A Scalable Semantic Routing Architecture for Grid Resource Discovery

Juan Li, Son Vuong
Computer Science Department, University of British Columbia
{juanli, vuong}@cs.ubc.ca

Abstract

Grids technology enables the sharing and collaborating of wide variety of resources. To fully utilize these resources, effective discovery techniques are necessities. However, the complicated and heterogeneous characteristics of the grid resource make sharing and discovering a challenging issue. In this paper we propose a comprehensive semantic based resource discovery framework, which performs an effective searching according to the semantic properties of what is searched. In the framework, nodes are grouped into clusters according to some criteria. Resources are indexed and aggregated with a highly compressed format. The summarized index can act as network knowledge to guide routing in the network. Intra-cluster and inter-cluster routing strategies are proposed to support scalable and efficient searching. Results from simulation demonstrate that this architecture is very effective in grid resource discovery.

1. Introduction

The popularity of the Internet, the ubiquity of computers, and the availability of the high speed network technologies have led to the development of grid computing [1,2]. The objective of creating grids is to share and access large and heterogeneous collections of resources. Thus an effective and efficient resource discovery mechanism is crucial to realize this goal. However, in grids, the resource can have potentially very large number and they may have varieties of types, such as computing power, storage systems, network bandwidth, data sources, software, and devices. Moreover, they can be geographically distributed and owned by different organizations. For all the above reasons, the issue of resource discovery in large scale grid environment is very challenging.

Traditionally, grid resource discovery is usually managed with centralized or hierarchical servers. For example, Globus MDS-2 [3] uses an LDAP based directory service for resource registration and lookup. Condor's Matchmaker [4] adopts a centralized mechanism to match the advertisement between resource requesters and resource providers. However, these centralized servers can become bottlenecks and points of failures. So the system would not scale well when the number of the nodes increases. Peer-to-peer (P2P) applications have been successful in sharing resource. A current trend is to combine P2P and grids techniques together. [20] argues that these two technologies will benefit from converging into each other's field. The unstructured P2P systems like Gnutella [5] and FastTrack [12] often exploit either flooding or broadcasting searching mechanisms, which is clearly not scalable. Recently, super-peer networks [13] like Morpheus [14] or KzzaA [15] have been proposed to get the benefits of both centralized and distributed search. The DHT based P2P systems [16-19] are efficient and scalable, while a missing feature is the ability to support complex query. More recently, A few studies [10,21,22] extended the DHT scheme to support keywords or multi-attribute query. However, these systems require retrieving large amount of results from many different peers to find the intersection.

In this paper, we use RDF [6,7] to represent both resource and query. To support complex query without flooding the whole network, we use a hierarchical semantic routing algorithm. The principle of the algorithm is to use the content of query and the knowledge of the network to drive routing decisions. Specifically, nodes in the network are grouped into clusters according to their mutual interest, and those sharing similar interests are in the same cluster. Therefore, most queries can be satisfied within the cluster. Nodes in the same cluster build a tree. To share resource among clusters, the roots of the trees in all the clusters form an overlay network. Consequently, the query routing has two phases: the intra-cluster routing and the inter-cluster routing. Both routing schemes utilize Bloom filter [8] based summarization to keep and aggregate knowledge about the network. The network knowledge can help route queries towards peers known to have related concepts. This approach guarantees a higher query hit ratio with respect to other approaches oblivious of the resource location. Moreover, it reduces both the network and the peers load, thus providing a greater scalability.

The remainder of the paper is organized as follows. Section 2 describes the RDF resource representation. Section 3 explains the hierarchical semantic routing scheme. Section 4 gives the experimental results. Section 5 concludes the paper.

2. Resource representation

Metadata plays a central role in the effort of providing search techniques that go beyond string matching. We utilize RDF metadata representation to encode the resources. The benefit of representing with RDF is that the information maps directly and unambiguously to a decentralized model. Unlike traditional database systems, RDF does not require all annotations of a resource stored on one server. The ability for distributed allocation of metadata makes RDF very suitable for the construction of distributed repositories. The basic building block of RDF is the triple which includes a subject, a predicate and an object. The following example shows a fragment of the metadata of an electronic book. The definition is derived from the “Dublin Core” metadata definition [23].

