A New Approach to Resource Discovery in Grid Computing

Leyli Mohammad Khanli[1],
Saeed Kargar[2] and Ali Kazemi Niari[2]
[1]*C.S. Dept., University of Tabriz*
[2]*Islamic Azad University, Tabriz Branch*
Iran

1. Introduction

The grid computing systems are one of great developments in the field of engineering and computer science and provide a clear future in the global use of various optimal distributed resources (hardware and software). Therefore with expanding grid systems and the importance of finding suitable resources for users (Foster & Kesselman, 2003), while saving time and space, resource discovery algorithms are very important. If one algorithm with less traffic and without reference to unnecessary nodes in a shorter time, can find the appropriate resource for users, it will significantly increase the efficiency of the system.

There are many approaches to resource discovery, such as flooding-based and random-based. These approaches decrease the system efficiency, because the users' requests pass through many unnecessary paths and create additional traffic. Therefore it is not suitable for a grid environment with numerous nodes.

Another approach is resource discovery tree using bitmap (Chang & Hu, 2010). This method decreases some disadvantages of previous methods such as unnecessary traffic and heavy load, and furthermore the cost of update is low. But in this method, users' requests are also sent to unnecessary nodes, so in a grid environment with numerous nodes and requests, the reference to unnecessary nodes will create additional and heavy traffic and decrease the efficiency of the system.

In this work, we use a weighted tree for resource discovery (Khanli & Kargar, 2011). We only use one bitmap for the identification of available resources in nodes and also resources of children and their descendant nodes. The users' request must be transformed into this bitmap. We record a footprint of resources in nodes. When a user's query reaches every node, we can use this footprint to access the directly appropriate resource without visiting additional and unnecessary nodes and no time is consumed. We compare our algorithm with other algorithms and show that our algorithm is very efficient.

We discuss the previous works about the resource discovery in Section 2. In Section 3, we explain our proposed mechanism. Section 4 details the diagrams for comparing our method with previous methods, and finally, Section 5 contains conclusions.

2. A brief history of the resource discovery in grid

There are different methods for resource discovery in grid environment. Centralized resource discovery approaches (Berman et al., 2003; Chien et al., 2003; Foster & Kesselman, 1997; Germain et al., 2000; Mutka & Livny, 1987; Neary et al., 1999) are one of the mechanisms which suffer from single point of failure and bottlenecks.

Most of the methods in resource discovery tend to peer to peer (Al-Dmour & Teahan, 2005; Ali et al., 2005; Basu et al., 2005; Bharambe et al., 2004; Cai & Hwang, 2007; Iamnitchi et al., 2002; Koo et al., 2006; Nejdl et al., 2002; Oppenheimer et al., 2004; Shen, 2009; Talia et al., 2006; Zerfiridis & Karatza, 2003) which for example use super-peer (Mastroianni et al., 2005) or hierarchical (Liu et al., 2006) models. Apart from similarities between grid and peer to peer systems (Iamnitchi & Talia, 2005), they have critical differences in the field of security, users, applications, scale and etc. (Trunfio et al., 2007).

In the grid some methods use resource brokers to match the user's request with available resources (Bradley et al., 2006). They can consider some factors including software /hardware capabilities, network bandwidth, resource cost and so forth.

Yuhui Deng et al. (2009) suggested a peer to peer model for resource discovery which uses ant colony optimization (ACO). The main idea of this method was inspired from ants, which search their environment for food.

Xue-Sheng Qi et al. (2006) suggested a mechanism which can find multi-accessible resources and choose one of them, which uses a table. When a user wants to use the resource, reservation table will be checked. If the desired resource does not exist, it will be added to the table and the resource will be reserved.

Another method for resource discovery problems would be semantic communities (Li & Vuong, 2005; Li, 2010; Nazir et al., 2005; Zhu et al., 2005; Zhuge, 2004) which allows the grid nodes to communicate with no requirement to a central visiting point.

In (Li, 2010), Li proposes a semantics-aware topology construction method where queries propagate between semantically related nodes. To route the query, it constructs and uses the Resource Distance Vector (RDV) routing table (RDVT).

Gregor Pipan (2010), used TRIPOD overlay network for resource discovery which is based on a hybrid overlay network. He also used a K-Tree. The recommended TRIPOD overlay in this method is especially designed for resource discovery, which is combined two structures in an overlay and also used synthetic coordinate system.

Tangpongprasit et al. (2005) proposed an algorithm which uses the reservation algorithm for finding suitable resources in a grid environment. In the forward path, if there are any resources, they will be saved and reserved, in the backward path, one of them will be selected (if more than one resource has been reserved) and added to the request.

In (2006), Ramos and de Melo propose a structure of master and slave. A master does the updating and the slave restores the information from the machine.

In (2010), Chang and Hu proposed a resource discovery tree using bitmap for grids. It uses two bitmaps called the ''index bitmap'' and ''local resource bitmap'', and the other bitmap would be the ''attribute counter''. The local resource bitmap registers information about the local resources of nodes and the index bitmap registers the information about its children nodes which exist in the nodes that have child (non-leaf nodes). In this method, the users' query at first becomes AND with the local resource bitmap and if there is no local resource in the node, it becomes AND with the index bitmap. If the result is a nonzero number, the query will be forwarded to all children until reaching the target node. If the result of the AND operation is zero, it means that there are no resource in children and the query will be sent to the father node.

There are some differences between our algorithm and the other ones:

Our algorithm uses a tree structure in which the edges have weight. The advantage of our method is that any node in our weighted tree has a unique path, so the user's query against all of previous methods is not sent to the extra and unnecessary paths. We can directly reach the target node using a resource footprint which is stored in nodes. Furthermore, for resource discovery we only use one bitmap in every node which is for the storing of information about its local resources and the resources of its children and descendant. Also it preserves a footprint of resources and if we need a resource which is available in its children or descendant, we can directly and without any referring to unnecessary and extra nodes, reach the target node. This method significantly reduces the system traffic and increases the performance of system.

3. Our proposed method

Here, we introduce a weighted tree structure for resource discovery. In this method, like (Chang & Hu, 2010) (explain in previous section) we use one bitmap except several bitmaps will replace one bitmap with different contents. In the current method, we will allocate to any kind of resource two positions of bitmap, where one of them is the ''counter'' and another is the ''path''. A user's request should be changed into this form. There will be one bitmap in every node in which the user's query will be multiplied by this bitmap position to a position in order that the availability of matched resources would be investigated. If the current node contains the requested resource, so the requested resource is found, otherwise if it does not contain the requested resource or have the requested resource in its children and descendant, the query will be sent to the father node. However, if the node does not contain the requested resource but it is available in one of its children or descendant, the target node can be found directly and without any referring to extra nodes using the information stored in this node and weighted edges. Therefore, our offered method decreases unnecessary referrals to other nodes and reduces the additional traffic, so the time is saved and the ability of resource discovery is improved.

