conclusions and future works are highlighted.

2. Definitions and Primary Concepts

Before representing our algorithm, it is necessary to be familiar with some definitions and primary concepts.

2.1. Connected Components

A graph is connected if for every pair of vertices, there is a path connecting them. A graph that is not connected can be divided into connected components (disjoint connected subgraphs). For example, the following graph is made of two connected components.

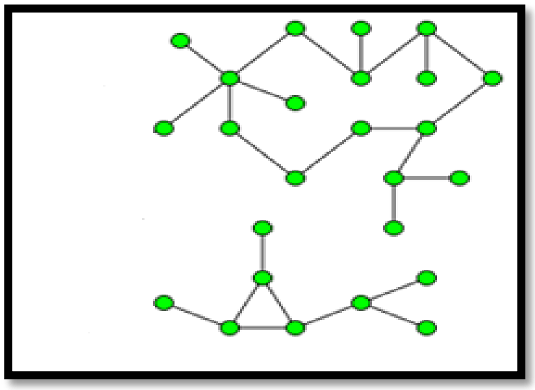

Figure 2.1. A Graph with Two Connected Components

2.2. Ontology and Semantic Similarity Function

Ontology is a formal structure that contains information about semantic description data. This structure contains a set of concepts and their connections. It can be used in the recovery of information that deals with users inquiries [17, 18]. Ontology O is defined as

$$O = \{C, \leq_C, R, \leq_R, A\},$$

Where C is a set of concepts, R is a set of relationships, $\leq_C$ is a partial order on C and $\leq_R$ is a partial order on R. Also, A is defined as a set of axioms [19, 20].

Semantic similarity function is used for computing similarity between two concepts. Similarity between concepts shows degree of their common. Similarity function is defined as sim(x, y): $C*C \rightarrow [0, 1]$. The output of this function is a real number in interval [0, 1] that shows the degree of similarity between two concepts x and y. The zero output means lack of sharing and output one shows complete similarity between two concepts [21-27].

2.3. Model Description

In this article like the represented model in [10] we graph the model by network that each of its nodes is a resource agent and each of these agents carry a specific type of resource. If k types of various resources exist in the system we show this k types with symbols of R1,…,Rk. It is necessary to mention that resource agents have competence to find their neighboring nodes to form resource chains needed for the tasks. A task is identified by a list of the resources which is needed for doing it. Tasks are distributed with a certain rate μ in the network and each of them needs m different types of resources ($m \leq k$). If we suppose p task distributes to the network, it could be shown by a $p \times m$ matrix that each row shows a task and each column shows a required type of resource for it. Like the model in [10] each node (resource agent) in the network has three states of inactive, committed and active. The inactive state shows the node is free and is not working on any task. The committed state means that the node is working on a task but the resource chain still is not complete and the active state means the node is working on a task and the resource chain is completed. Here Like [10], semantic similarity is represented by an $n \times n$ matrix which its entries specify the rate of the similarity between the required resource type and the resource type of a resource agent. In this article we assume that semantic similarity threshold is one thus, semantic similarity matrix elements are zero and one that one means full similarity and zero means no similarity.

2.4. Resource Agent

Every resource agent has an individual number and is carrier of specific type of a resource. Also each agent is aware of its neighbor's resource type and the task that is working on. Here for operate our algorithm, it is necessary to equip each resource agent to below information:

1. **Resources List:** Here to form the chain, the first randomly selected agent should supply a list of the resource types (except its resource type) necessary for the task and send it to the free neighbors that their resource type exist in the list. After receiving the list, each resource agent deletes its resource type from the list and repeats this process. Here we nominate this list as *resources list*.

2. **Committed list:** When a resource agent deletes its resource type from the received resources list, the obtained list may be empty. In this situation or in the condition that there is no other free resource agent in its neighborhood, it creates a list which we call *committed list*. Then the resource agent inserts its resource type and its node number in it and sends the committed list to the neighbor that had received the resources list from it. Every agent after receiving the committed lists from all of the neighbors that had sent the resources list to them, joins received the committed lists, adds its resource type and node number to it and sends the obtained committed list to the neighbor that had received the resources list from it. Finally the first selected agent prepares the main committed list. In this step the primary resources chain has been constructed but it is possible that there are some superfluous resources in the chain.