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-
ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/">

<rdf:Description rdf:about="http://somewhere/Java
programming langrage">
  <dc:title> Java programming </dc:title>
  <dc:description>Java-Computer program
language</dc:description>
  <dc:creator>Ken Arnold</dc:creator>
  <dc:date>2002-09-01</dc:date>
  <dc:type> program language </dc:type>
  <dc:format>text/html</dc:format>
  <dc:language>en</dc:language>
</rdf:Description>
</rdf:RDF>
```

Figure 1. Resources representation with RDF syntax

With RDF representation, the resource providers can give resources better descriptions and the resource requesters can customize their requirements to make queries more precise and flexible.

3. Semantic routing scheme

3.1. Frame work

The framework utilizes P2P technology and hierarchical structure to improve the scalability and robustness. Nodes are grouped into clusters according to certain criteria, such as mutual interest, administrative domain and network distance. In this paper, nodes are clustered according to their registered interests. Those sharing the same interest are in the same cluster. Therefore, most queries will be satisfied within the cluster. Nodes in the cluster build a semantic tree. Queries are forwarded along tree paths leading only to matching nodes. We utilize the Prinkey method [24] to create the tree structure. In the tree, every node has a local resource summary as well as aggregated summaries from children branches. The query routing is based on these summaries. The main improvement of our scheme over the Prinkey scheme is the Bloom filter indexing method.

The root node in every tree cluster has complete knowledge of the entire cluster. To share resources among clusters, root nodes connect with each other forming an overlay network on top of the clusters. Queries cannot be satisfied in the local cluster should be forwarded to other clusters through the root nodes. The overlay forwarding strategy has a great impact on grids efficiency and scalability. We use semantic routing to forward the query to nodes which are most likely able to satisfy the query. Every root node computes the overlay routing table according to its knowledge of the local cluster as well as its knowledge of neighbor clusters. They exchange routing info by using a protocol similar to distance vector based IP routing protocol, but with summarized resource information inside. In this way, queries will be forwarded to the nearest resource provider. This hierarchical structure can achieve a balance between the inherent efficiency of centralized search, and scalability offered by distributed search. Figure 2 illustrates the resource discovery architecture.

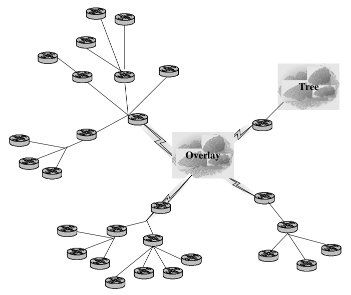


Figure 2. System architecture

3.2. Resource summarization

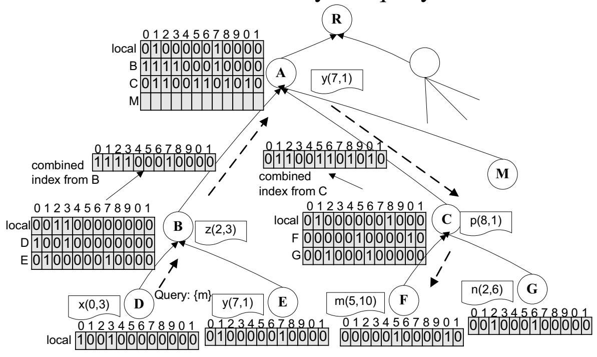
To make the routing "smarter", we should utilize the prior knowledge of where desired objects are likely to be to route the query. The RDF resource index is such kind of knowledge. However, exchanging the RDF indices between nodes is almost impossible, because each node may maintain a great amount of resources. Our strategy is summarizing the resource metadata. Summarization can not only save bandwidth and storage for transmitting and storing the metadata, but also can decrease the query lookup time, because summaries can be stored in the main memory. We use Bloom filter to summarize the resource info.