Due to the existence of different resources in the grid environment and also the various types of resources (attributes), the resources and their attributes should be identified in our bitmap. This bitmap has positions that numbers will be stored in. The length of the bitmap depends on the attributes of each resource. For example, if our expected resource is an operating system (OS), the different types of operating systems that are used in our grid environment determine the length of our bitmap. If the types of operating systems are XP, Linux, Unix and MacOS, our bitmap will have six positions. Actually, we allocate two positions for each type in the bitmap: the counter and path. All of the users' queries should be transformed into this form. Fig. 1, shows a sample of this bitmap that contains the XP operating system in the related node.

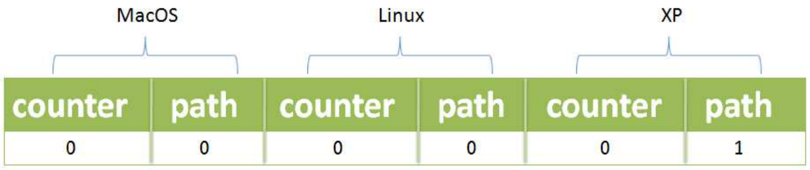


Fig. 1. A sample of OS type bitmap.

A node having a special resource locates the numbers 01 in two related positions. Number 0 is for counter and because this resource exists in the node itself, so the number would be 0 and the next one (Number 1) indicates the availability of resources in this node (path). Another position would be 0 indicating that no resources would be available in the node or its children. Each node in our tree contains such a bitmap at the first step and in the next steps the contents of these positions will be changed.

Our algorithm would be on a tree in which a weight will be allocated to any edge by the father nodes, i.e. each node allocates a special binary number to each child depending on the number of children. This node can determine the number of weight bits using following simple formula:

$$y = \left\lceil \log_2^n \right\rceil$$

where n is the number of children of this node, and y is at least the number of bits that should be allocated to edges as weight.

In Fig. 2, we show a weighted tree with its nodes and local resources which is regarded as the first step in constructing a weighted tree with local resources of nodes without considering the resources of children. In Fig. 3, we show same tree after gathering the children information by the father nodes which is the final form of the resource discovery tree. Now we describe the construction of the final form of a tree with an example.

Suppose that our expected resource is an operating system (OS) as Fig. 1, depicts. Each pair position is representative of one type of resource (which here is one type of OS). In Fig. 2, node K has operating system XP (01 in the last two positions), node J has operating system Linux, node F has operating system MacOS etc. The weight of edges is also written on the edges. As said before, these weights, for example 0 between node A and node B is a number in which node A is considered for node B in its memory. Now these nodes should form the final form of the tree in order that the users' requested resources can be found.

For example, we explain the method of gathering information of children and construct the final form of the bitmap in node E. First consider nodes J, K and L. Node E as the father node, takes bitmaps of these three nodes, and analysis them, storing the final information on its bitmap. Suppose that node E takes the information of nodes J, K and L. First we begin from lower positions (from right to left). The first pair position of nodes J, K and L are removed. Two of them contain 00 and the other one contains 01, node E itself has 00, so there is only one resource that is located in node K, which should be registered in the first pair position of node E. Because this resource information is received through a path with weight 01, so we write in the first position the path that the resource information comes from, i.e. 01. We transform this binary number to a decimal number and then record. If the second position which is the counter, increased by two units, namely it would be 2 (increased two units because the weights of node E are two bits). Finally, two numbers, 2 and 1 are registered in the first two positions of node E.

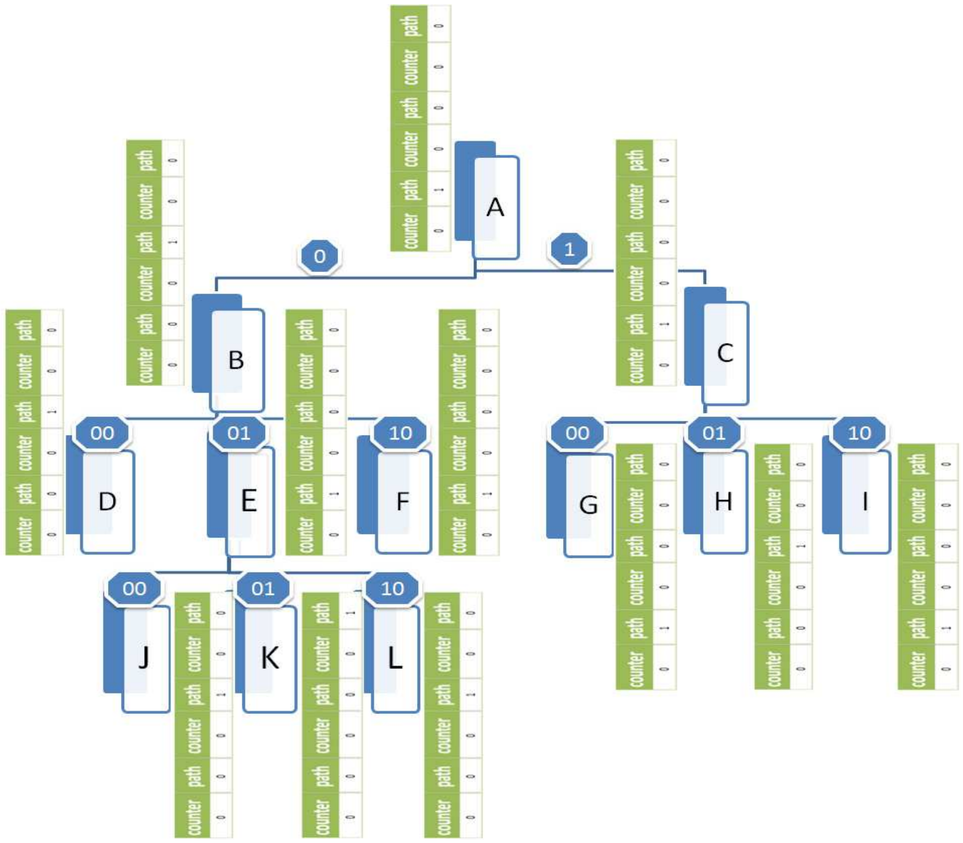


Fig. 2. Our proposed tree before gathering the children information.

Now the next two positions of nodes J, K and L are surveyed (related to the Linux operating system). Two positions related to node K, are 0 0, and nodes J and L are 0 1, meaning that nodes J and L own this resource and the information of one of these resources should be registered in the bitmap of the father node (E). In our method, any one whose counter shows a small number indicates an association with higher levels of the tree. In this case both of them are equal, so we choose one of them randomly (node J). Finally, two numbers, 2 and 0 are registered in the second two positions of node E.

In the next two positions, because node E owns this resource, there is no need for the resource information of children to be replaced by its own resource information. So, these two positions remain as 0 and 1.

The remaining nodes are made in this way using the suggested method. We show in Fig. 3, all related information and our resource discovery tree is formed as mentioned.