3. **Connectivity field:** As we said after constitution of the resources chain, it is possible that there are some superfluous resources in the chain. A resource type in the chain is superfluous, if it has repeated type in the main committed list. A superfluous node that its deletion causes no chain disconnectivity is called *removable*, otherwise it is called *irremovable*. Each node has a field which we call it *connectivity field*. At first the connectivity field of each node is empty. We will explain more about these concepts in the next section.

3. Proposed Algorithm

3.1. Constructing Primary Resources Chain

A task is identified by a list of the resources which are needed to doing it (task requires). After loading a task, a free resource agent in the network is selected randomly and if its type exists in the loaded task (list), it deletes own type from the task and puts others to a new list called resource list. We call this agent the *admin node*. Then this agent changes its state to committed state and sends this list to other neighbors that are free and their type is in the list. By receiving resource list, the neighbors of the resource agent change their states to committed state and delete their resource type from the received resource list and repeat this process. When a resource list of a resource agent becomes empty or it does not have another neighbor that is free and its resource type be existed in the resource list, it puts its resource type and its node number in a new list called committed list and sends it to the neighbor which had received the resource list from it. After receiving the committed lists from all of the neighbors that had been sent the resources list to them, joins them together, adds its resource type and node number to it and sends the formed committed list to the neighbor that has received the resource list from it. The process of sending committed lists repeats till the admin node receives the committed lists from all of the neighbors that had sent the resources list to them. Then it joins committed lists together and adds its resource type to it and makes the *main committed list*. If the obtained committed list dose not supplies the required resources for the loaded task, it sends an inactive order to the committed nodes in the constructed chain and set itself to the inactive state. Otherwise, A resources chain is constructed which we call it *primary chain*. Algorithm 3.1 displays a pseudo code which describes forming of the primary resources chain. Note that it is possible that the primary chain contains some superfluous resources. We discuss about removing these nodes in the next subsection.

Now for understanding our Algorithm, we get an example of a network that contains 16 nodes labeled from 1 to 16 with 4 different types of resources (Figure 3.2a). We suppose that the task T = (1, 2, 3, 4) loads in the network and node 1 is the admin node (Figure 3.2b). This node is of the type 4 which is in the list (1, 2, 3, 4) and in the neighboring of it, the nods 3 and 14 can supply another needed resource types for T, so it goes to committed state (committed state is shown with pink color in Figure 3.2). Then the node 1 deletes its resource type from (1, 2, 3, 4) and sends the new resources list (1, 2, 3) to the nodes 3 and 14. The nodes 6 and 8 in the neighboring of the node 14 are free and their types are in the list (1, 2), so the node 14 goes to committed state and after deleting its resource type form the list (1, 2, 3) sends (1,2) to the nodes 6 and 8. But node 3 doesn’t have any neighbor of the type 2 or 3, thus it forms a committed list, inserts its number and type in it and sends the list ((3,1)) to the node 1. On the other hand since the nodes 6 and 8 don't have any neighbor of the type 2 and 1 respectively; they create new committed lists, insert their numbers and resources types into it, and send it to the node 14. Then the node 14 receives committed lists ((6,1)) and ((8,2)) from the nodes 6 and 8 respectively, joins them to each other, adds its type and node number to it and sends ((14,3), (6,1), (8,2)) as the committed list to node 1. After receiving committed lists ((3,1)) and ((14,3), (6,1), (8,2)), the node 1 creates the main committed list ((1,4), (3,1), (14,3), (6,1), (8,2)) or in summary (4,1,3,1,2). This list contains all of the needed types for T and hence the primary chain could be constructed (Figure 3.2c). Note that the type 1 in the main committed list is repeated.

```
ALGORITHM  proposed for forming Primary resources chain
1:   Input: Task  needs
2:   Output: Primary resources chain
3:   Assumptions
4:   Task T loads in the network
5:   Committed list (cmtlist) is empty set for all inactive nodes in the network
6:   A node is selected randomly and it receives the task needs (Rlist) for T
7:    \\ To construct the primary chain all nodes run the following steps individually
8:   Begin  Algorithm
9:     If   Rlist is received Then
10:      If  there is a neighbor that its resource type exist in the Rlist and its state is inactive
     Then
11:         Set your state to committed state
12:         Delete your resource type from the Rlist
13:         Send the Rlist to all neighbors that their types exist in the Rlist and their state are
     inactive
14:       Else
15:         Add your resource type and node number to cmtlist and send it to the node which you
            have received Rlist from it.

16:      END If
17:    END If
18:    If all cmtlists are received  from the neighbors that Rlist had been sent to them Then
19:       Create a cmtlist by joining received cmtlists and add your resource to it
20:       Send cmtlist to the neighboring node that you had been received the Rlist from it
21:   END If
22:  End  Algorithm
```