To map a resource to the Bloom filter bitmap, we hash all attribute combinations to the bitmap. For example, a resource *R* with *4* attributes *{a,b,c,d}*, can potentially satisfy *15* queries: *{a}*, *{b}*, *{c}*, *{d}*, *{ab}*, *{ac}*, *{ad}*, *{bc}*, {*bd}*, *{cd}*, *{abc}*, *{abd}*, *{acd}*, *{bcd}*, *{abcd}*. Therefore, all these *15* combinations should be put into the bitmap. While, a resource with *n* attributes may have 2^n-*1* combinations. When *n* is large, that could be a huge number. To solve this problem, the number of combinations should be restricted. We set a maximal length *m (m<=n)*, and only hash those combinations, whose lengths are not greater than *m*, to the bitmap. For resource *R* in the above example, if we set *m* as *2*, we only map *10* combinations (*{a}*, *{b}*, *{c}*, *{d}*, *{ab}*, *{ac}*, *{ad}*, *{bc}*, *{bd}*, *{cd}*) out of the 15 combinations to the bitmap. To determine if a query can be satisfied by resources mapped into the bitmap, we check all of the query's constraint combinations length up to *m*. If any of them is not in the bitmap, then certainly the query cannot be matched. Otherwise we conjecture that the query can be matched, although there is a certain probability of "false positive." Restricting the attribute concatenation length reduces the complexity of summarization, but it introduces more false positives. Fortunately, we do not need the strict accuracy for the summarization, because finally we will check the accurate RDF to confirm the match.

3.3. Intra-cluster Routing

3.3.1. Overview. Since any connected graph can be represented by a tree, tree is a natural representation of connected graph. We adopt Prinky's tree structure[24] as a basic structure for intra-cluster routing, but improve its approximate indexing scheme. In our tree structure, every non-leaf node maintains a routing table including several Bloom filter bitmaps: one bitmap for local resource and the rest others for children. Each node sends the merged bitmap to its parent. So every internal node has a summarized view of a sub-tree rooted by itself, and the root has a summarized view of the entire tree. When a node receives a query, it checks its routing table. If it finds match in local bitmap, it just gives a positive reply. If it finds match from a child's bitmap, it forwards the query to that child. If neither of them matches the query, and if the query is not received from its parent, the query will be sent to its parent. The parent will perform the same procedure. This routing scheme forwards query only to nodes lying on branches which potentially can satisfy the query and avoids sending the query to other nodes.

3.3.2. Routing example. Figure 3 illustrates the index aggregating and query forwarding process. In this example, the Bloom bitmap size is *12* bits and *2* hash functions (H_1,H_2) are used to map a resource. In reality the size of the bitmap is much larger, and the number of hash functions is always more. In the example, node *B*'s routing table includes a local bitmaps, and two children (*D* and *E*) bitmaps. A local resource *z* is mapped to two positions: *2* and *3* in the bitmap ($H_1(z)=2$, $H_2(z)=3$). So in *B*'s local bitmap, $B_2=1$, $B_3=1$. *B* merges these three bitmaps by bitwise *OR*, and sends the merged bitmap to its parent *A*. The merged bitmap represents all resources from *B* and its descendants. Now suppose *D* receives a query for resource *m*. It first uses the two hash functions H_1 and H_2 hashing *m* to *2* bits: 5 and 10 in the bitmap. Because *D* cannot find match locally, it forwards the query to its parent *B*. *B* cannot find match in its routing table either. So *B* forwards the query to its parent *A*. *A* finds match in child *C*'s bitmap (because C_5=C_{10}=*1*), then *A* forwards the query to *C*. Similarly *C* finds match from child *F*, so the query is then forwarded to *F*. Finally *F* finds match in its local bitmap and it will check its RDF database to further verify the query.



R
A
M
B
C
D
E
F
G
0 1 2 3 4 5 6 7 8 9 0 1
local 0 1 0 0 0 0 0 1 0 0 0 0
B 1 1 1 1 0 0 0 1 0 0 0 0
C 0 1 1 0 0 1 1 0 1 0 1 0
M
y(7,1)
combined index from B
1 1 1 1 0 0 0 1 0 0 0 0
combined index from C
0 1 1 0 0 1 1 0 1 0 1 0
local 0 0 1 1 0 0 0 0 0 0 0 0
D 1 0 0 1 0 0 0 0 0 0 0 0
E 0 1 0 0 0 0 0 1 0 0 0 0
z(2,3)
local 0 1 0 0 0 0 0 0 0 1 0 0 0
F 0 0 0 0 0 1 0 0 0 0 1 0
G 0 0 1 0 0 0 1 0 0 0 0 0
p(8,1)
x(0,3)
Query: {m}
y(7,1)
m(5,10)
n(2,6)
local 1 0 0 1 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 1 0
0 0 1 0 0 0 1 0 0 0 0 0

Figure 3. Tree routing

3.3.3. False positive. As mentioned, the Bloom filter index may raise false positives and the aggregation of index may incur more of them. Therefore, the index is only an approximation of the resource. It may lead queries to nodes or branches that do not contain relevant information. Luckily, this will not affect the fidelity of the final query result, because node that finally receives the routed query will check the accurate RDF database to further verify it. As long as the false positive rate is small, queries will be routed along nearly optimal paths and most of the nodes that finally receive queries will in fact contain relevant information.