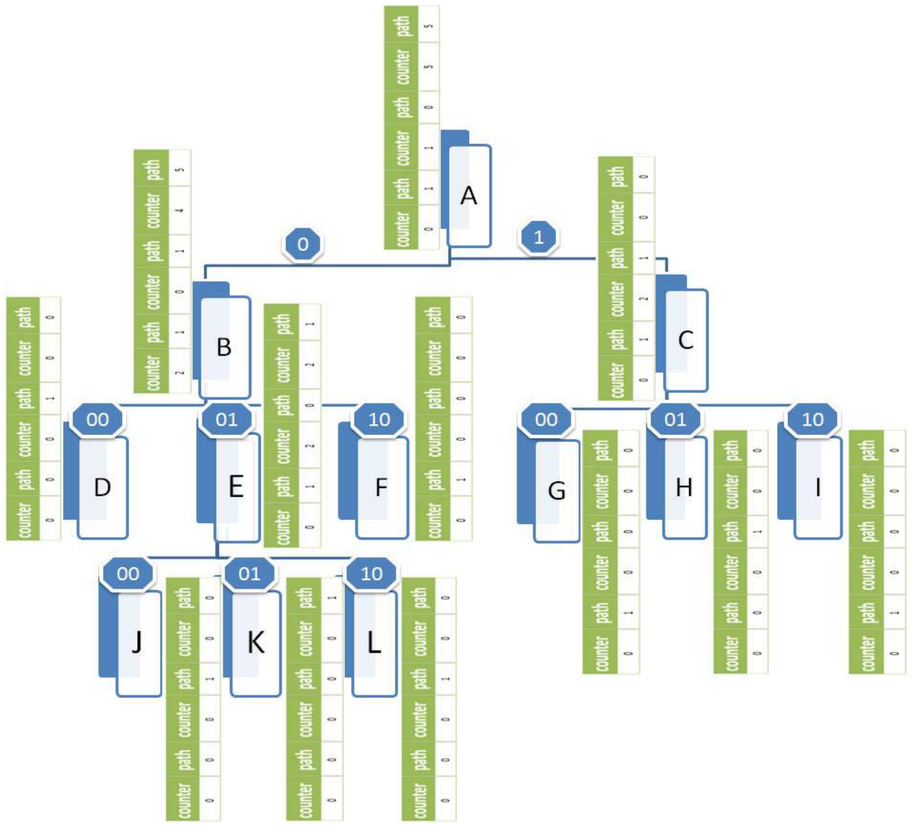
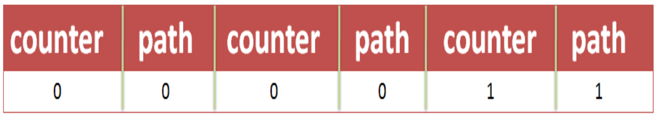


Fig. 3. Our proposed tree after gathering the children information.

3.1 Resource discovery

When any user sends his request to one of the nodes of our resource discovery tree which should be in the form of bitmaps stored in nodes, i.e. in the previous tree structure, the user's query should be in the form of a bitmap with the length of six positions (Fig. 4). The user should allocate number 1 in the pair position (1 1) related to the resource he requires. In Fig. 5, we show the method of resource discovery.

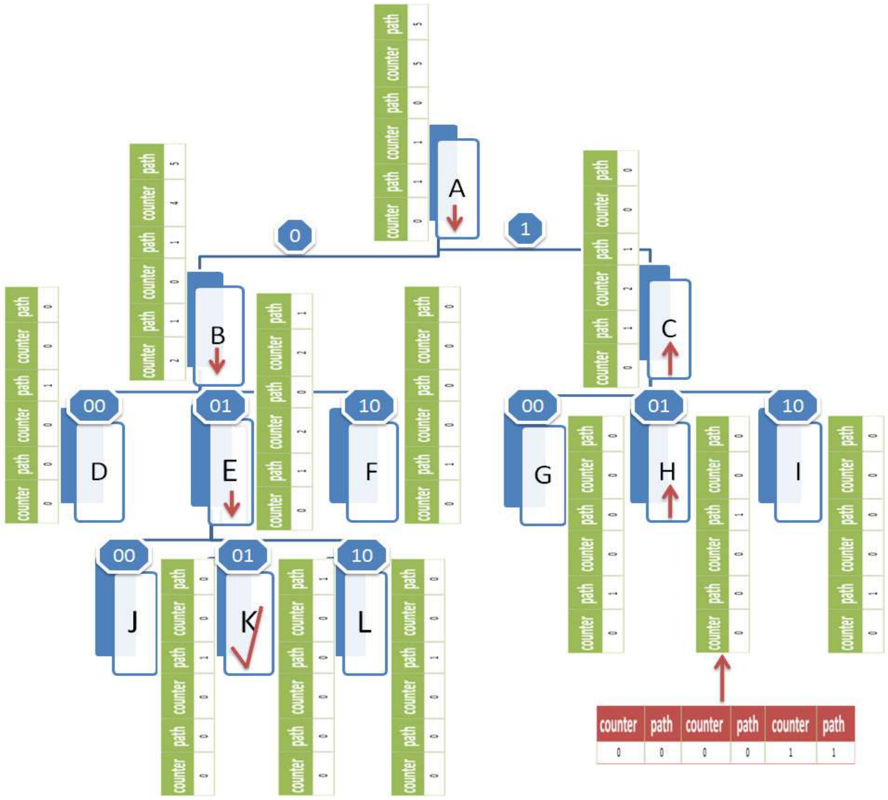


counter	path	counter	path	counter	path
0	0	0	0	1	1

Fig. 4. A sample of query bitmap.

Suppose that a user needs the operating system XP. The user should create a query in the form of a bitmap that has been shown in Fig. 4. All positions of the query bitmap are multiplied with the stored bitmap in the nodes. If the result is 0, the query will be sent to the father node. Otherwise the resulted numbers will be used for resource discovery.

Suppose in this example, the query is first sent to node H. All positions are multiplied with each other and the result would be 0. So, the request will be sent to its father node i.e. C (as Fig. 5). In node C, all positions of request are multiplied with the stored bitmap in this node and the result again would be 0. So the request will be sent to its father node i.e. A. It is observed that the result of multiplication in the node would be the numbers except 0 (5 5). So, number 5 should be changed into a binary number with length 5 so the result is 00101. Using this binary number, we can directly obtain the node which contains the requested resource; the binary number is taken through left to right and goes forward in the edges of the path. The result node would be the target node. Fig. 5 shows this search which resulted in the resource discovery in node K. As soon as a number except zero is achieved, we can directly refer to the destination node without any referring to an extra and unnecessary node.



counter	path	counter	path	counter	path
0	0	0	0	1	1

Fig. 5. An example of resource discovery in our weighted tree.

3.2 Improvement the method in a real grid environment

In the current method, the weight of edges in each node is identified and after changing into a decimal number, they are stored in their related places.

The method can be corrected in a real grid environment, because if we suppose a real grid environment with many nodes and edges, the number of 0 and 1s which are to be attached to each other will be large and in storing or converting them into their equivalent decimal, a very large number will result (maybe incomputable).