Algorithm 3.1. Pseudo Code which Describes Forming of the Primary Resources Chain

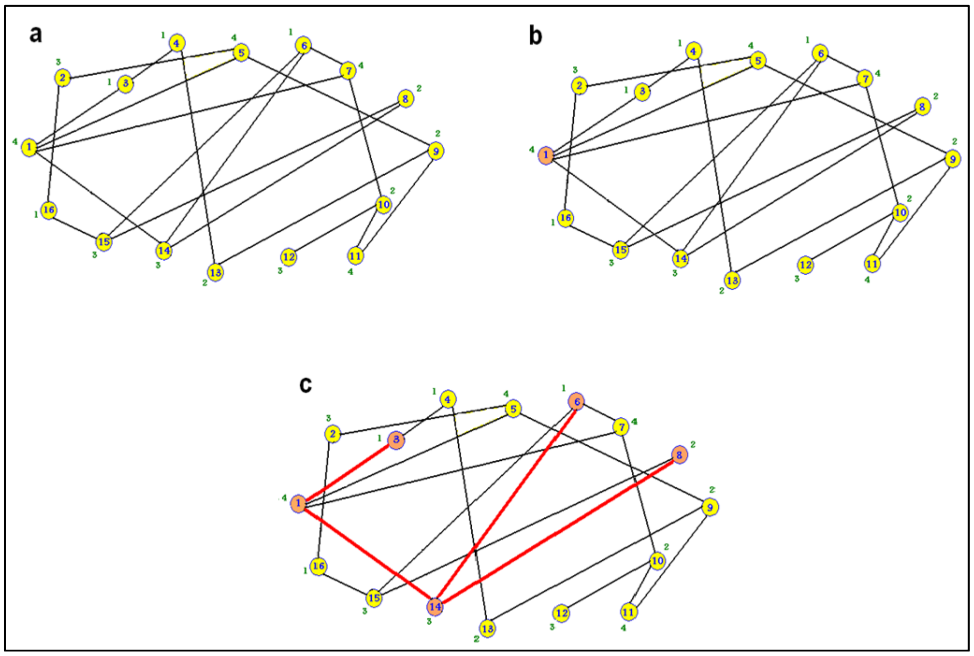

Figure 3.2. Constructing the Primary Resources Chain

3.2. Removing Superfluous Nodes

In this part we want to remove removable superfluous nodes from the primary resources chain and reach an optimal resource chain. Before explaining how admin node removes removable superfluous nodes in details, let us look at the Algorithm 3.3, which describes the process of forming the optimal resources chain from the primary resources chain.

ALGORITHM proposed for pruning removable superfluous nodes of the primitive resources chain

```
 1:  Input: primary resources chain
 2:  Output: optimal resources chain
 3:  Assumptions
 4:  cmtlist is the main committed list
 5:  \\ the admin node performs the following steps to remove the removable nodes
 6:  Begin  Algorithm
 7:  While  there is a removable superfluous node of a type R in the cmtlist
 8:           If all of the repetitive elements of type R are removable Then
 9:             keep one of the repetitive elements and remove others from cmtlist
10:           Else
11:             remove all removable elements of type R from cmtlist
12:           End If
13:        End While
14:      End If
15:  End  Algorithm
```

Algorithm 3.3. Pseudo Code which Describes Forming of the Optimal Resources Chain

For an example consider Fig 3.4a. Here the task T = (1, 2, 3, 4, 5) has been loaded in the network, the node 31 has been selected firstly (admin node) and a primary chain has been constructed. As we explained in the previous section, the main committed list is equal to (1, 4, 3, 2, 3, 4, 5). As we see in Figure 3.4b, according to Algorithm 3.3, the admin node (node 31) deletes the nodes 35 and 12 which are removable. That means the state of the nodes 35 and 12 changes to inactive. Therefore a chain with nodes 23,31,11,15 and 37 will form for the loaded task (Figure 3.4c).