3.4. Inter-cluster Routing

3.4.1. Overview. To share resource grid wide, we need route queries among clusters. Because the root of every cluster has a summarized index of the entire cluster, naturally it becomes a representative of the cluster. Root nodes connect with each other forming an overlay network. We call the overlay routing algorithm: resource-distance-vector (RDV) routing. It utilizes the distance vector to route queries to the nearest matching nodes. Every node (the root node) in the overlay network maintains a resource routing table. It utilizes the Bloom filter index, and adds distance information into the index. Nodes maintain resource information sent by their neighbors and update relevant entries in their routing tables. The distance information is updated from node to node and plus one whenever passing through a node. We set a TTL, which we call *Radius*, to limit the number of hops the resource information can travel. When a node receives a query request, the algorithm will choose the shortest route to forward the query to. If there is more than one provider supplying the same resource, with high probability, the algorithm will forward the request to the nearest one.

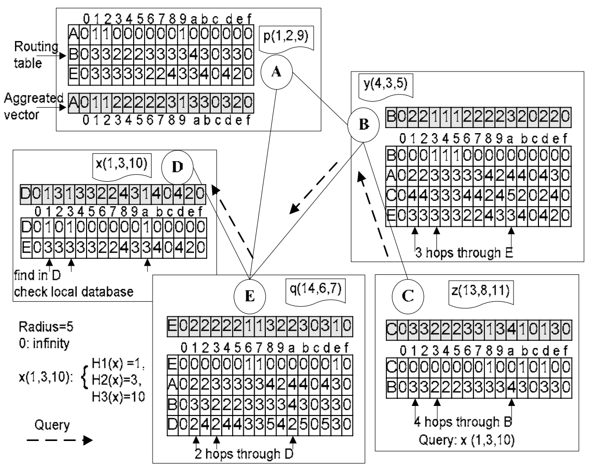


Figure 4. RDV routing

3.4.2. Routing table. Each node maintains a resource routing table which contains local and neighbor resource vectors. All the vectors are implemented with Bloom filters. Besides resource info, the vector also records the distance (in terms of number of hops) to the resource. So the Bloom index is now a vector of numbers instead of a vector of bits. Each number is the minimum distance to a matching resource. Figure 4 illustrates the formation of routing tables. In this example, we use *3* hash functions to map a resource to the vector. Node *A*'s local resource *p* is mapped to 3 numbers: *1*, *2*, and *9*, so in *A*'s local vector, those 3 positions are set *1*, representing *1* hop to the resource. (We assume the root node of the cluster is *1* hop away to any resources in that cluster). In *A*'s routing table, neighbor *B*'s vector has *3* elements B_4, B_3, B_5 set to *2*. That means resource *y (4,3,5)* is *2* hops away from *A*.

Node *A* merges all vectors in its routing table and sends the merged vector to each of its neighbors. The problem is how to decide the distance information in the merged vector. The principle is: if there are multiple paths to a resource, the node should always choose the shortest one. Assume *A*'s combined vector is *X*. So the value of *X*'s i^{th} element is the minimal value of all vectors' i^{th} element. (Note: *0* represents ∞ in figure 4). For example, $X_3 = min(A_3,B_3,E_3) = min(\infty,2,3) = 2$. By doing this, hash positions related to a resource may have different values. For example, in *A*'s combined vector *X*, to check a resource *x (1,3,10)*, we find $X_1=1$, $X_3=2$, $X_{10}=3$. According to the aggregation process, it is not difficult to see that the maximal value represents the real distance. Therefore, the distance from node *A* to resource *x* is *3*. The combined vector is then added by *1* to every element and sent to all of *A*'s neighbors. We set a hop count, which we call *Radius*, to limit how far the resource info can travel. When resource info passes as many as *Radius* hops, it should not be forwarded any more. In figure 4 *Radius* is set to *5*. So when $X_i>5$, X_i is set to *0*. Through the aggregated vector, neighbors of *A* would know what resources are available from node *A*, and how far they are.