The solution would be that for storing 0 and 1s, we suppose a threshold number. When a node takes information from its children nodes, including the bitmap, the node first takes the contents of this bitmap then converts the contents of the path position to its equivalent binary number (in the length of counter) and attaches the weight of edge. If the resulted binary number is larger than threshold, this node instead of storing this large number, supposes that the node which this a large number comes from there, is a target node, namely this resource exists in this node, and so stores just the related weight of the edge in the counter, and in fact, the counter in the bitmap starts again. Then this information is sent to the father node and the above operation again will be performed.

Now, when the request reaches one of the nodes and is directed to the target node, a comparison should be made in the target node to identify if the stored number in the target node is 0 and 1, which means the resource is available in this node, otherwise the existing number moves forward of the target node and the above operation again will be performed. Fig. 6, shows this method with threshold = 2 bit, that when the length of bits becomes large in node B from 2 bits, so node B supposes that node C is a target node and stores the address of node C in its bitmap, and in fact from node B the counter will start again.

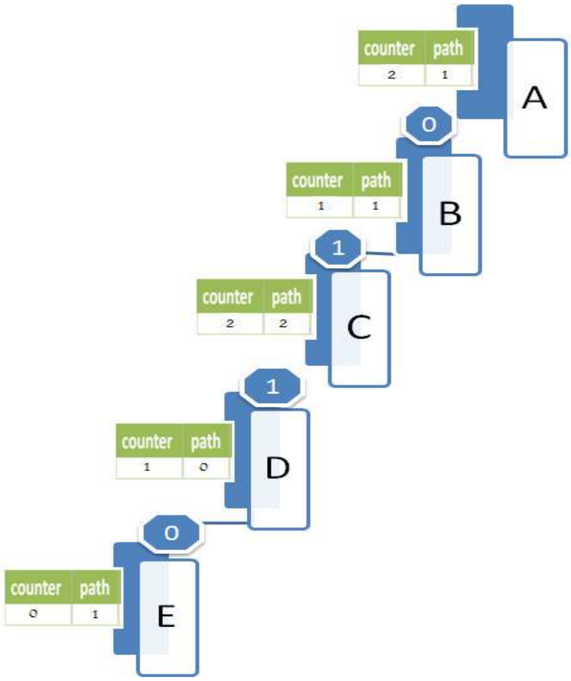


Fig. 6. An example of resource discovery tree with threshold = 2.

Another problem is the large number of the participating nodes and heterogeneity of their available resources. A user may need several resources which may send them in the form of several query bitmaps. It means that a lot of information will not be sent in the form of a unique query. But, we should identify the queries in nodes, i.e. if the query reached, relates to resource 1 or resource 2 etc. (for example OS, CPU, . . .).

We can add some bits to any query bitmap to identify the resource. For example, if there are 16 types of resources in our grid environment, we can add 4 bits to any query bitmap, in order that the receiver can identify with which resource the bitmap should be compared to. This enables us to manage several resources in the grid environment.

3.3 Complexity analysis

Here, the time and space complexities for our method will be investigated. We suppose that there are n nodes and r resources in the grid environment. Every resource has a attributes and the length of the bitmap would be b bits (Chang & Hu, 2010). Also assume the maximum number of children for a node would be m.

3.3.1 Time complexity

Our method applies to a tree structure. Users send their requests to one of the nodes and the request moves in the tree paths for the proper resource to be found. The consumed time includes the required time for computations and communications.

The computation time would be the time consumed in every node to compare the query bitmap with local bitmaps. So, we can calculate the number of nodes visited in resource discovery in our tree for computation time.

The latter would be the time lost between nodes to send a query. Here we can also calculate the number of links. The number of links and visited nodes in resource discovery would be in the same order in tree paths, so both times are supposed the same. One of these times would be larger depending on the network conditions and type of nodes. Because most nodes in the grid environment are of good computation power, but are located in distant intervals in various networks, in the following we assume the communication time through a link is larger than the computation time in a node.

In Our method like (Chang & Hu, 2010), the worst case in the resource discovery is the number of visited links from a leaf node upward to the root then downward to another leaf node. Therefore, the time complexity for our method will be O ($\log^{n}{}_{m}$) in the worst case.

3.3.2 Space complexity

To calculate the space complexity, every node in our tree requires $2 \times r \times a \times b$ places for the path bitmap (any attribute needs two places in the path bitmap) and $r \times a \times b$ places for the counter bitmap and $(m + 1)$ links. Like (Chang & Hu, 2010), if supposing that every place and each link needs 32 bits, the total number of bits required is $n \times (32 \times (2 \times r \times a \times b + r \times a \times b + (m + 1))) = O(nrab + nm)$.

3.4 Update method

For updating, an extra bitmap called a ‘‘counter bitmap’’ will be used. The number of positions in this bitmap depends on the number of types of resources in the grid environment. For example, if there are 5 kinds of operating systems, this counter bitmap will also have 5 positions. The current bitmap represents the number of resources existing in children or descendant of that node.

The method is that the user places two 1s in pair positions which are requested for that resource, and if the request reaches a node, multiplies with the bitmap present in that node until it reaches a node in which its multiplication result would be a number except zero. Using these two numbers and the weight of edges, the target will be found. In the update method, when a request reaches a node with information opposite zero, numbers are removed from the pair positions of that node and two number 0s are placed in two positions which the numbers are removed from and this process will be continued in the path nodes until reaching the target node. When reaching the target node, the numbers 0 and 1 available in two related positions would be 0 and 0.

If the expected resource is to be delivered to the user, no node must have access to the resource address, in other words another user cannot use it. In the current method, the path nodes wait some time after the user's query makes some changes in their path and counter bitmaps, until a message is received from the same path to which the user's query had been sent, to update the changed positions (0 0). If the period of time (time out) ends, this node restores its previous information to the related places.

Otherwise it receives a bitmap. First, this node investigates the pair positions which have been changed in its bitmap. If it contains the information of a resource, puts the contents of the mentioned pair positions in a related place in its bitmap and sends this to its father node and the father node again iterates the process, and because it observes that the contents of these two positions are opposite zero, it tries to reach the root which then the tree is updated.

In the next state, the pair positions that have arrived from the child contain the address of no resource (to be 0 0), so the node checks the contents of the related position in its counter bitmap. If this position is zero, it means that there is no resource in the children and descendant of this node and the bitmap will be sent to its father node, otherwise if this position is any number except zero, there will at least be information related to the expected resource in one of its children. So, this node takes from its children, the path bitmaps, and finds the address of considered resource from one of children, and replaces it with the removed pair positions and then delivers the path bitmap to its father node and the father node, there will be no need to collect the information of children and only the path bitmap which when the contents of its two positions is changed moves to the root node until it stops. In Fig. 7, we show an example of update. Suppose that there are two types of resources in this grid environment, so our path bitmap will have 4 positions and the counter bitmap 2 positions. If a request sent to node B, as (Fig. 7), the result of multiplication of this request with the path bitmap of node B would be zero and the request will be sent to the father node i.e. node A, there is a resource in address 3 6 (110).