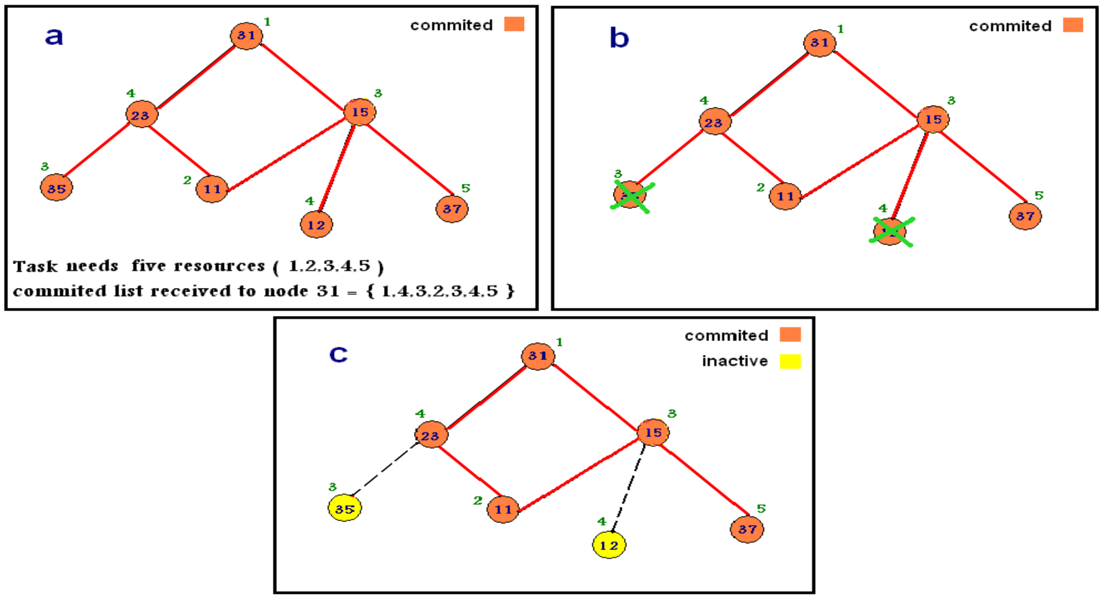
a
commited
31
1
4
23
3
15
3
35
2
11
4
12
5
37
Task needs five resources (1.2.3.4.5)
commited list received to node 31 = { 1.4.3.2.3.4.5 }
b
commited
c
commited
inactive

Figure 3.4. A Schema of the Algorithm 3.3 for Constructing the Optimal Resources Chain

Now we explain the process of identifying the removable nodes in the primary chain. For identifying a superfluous node is removable or not, the admin node sends a "check connectivity" message to it. After receiving this message, the superfluous node inserts its node number to its connectivity field and sends the connectivity field together with the received message to one the its neighbors in the primary chain. Then each of the nodes in the primary chain performs the following steps individually:

1. After receiving connectivity field together with "check connectivity" message from one of the neighbors in the primary chain, if your connectivity field is empty, put the received connectivity field in your connectivity field and send it together with the received message to other neighbors in the chain. If your connectivity field is not empty send "End of connectivity checking" message to the neighbor that has received "check connectivity" message from it.

2. If you receive "End of connectivity checking" message from all of the neighbors that had sent "check connectivity" message to them, send "End of connectivity checking" message to the neighbor that had received "check connectivity" message from it.

After receiving the message of "end of connectivity checking", the superfluous node considers the connectivity field of whole neighbors that are in the chain. If all of them are equal to its connectivity field, it sends a message that includes "removable" to admin node and changes its state to inactive. Otherwise a message that includes "irremovable" is sent to admin node.