Each node sends updates to and receives updates from its directly connected neighbors. When a node receives routing information from a neighbor, it updates its local table if the neighbor suggests a "better" route than what it already knew about. Eventually the table will stabilize and all resources within the *Radius* range will be known. Nodes need periodically "ping" neighboring nodes to make sure they are still alive. In order to reduce the overhead for transmitting the routing information, nodes in the overlay network use a soft state routing update, that is, periodically exchanging their resource info. At any given time, the resource routing information may potentially be stale or inconsistent. While, as we mentioned, the algorithm dose not need the routing table to be strictly correct, and the approximation will not affect the system's fidelity.

3.4.3. Query forwarding. If a node cannot match a query locally, it will choose a "right" neighbor to forward the query to. A query may be transferred by several hops until it arrives at the matching node or the TTL expires. In figure 4, node *C* receives a request for resource *x (1,3,10)*. It cannot find match in its local vector. Then it checks its neighbor vector *B* and finds $B_1=3, B_3=2, B_{10}=4$. As mentioned, the maximum value of B_1, B_3 and B_{10} represents the distance to resource *x*, thus *4* hops are needed to locate *x* through *B*. Then the request is forwarded to *B*. When *B* receives the query, it looks up its routing table and finds *3* different paths to *x*: path through neighbor *A* with *4* hops $(A_1=2, A_3=3, A_{10}=4)$; through *C* with *5* hops $(C_1=4, C_3=3, C_{10}=5)$; through *E* with *3* hops $(E_1=3, E_3=3, E_{10}=3)$. So the shortest distance to resource *x* is *3* and through neighbor *E*. Therefore the query is sent to *E*. Similarly, *E* forwards it to *D*. *D* finds match in its local vector, and then it checks its RDF database to further confirm it.

4. Experiments

To better understand the system's performance, we evaluate the performance of its three key components: the multi-attribute Bloom filter, the intra-cluster routing algorithm, and the inter-cluster routing algorithm. Then we integrate them as a complete system and test its efficiency and scalability.

To become good summarization, the Bloom filter index should be succinct while accurate enough. We measure the Bloom filter performance in the metric of false positive. In this experiment, there are 1000 resources. Every resource has at most 20 attributes. It is difficult to emulate the real query pattern. So we made up queries by randomly picking attributes from all possible values and concatenating them together. Half of the queries should be satisfied with the resources. MD5 is utilized as the hash function and the number of hush functions is 4. The maximal attribute concatenation length is 3. Figure 5 illustrates the relationship of false positive and Bloom filter size. From the graph, we see that there is a tradeoff between the false positive and the Bloom filter size: the larger the bitmap the lower the false positive rate. As long as the Bloom filter size is big enough, the percentage of false positive can be very low. Therefore, we can use Bloom filters to save storage with just slight risk of false positives.

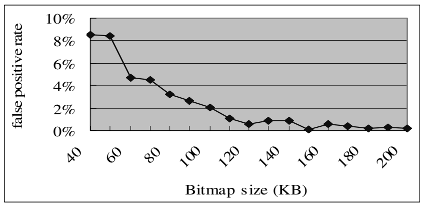


Figure 5. Percentage of false positive

To test the effectiveness and the cost of the routing scheme, we measured the intra-cluster routing and the inter-cluster routing respectively with simulation. The simulator models the physical network as a graph where each node corresponds to a peer. A number of RDF documents and a Bloom filter based routing table are associated with each node. The resources are randomly distributed to all nodes in the network with duplication rate 35%. Every node may have 1 to 100 resources. To make comparisons, we simulate our routing algorithms in conjunction with two well known algorithms: Gnutella flooding and Random Walk.

First we compare the intra-cluster tree routing with the other two routing algorithms in two important performance metrics: the number of messages and the routing hops to resolve a query. To guarantee every query can finally be satisfied， we set the query TTL as infinity. From the results in figure 6 and figure 7, it is clear that the intra-cluster tree routing algorithm has better performance on both metrics than either flooding or random work does.