The request removes 3 6 from path bitmap of node A, and also decreases by one unit the contents of the related position in the counter bitmap. Then the request goes to node C and the process continues. Finally, it reaches node F, the target node, so its contents which have been 0 1 will be 0 0 and it reserves this resource.

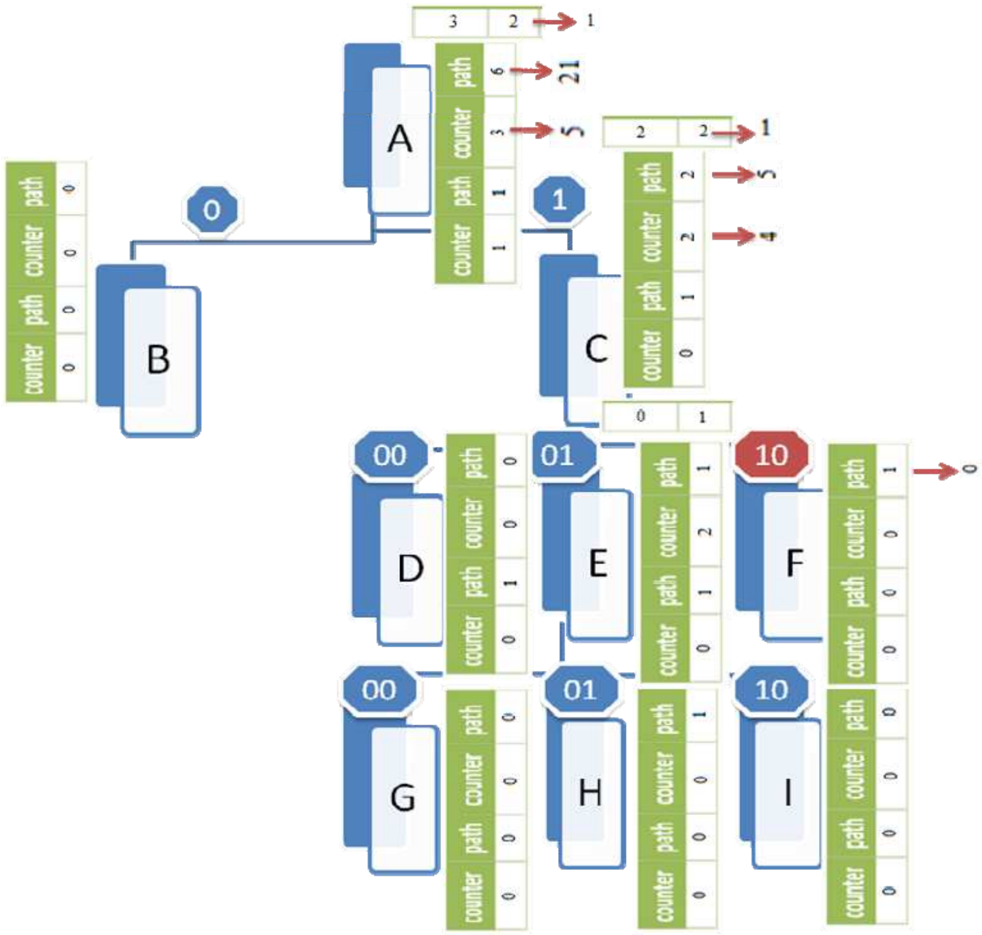


Fig. 7. An example of tree update.

In the update path, node F sends its changed bitmap to node C. If this information does not reach before the time that node C has been considered, node C restores its bitmap to the previous state, otherwise node C, after receiving the bitmap of node F, investigates the condition of removed pair positions and checks the related position of these pair positions in the counter bitmap which is 1 in node C (a nonzero number), so for the information of a resource available in one of the children of node C (nodes D or E), node C takes the information of the children nodes (except a child which has taken that changed bitmap (here node F)). Finally, node C, places numbers 4 and 5 in the condition of removed positions after investigating the information of nodes D and E. So, the bitmap will be sent to node A. Node A after observing that the contents of related pair positions are nonzero, sees there is no need to collect the information of the children, and with this information, updates its previous information. In our update method, information of children is gathered only in one of levels, and all through the path, just visited by one node in any level of tree.

4. Simulation results

In this section, we compare our simulation results with the recommended algorithm in (Chang & Hu, 2010), flooding-based approach and the algorithm proposed in (Marzolla et al., 2005; Marzolla et al., 2007) (MMO). In the first simulation tests, the total number of nodes which are visited during in the resource discovery and in updating are compared with algorithm (Chang & Hu, 2010) for 400 queries. In the second experiment, the nodes which queries send during resource discovery for the flooding-based approach, MMO algorithm and resource discovery tree algorithm, are shown and compared with our method.

As we know, for calculating the traffic which a method causes, we can calculate the number of links which are occupied for resource discovery or updating in the method. If the number of links visited for resource discovery or updating in a method is lower, the method has lower traffic and would be more efficient.

In the next simulation tests, the traffic in our method which is caused by the increased number of tree nodes in resource discovery and updating is indicated and the results are obtained for different nodes with 300 and 1000 queries. In the last simulation, we observe that the traffic caused in our method would be lower than other methods. This test is performed supposing there are 300 queries for different nodes, and the flooding-based, MMO, resource discovery tree and our methods are compared.

4.1 Simulation settings

We performed the simulation in the MATLAB environment. Our resource discovery tree is a full n-ary tree. It means that any non leaf node should have exactly n children. According to Mastroianni et al. (2007), and Chang & Hu (2010), we also assume that the resource discovery tree for simulation has height 4.

In the first experiment tests, we compare our algorithm with a resource discovery tree with a different number of index servers. Like (Chang & Hu, 2010), in this experiment there are 200 nodes in resource discovery tree and we perform this experiment with 180 queries.

We place the resources randomly in each node and then queries are sent through tree paths and compare the number of nodes that visited in each method. In Fig. 8, the difference between the number of visited nodes with two methods are observed. In this experiment, the number of visited nodes is investigated with changing the number of index servers. For example, when the number of index servers in tree is 10, so there are 10 nodes in level 1 and 190 nodes in level 2 (19 children for each node in level 1) for a tree with height 3. Because our method in the forward path just visits one node in every level so in Figs. 8, the simulations related to our method almost show the fix values.

In the second simulation tests, we assume there are 300 queries and a tree with height 4. In Fig. 9, we compare the number of nodes that queries send in our algorithm with the resource discovery tree and show that the number of nodes visited in our proposed method is lower than the previous method.

In the third simulation tests, we show the total number of nodes that were visited in the resource discovery path and for tree updating with assume 400 queries and different number of nodes for our method and compare with the other one. In Fig. 10, it is indicated that in our algorithm, fewer nodes are visited compared with the previous one.