4. The Simulation Results

In this section we will concentrate on simulation results of the presented Algorithm. The presented algorithm in the previous section is applicable on every network structure. Here the network is displayed by a small world network graph with 128 nods 16 resource types which is constructed based on available methods in [15,16]. We assume that each of the tasks needs 16 resource types. In each experiment we run 10 simulation with different loading rates of task generation that is between interval of [0.05, 1] and at each rate point, we publish 1000 tasks within the time step.

We study the relation of the task success probability under task complexity and different task loading rate along with network scale with consideration of semantic similarity threshold one.

We define task success probability as the ratio of number of tasks completion and total tasks published.

4.1. The Relationship between Task Success Probability under Different Levels Task Complexity and Different Task Load Rates

We consider the task complexity as the ratio of required resources for the task and the variety of resources are in the network. Here we analyze the task in different lengths of 4, 8, 12, 16 and evaluate the success probability in similarity threshold of one. Figures 4.1(a) - 4.1(d) display the results of these experiments regarding the new method and the method proposed in [10].

As we see the success probability of the new method is greater than the previous method and in both methods with increasing task complexity, success probability will decrease. Also, if loading rate increases, success probability will decrease.

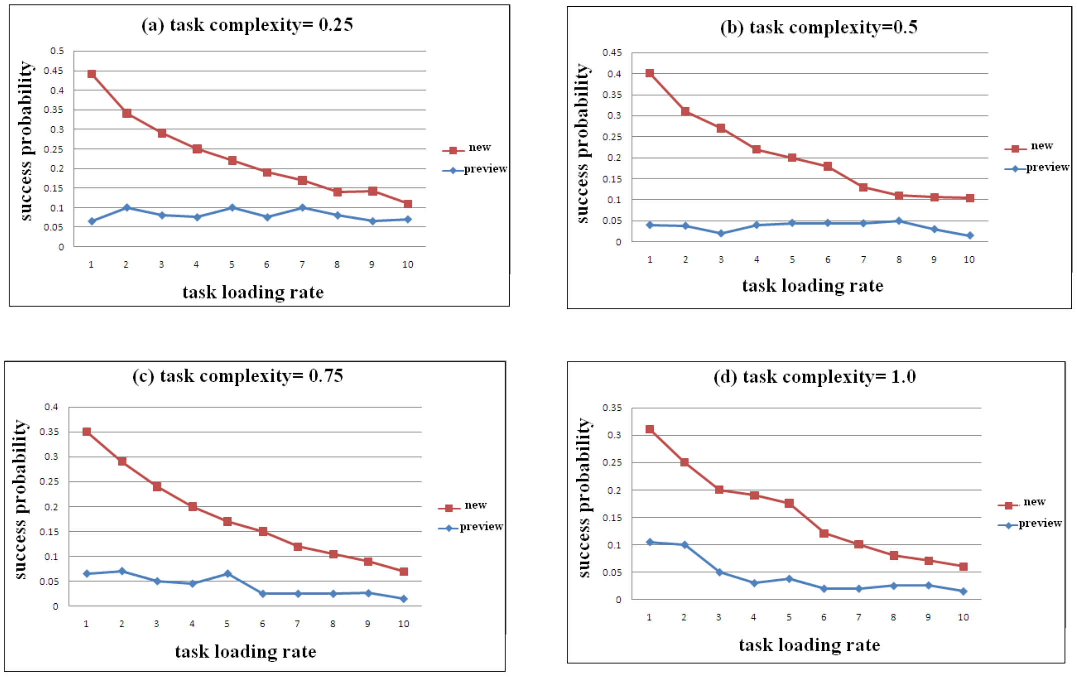

Figure 4.1. The Relationship between Success Probability under Different Task Complexity

4.2. The Relationship between Success Probability and Increasing the Number of Nodes

Here we model networks with 64, 128 and 256 nodes that contains 16 various resource types that are uniformly distributed and in every time step we send 1000 tasks to them.

We consider success probability of task regarding these different network scales. Here task complexity has the fixed amount of 0.5. As we observe, with increasing size of the network the new method is work better that the previous method and in both methods when network scale increases, success probability will increase (Figure 4.2).