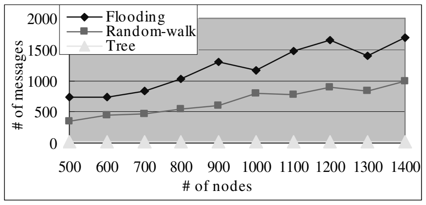


Figure 6. Intra-cluster routing: average number of messages per query

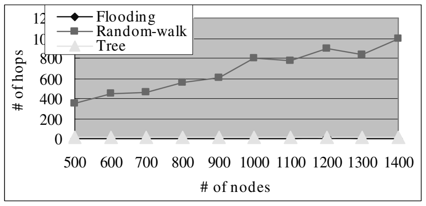


Figure 7. Intra-cluster routing: average number of hops per query

We also simulated the inter-cluster routing algorithm: the RDV routing algorithm. The *Radius* of the RDV algorithm is set to 3, and the average node degree (number of neighbors) is 6. The resource parameters are configured similar to the intra-cluster routing. Figure 8 compares the number of messages created to forward a query by each of the three routing algorithms. We can see RDV algorithm created much fewer messages than the other two algorithms. Figure 9 illustrates the relation of the query success rate with the query TTL.

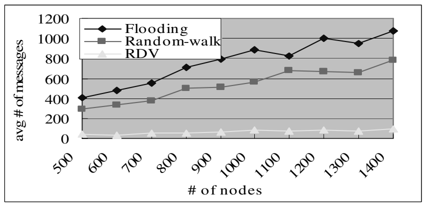


Figure 8. Inter-cluster routing: average number of messages per query

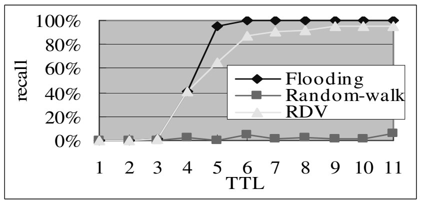


Figure 9. Inter-cluster routing: percentage of query success rate vs. TTL

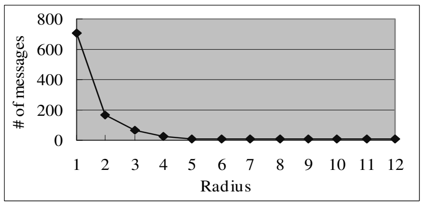


Figure 10. Influence of Radius

Figure 10 shows the influence of the *Radius* to the query overhead. Initially, increasing the *Radius* will increase the nodes' knowledge of the network, thus improving the query performance. When the *Radius* grows to 4, nodes almost have a complete knowledge of the network, then further increasing the *Radius* will not bring more benefit.

Finally, we integrate the two phases of the routing together and test them as a complete system. The network size is fixed to 2000 nodes. Nodes are randomly grouped into clusters and the size of a cluster is from 30 to 200 nodes. Figure 11 compares the query overhead to achieve certain successful hits by each of the three routing algorithms. Obviously, our hierarchical routing algorithm dramatically decreases the query overhead. In this experiment, we randomly cluster nodes. If we cluster them according to interest, the system can achieve better performance. Figure 12 compares the performance of different clustering strategies. It is clear that the interest-based clustering performs better than the random clustering. That is because nodes sharing the same interest are in the same cluster, then most of the queries can be satisfied within the cluster.

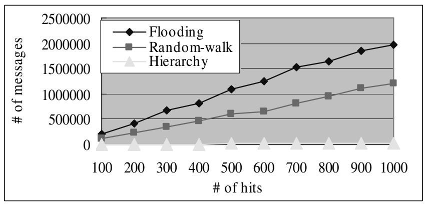


Figure 11. Hierarchical routing: number of messages vs. hits

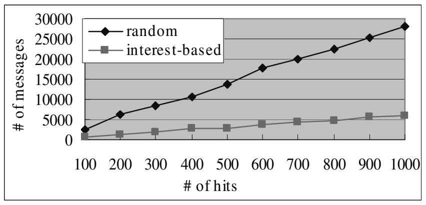


Figure 12. Influence of clustering method

5. Conclusion

As more and more resources appear in grids, there is an increasing need to discover these resources effectively and efficiently. In this paper, we present a novel design for resource discovery in large scale grids. It is based on the P2P model and it provides complex query interface. The system is designed to scale to large number of groups, large group size and to support complex resource query.