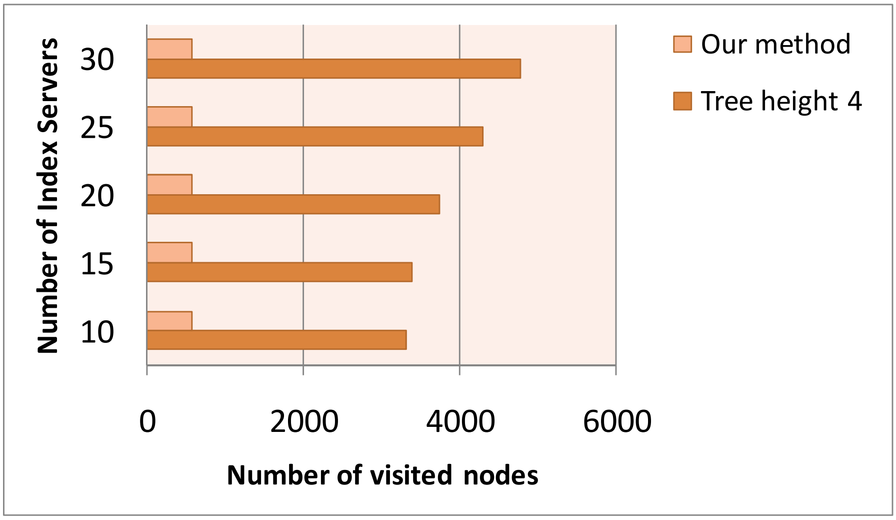


Fig. 8. The number of nodes that queries are forwarded to for 180 queries.

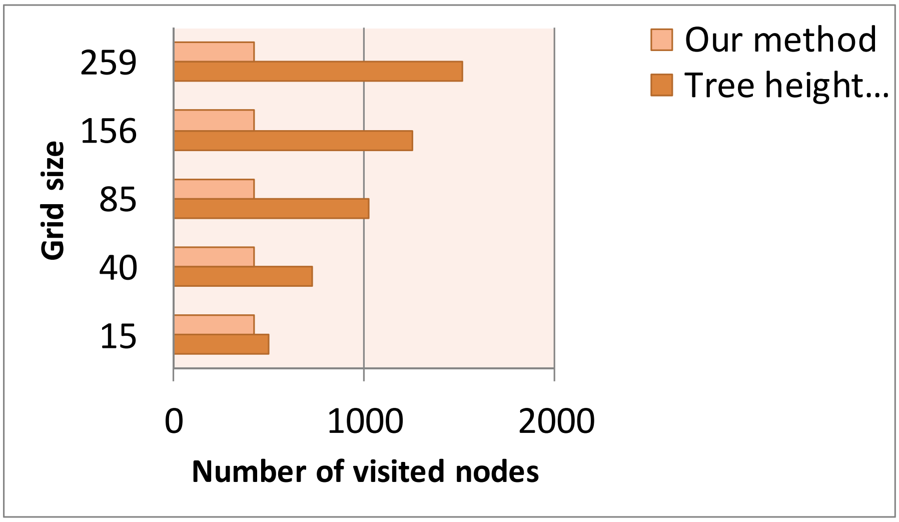


Fig. 9. The number of nodes that queries are forwarded to.

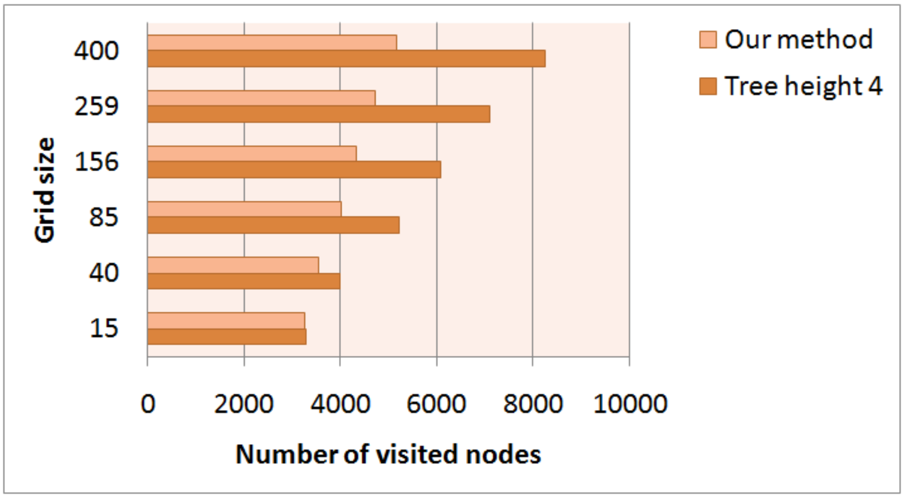


Fig. 10. Total number of nodes that are visited in resource discovery and updates.

In the next simulation tests, our method is compared with flooding-based method, MMO and resource discovery tree algorithm. In the current experiment supposing that there are 300 queries, in Fig. 11, it is indicated that the average number of nodes that queries are sent to is lower than other methods in our proposed method. The test is performed in a tree with height 4.

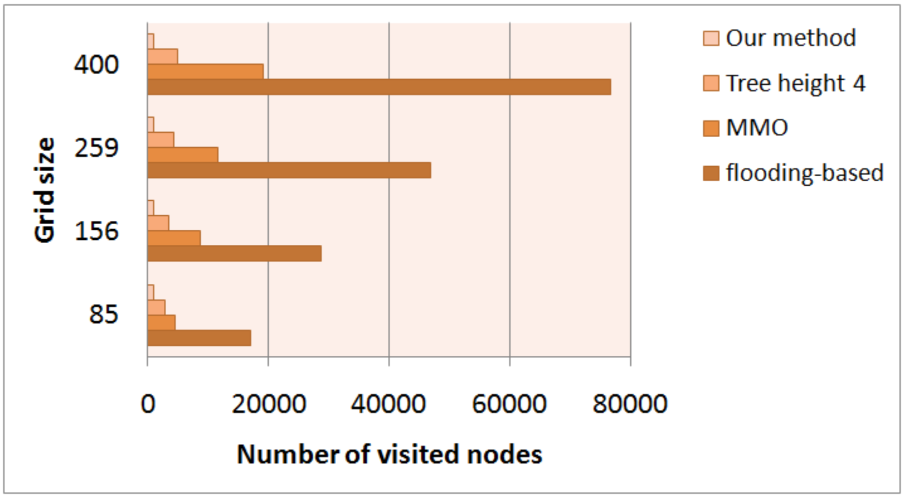


Fig. 11. Average number of nodes that queries are forwarded to using different methods.

In the next simulation tests, the number of occupied links in our method during the resource discovery phase and update phase is observed for different nodes with 300 and 1000 queries. In Fig. 12, we show the occupied links (traffic) for resource discovery and updating for 300 (Fig. 12) and 1000 queries (Fig. 13).