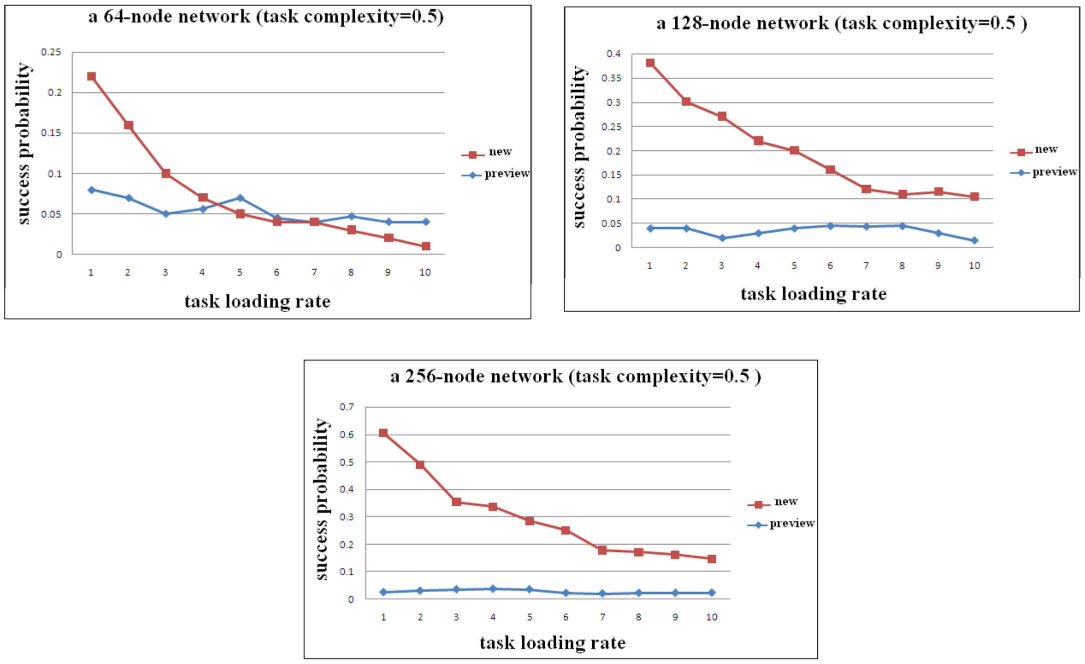

Figure 4.2. The Relationship between Success Probability under Different Network Scale

5. Conclusions and Future Works

It pointed to a decentralized method that is upon agent, supports the semantic description and provides a flexible mechanism for resource allocation and nomination. Adding some extra information to each of the resources agent, it was tried to make a suitable chain of resources firstly and then deleting extra nodes from the constructed chain, we obtained the desired chain. Also we compared our algorithm with some other known Algorithms. The experiments show the algorithm could flexibly and dynamically discover resources.

The future work will focus on increasing security of the proposed method and also fault tolerance while failing each of the resource agents after forming an optimal resources chain.

References

[1] F. and C. Kesselman, “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann, **(1999)**.
[2] B. Jacobi, M. Brown, K. Fukui and N. Trivedi, “Introduction to Grid computing”, IBM Redbook, **(2005)**.
[3] A. Iamnitchi and I. Foster, “On Fully Decentralized Resource Discovery in Grid Environments”, IEEE International Workshop on Grid Computing, Denver, CO, **(2001)**.
[4] A. Iamnitchi, I. Foster and D. C. Nurmi, “A Peer-to-Peer Approach to Resource Discovery in Grid Environments”, Proceeding of the 11th Symposium on High Performance Distributed Computing, Edinburgh, UK, **(2002)**.
[5] W. Hoschek, “A Unified Peer-to-Peer Database Framework for Scalable Service and Resource Discovery”, Proceeding of Third International Workshop on Grid Computing: GRID 2002, Baltimore, MD, Springer, pp. 126-144.