The system supports rich resource description and query by encoding the resource and query with RDF. Therefore, the resource providers can give resources better descriptions and the resource requesters can customize their requirements to make the query more powerful. To improve the system scalability, nodes are grouped into clusters. Two efficient routing algorithms: the intra-cluster routing and the inter-cluster routing are proposed. Both routing algorithms utilize Bloom filters as the basic data structure to aggregate resource information and help route the queries. The intelligent routing scheme is able to route queries to the nodes where the target resources are located, and to avoid flooding the queries to all other irrelevant nodes. The system has been evaluated by simulations. The experiment results prove that the routing schemes are efficient and scalable.

6. References

[1] I. Foster, C. Kesselman, and S.Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, 15(3), 2001

[2] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. "The physiology of the Grid: An Open Grid Services Architecture for distributed systems integration." *Technical report, Open Grid Services Architecture WG, Global Grid Forum*, 2002

[3] K. Globus Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing". *Proc. of 10th IEEE symposium on High Performance Distributed Computing* (2001).

[4] R. Raman, M. Livny, M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing". *Proc. of IEEE Intel. Symp. On High Performance Distributed Computing*, Chicago, USA (1998).

[5] Gnutella website. http://gnutella.wego.com/

[6]Ora Lassila and Ralph R. Swick, "W3C Resource Description framework (RDF) Model and Syntax Specification".

[7]Dan Brickley and R.V.Guha. "W3C Resource Description Framework (RDF) Schema Specification". http://www.w3.org/TR/1998/WD-rdf-schema/

[8] B.Bloom. "Space/time tradeoffs in hash coding with allowable errors". *Communications of the ACM,* pages 13(7):422-426, July 1970.

[9] I. Foster, C. Kesselman, "Globus: A Toolkit-based Grid Architecture". In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999

[10] S. Shi, Y. Guanwen, D. Wang, J. Yu, S. Qu and M. Chen "Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning". *Proc. Of the 3rd International Workshop on Peer-to-Peer Systems*, San Diego, CA, USA, February.

[11] The Napster protocol specification. http://opennap.sourceforge.net

[12] The FastTrack website. http://www.fasttrack.nu/

[13] B.Yang and H.Garcia-Molina, "Designing a Super-Peer Ntrwork", *Proc. 19th Int'l Conf. Data Engineering,* IEEEE Computer Society Press, Los Alamitos, CA, March 2003

[14] Morpheus home page, http://www.morpheus.com

[15] Kazaa website. http://www.kazaa.com/

[16] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," *Technical Report,* UCB/CSD-01-1141, April 2000.

[17] A. Rowstron and P. Druschel. "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Middleware, November 2001.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H.Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *ACM SIGCOMM*, August 2001, pp. 149-160.

[19] S. Ratnasamy, P.Francis, M.Handley, R.Karp, and S. Shenker. "A Scalable Content-Addressable Network," *ACM SIGCOMM*, August 2001, pp. 161-172.

[20] A. Iamnitchi, I. Foster, J. Weglarz, J. Nabrzyski, J. Schopf, and M. Stroinski, "A Peer-to-Peer Approach to Resource Location in Grid Enviroments", eds. *Grid Resource Management*, Kluwer Publishing, 2003.

[21] Reynolds, P. And A. Vahdat, "Efficient Peer-to-Peer Keyword Searching". *ACM/IFP/USENIX International Middleware Conference,* Rio De Janeiro, Brazil, June 2003

[22] M. Cai, M. Frank, J. Chen and P. Szekely, " MAAN: A Multi-Attribute Addressable Network for Grid Information Services" *The 4th International Workshop on Grid Computing*, 2003.

[23] Dublin Core metadata definition: http://dublincore.org/

[24] Michael T. Prinkey, "An Efficient Scheme for Query Processing on Peer-to-Peer Networks," Aeolus Research, Inc.