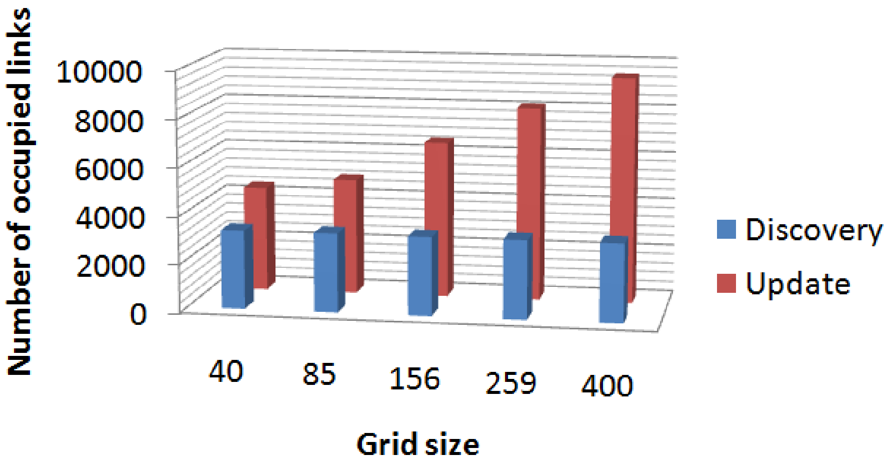


Fig. 12. Number of links that resource discovery queries and updates are forwarded to for 300 queries.

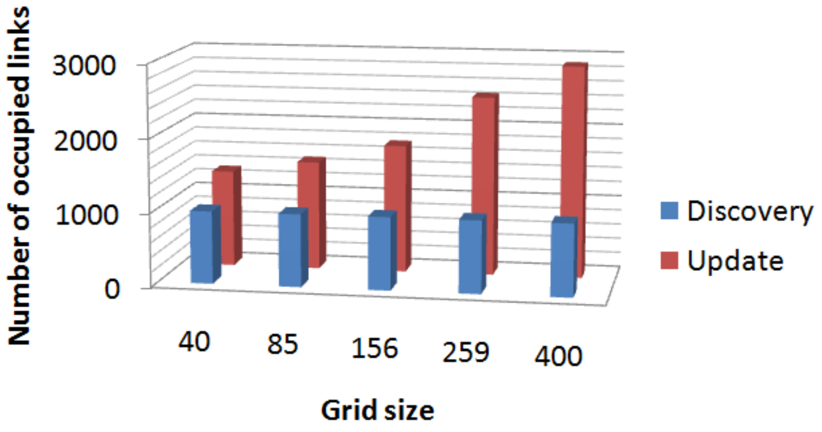


Fig. 13. Number of links that resource discovery queries and updates are forwarded to for 1000 queries.

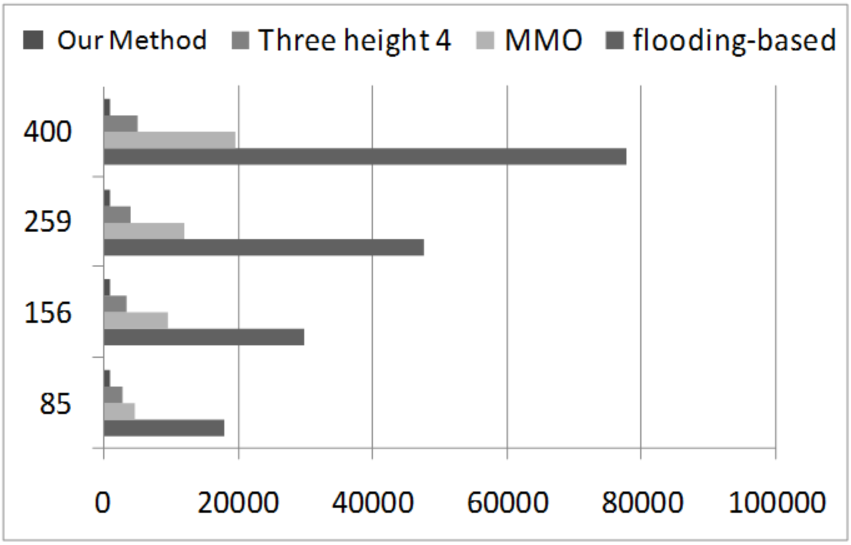


Fig. 14. Number of links that are visited by resource discovery queries for 300 queries.

In the last experiment, we supposed that there are 300 queries, and we show the visited links (traffic) which are caused during resource discovery in our method and compared with flooding-based, MMO and the resource discovery tree and for different numbers of nodes. In Fig. 14, we can see the traffic caused in our method is lower than other methods

5. Conclusions and future work

In this work, we use a tree structure in which the edges have weight. The advantage of our method is that any node in our weighted tree has a unique path, so the user's query against all of previous methods is not sent to the extra and unnecessary paths. We can directly reach the target node using a resource footprint which is stored in nodes.

Furthermore, for resource discovery we only use one bitmap in every node which is for the storing of information about its local resources and the resources of its children and descendant. Also it preserves a footprint of resources and if we need a resource which is available in its children or descendant, we can directly and without any referring to unnecessary and extra nodes, reach the target node. This method significantly reduces the system traffic and increases the performance of system.

We compare our algorithm with previous algorithms using simulations and results and show that the number of nodes visited in our resource discovery algorithm is less than that for other algorithms, and the difference would be significant with an increase in the number of nodes. Also the cost of update in our proposed algorithm is low.

In future, if we could a present a technique that could locate several heterogeneous resources (with different attributes) of a grid environment in smaller forms with a lower volume, or placed in one bitmap involving some factors, for example allocated costs for resources etc., we could improve the algorithm.

6. References

Al-Dmour, N.A. & Teahan, W.J. (2005). Peer-to-Peer protocols for resource discovery in the grid, in: *Parallel and Distributed Computing and Networks*, Innsbruck, Austria.

Ali, K.; Datta, S.; Aboelaze, M. (2005). Grid resource discovery using small world overlay graphs, in: Proceedings of the 18th IEEE Canadian Conference on Electrical and Computer Engineering.

Basu, S.; Banerjee, S.; Sharma, P.; Lee, S. (2005). NodeWiz: peer-topeer resource discovery for grids, in: 5th International Workshop on Global and Peer-to-Peer Computing (GP2PC) in Conjunction with CCGrid.

Berman, F. & et al. (2003). Adaptive computing on the grid using AppLeS, TPDS 14 (4).

Bharambe, A.R.; Agrawal, M. & Seshan, S. (2004). Mercury: Supporting scalable multiattribute range queries, in: *Proc. of ACM SIGCOMM*, pp. 353–366.

Bradley, A.; Curran, K.; Parr, G., (2006). Discovering resource in computational GRID environments. Journal of Supercomputing 35, 27–49.

Cai, M. & Hwang, K. (2007). Distributed aggregation algorithms with load- balancing for scalable grid resource monitoring, in: *Proc. of IPDPS*.

Chang, R.-S. & Hu, M.-S. (2010). A resource discovery tree using bitmap for grids, *Future Gener. Comput. Syst.* 26 29–37.