[6] W. Li, Z. Xu, F. Dong and J. Zhang, "Grid Resource Discovery Based on a Routing-Transferring Model", Proceeding of Third International Workshop on Grid Computing: GRID 2002, Baltimore, MD, Springer, pp. 145-156.
[7] M. Maheswaran and K. Krauter, "A Parameter-based Approach to resource discovery in Grid computing systems", 1st IEEE/ACM International Workshop on Grid Computing, **(2000)**.
[8] M. Harchol-Balter, T. Leighton and D. Lewin, "Resource Discovery in Distributed Networks", 18th ACM Symposium on Principles of Distributed Computing, **(1999)**.
[9] Y. Huang and N. Venkatasubramanian, "QoS-based Resource Discovery in Intermittently Available Environments", Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02) 1082-8907/02 IEEE.
[10] H. Liangxiu and B. Dave, "Semantic-supported and agent-based decentralized grid resource discovery", Future Gener. Comput. Syst., vol. 24, **(2008)**, pp. 806-812.
[11] A. Harth, Y. He, H. Tangmunarunkit, S. Decker and C. Kesselman, "A semantic matchmaker service on the grid, **(2004)**, pp. 326–327.
[12] T. S. Somasundaram, R. A. Balachandar, V. Kandasamy, R. Buyya, R. Raman, N. Mohanram and S. Varun, "Semantic-based grid resource discovery and its integration with the grid service broker", Technical Report, GRIDS-TR-2006-10, Grid computing and Distributed Systems Laboratory, The University of Melbourne, Australia, **(2006)**.
[13] S. Venugopal, R. Buyya and L. Winton, "A grid service broker for scheduling distributed data-oriented applications on global grids", Proceedings of the 2nd International Workshop on Middleware for Grid computing, Co-located with Middleware 2004, Toronto, Canada, ACM Press, USA, **(2004)**.
[14] G. Vega-Gorgojo, M. L. Bote-Lorenzo, E. G'omez-S'anchez, Y. A. Dimitriadis and J. I. Asensio-P'erez, "A semantic approach to discovering learning services in grid-based collaborative systems", Future Generation Computer Systems, **(2006)**, vol. 22, pp. 709-719.
[15] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks", Nature, **(1998)**, vol. 393, pp. 440-442.
[16] D. J. Watts, "The dynamics of Networks between order and randomness", Small Worlds, Princeton University Press, Princeton, New Jersey, **(1999)**.
[17] H. Stuckenschmidt, "Ontology-based information sharing in weekly structured environments", Ph.D. thesis, AI Department, Vrije University Amsterdam, **(2002)**.
[18] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing", KSL-93-04, Knowledge Systems Laboratory, Stanford University, **(1993)**.
[19] R. Rada, H. Mili, E. Bicknell and M. Blettner, "Development and application of a metric on semantic nets, IEEE Transactions on Systems, Man, and Cybernetics, vol. 19, **(1989)**, pp. 17-30.
[20] T. Andreasen, H. Bulskov and R. Knappe, "From ontology over similarity to query evaluation", R. Bernardi and M. Moortgat (Eds.): 2nd CoLogNET-ElsNET Symposium - Questions and Answers: Theoretical and Applied Perspectives, Amsterdam, Holland, **(2003)**, pp. 39-50.
[21] O. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity and natural language", Journal of Artificial Intelligence Research, vol. 11, **(1999)**, pp. 95–130.
[22] R. Richardson, A. Smeaton and J. Murphy, "Using WordNet as a knowledge base for measuring semantic similarity between words", Tech. Report Working paper CA-1294, School of Computer Applications, Dublin City University, Dublin, Ireland, **(1994)**.
[23] M. A. Rodriguez and M. J. Egenhofer, "Determining semantic similarity among entity classes from different ontologies", IEEE Transactions on Knowledge and Data Engineering, vol. 15, **(2003)**, pp. 442-456.
[24] N. Seco, T. Veale and J. Hayes, "An intrinsic information content metric for semantic similarity in WordNet", Tech. report, University College Dublin, Ireland, **(2004)**.
[25] A. Tversky, "Features of similarity", Psycological Review, **(1997)**, vol. 84, pp. 327-352.
[26] M. Richter, "Classification and learning of similarity measures", Technical Report SR-92-18, **(1992)**.
[27] R. Knappe, H. Bulskov and T. Andreasen, "Similarity graphs", LNAI 2871, N. Zhong, Z. W. Ras, S. Tsumoto, E. Suzuki (Eds.): 14th International Symposium on Methodologies for Intelligent Systems, ISMIS 2003, Maebashi, Japan, **(2003)**, pp. 668-672.