Chien, A.; Calder, B.; Elbert, S. & Bhatia, K. (2003). Entropia: Architecture and performance of an enterprise desktop grid system, J. Parallel Distrib. Comput. 63 (5).

Deng, Y.; Wang, F.; Ciura, A. (2009). Ant colony optimization inspired resource discovery in P2P Grid systems, J Supercomput 49: 4–21.

Foster, I. & Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit, *Int. J. High Perform. Comput. Appl.* 2 115–128.

Foster, I. & Kesselman, C. (2003). The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., San Francisco, CA.

Germain, C.; Neri, V.; Fedak, G.; Cappello, F. (2000). XtremWeb: Building an experimental platform for global computing, in: Proc. of IEEE/ACM Grid, December.

Iamnitchi, A.; Foster, I.; Nurmi, D. C. (2002). A peer-to-peer approach to resource location in grid environments, in: Proceedings of the 11th Symposium on High Performance Distributed Computing, Edinburgh, UK, August.

Iamnitchi, A. & Talia, D. (2005). P2P computing and interaction with Grids, *Future Gener. Comput. Syst.* 21 (3) 331–332.

Khanli, L.M. & Kargar, S. (2011). FRDT: Footprint Resource Discovery Tree for grids, Future Gener. Comput. Syst. 27 148–156.

Koo, S.G.M.; Kannan, K. & Lee, C.S.G. (2006). On neighbor selection strategy in hybrid Peer-to-Peer networks, *Future Gener. Comput. Syst.* 22, 732–741.

Li, J. (2010). Grid resource discovery based on semantically linked virtual organizations, *Future Gener. Comput. Syst.* 26, 361–373.

Li, J. & Vuong, S. (2005). Grid resource discovery using semantic communities, in: *Proceedings of the 4th International Conference on Grid and Cooperative Computing*, Beijing, China

Liu, H.; Luo, P. & Zeng, Z. (2006). A structured hierarchical P2P model based on a rigorous binary tree code algorithm, *Future Gener. Comput. Syst*.

Marzolla, M.; Mordacchini, M. & Orlando, S. (2005). Resource discovery in a dynamic environment, in: *Proceedings of the 16th International Workshop on Database and Expert Systems Applications*, DEXA'05, pp. 356–360.

Marzolla, M.; Mordacchini, M. & Orlando, S. (2007). Peer-to-peer systems for discovering resources in a dynamic grid, *Parallel Comput.* 33 (4–5) 339–358.

Mastroianni, C.; Talia, D. & Verta, O. (2005). A super-peer model for resource discovery services in large-scale grids, *Future Gener. Comput. Syst.* 21 (8) 1235–1248.

Mastroianni, C.; Talia, D. & Versta, O. (2007). Evaluating resource discovery protocols for hierarchical and super-peer grid information systems, in: *Proceedings of the 15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing*, PDP'07, pp. 147–154.

Mutka, M.; Livny, M. (1987). Scheduling remote processing capacity in a workstation processing bank computing system, in: Proc. of ICDCS, September.

Nazir, F.; Ahmad, H.F.; Burki, H.A.; Tarar, T.H.; Ali, A. & Suguri, H. (2005). A Resource Monitoring and Management Middleware Infrastructure for Semantic Resource Grid, SAG 2004, in: LNCS, vol. 3458, pp. 188–196.

Neary, M.O.; Brydon, S.P.; Kmiec, P.; Rollins, S. & Capello, P. (1999). JavelinCC: Scalability issues in global computing, *Future Gener. Comput. Syst.* J. 15 (5–6), 659–674.

Nejdl, W.; Wolf, B.; Qu, C.; Decker, S.; SIntek, M.; Naeve, A.; Nilsson, M.; Palmer, M.; Risch, T.; Edutella. (2002). A P2P networking infrastructure based on RDF, in: Proceedings of the WWW2002, May 7–11, Honolulu, Hawaii, USA, pp. 604–615.

Oppenheimer, O.; Albrecht, J.; Patterson, D.; Vahdat, A. (2004). Scalable wide-area resource discovery, Technical Report TR CSD04-1334, Univ. of California.

Pipan, G. (2010). Use of the TRIPOD overlay network for resource discovery, Future Generation Computer Systems 26, 1257_1270.

Qi, X.S.; Li, K.L. & Yao, F.J. (2006). A time-to-live based multi-resource reservation algorithm on resource discovery in Grid environment, in: *Proceedings of the 2006 1st International Symposium on Pervasive Computing and Applications*, pp. 189–193.

Ramos, T.G. & de Melo, A.C.M.A. (2006). An extensible resource discovery mechanism for grid computing environments, in: *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, CCGRID'06, pp. 115–122.

Shen, H. (2009). A P2P-based intelligent resource discovery mechanism in Internetbased distributed systems, *J. Parallel Distrib. Comput.* 69, 197–209.

Talia, D.; Trunfio, P.; Zeng, J. & Högqvist, M. (2006). A DHT-based Peer-to-Peer framework for resource discovery in grids. Technical Report TR-0048, Univ. of California.

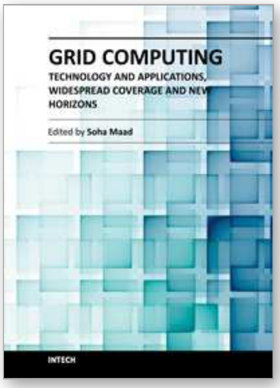
Tangpongprasit, S.; Katagiri, T.; Honda, H. & Yuba, T. (2005). A time-to-live based reservation algorithm on fully decentralized resource discovery in grid computing, *Parallel Comput.* 31 (6) 529–543.

Trunfio, P.; Talia, D.; Papadakis, H.; Fragopoulou, P.; Mordacchini, M.; Pennanen, M.; Popov, K.; Vlassov, V. & Haridi, S. (2007). Peer-to-Peer resource discovery in Grids: Models and systems, *Future Gener. Comput. Syst.* 23, 864–878.

Zerfiridis, K.G.; Karatza, H.D. (2003). Centralized and decentralized service Discovery on a peer-to-peer Network – a simulation study, in: Proceedings of the Sixth United Kingdom Simulation Society Conference, UKSim 2003, Cambridge, England, 9th–11th April, pp. 171–177.

Zhu, C.; Liu, Z.; Zhang, W.; Xiao, W.; Xu, Z. & Yang, D. (2005). Decentralized grid resource discovery based on resource information community, *J. Grid Comput.*

Zhuge, H. (2004). Semantics, resource and grid, Future Gener. Comput. Syst. 20 (1) 1–5.


GRID COMPUTING
TECHNOLOGY AND APPLICATIONS, WIDESPREAD COVERAGE AND NEW HORIZONS
Edited by Soha Maad
INTECH

Grid Computing - Technology and Applications, Widespread Coverage and New Horizons
Edited by Dr. Soha Maad

ISBN 978-953-51-0604-3
Hard cover, 354 pages
Publisher InTech
Published online 16, May, 2012
Published in print edition May, 2012