A Semantics-based Routing Scheme for Grid Resource Discovery

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

Abstract

Grid technologies enable the sharing of a wide variety of distributed resources. To fully utilize these resources, effective resource discovery mechanisms are necessities. However, the complicated and dynamic characteristics of grid resources make sharing and discovering them a challenging issue. In this paper, we propose a peer-to-peer (P2P) based overlay network to assist the efficient resource discovery and query. The framework is based on the RDF metadata infrastructure, allowing a rich and extensible description of resources. To avoid flooding the network with a query, we propose a comprehensive semantics-based query forwarding strategy, which only forwards queries to semantically related nodes. After the related nodes have been located, the original RDF query is used to do the final query and retrieval. Results from simulation experiments demonstrate that this architecture is scalable and efficient.

1. Introduction

The Information Service, one of the key services of grids, provides resource information to users. To make information available to users quickly and reliably, an effective and efficient resource discovery mechanism is crucial. However, grid resources are potentially very large in number and variety; individual resources are not centrally controlled, and they can enter and leave the grid systems at any time. For these reasons, resource discovery in large-scale grids can be very challenging.

Traditionally, resource discovery in grids is based mainly on centralized or hierarchical models. For example, in the Globus Toolkit [6], users can get a node's resource information by directly querying a server application running on that node, or by querying dedicated information servers that retrieve and publish an organization's resource information. Although interactions between these information servers are supported, a general-purpose decentralized service discovery mechanism is still missing.

To discover resources in more dynamic, large-scale, and distributed environments, P2P techniques have been used in grids. For example, [30] organizes information nodes into a flat unstructured P2P network and random-walk based methods are used for query forwarding. Random-walks are not efficient in response time for a very large system. [12] proposes a hierarchical structure to organize information nodes to reduce redundant messages. However, a well-defined hierarchy does not always exist, and the global hierarchy is hard to maintain in a dynamic environment. Another application [31] randomly groups information nodes into clusters to reduce the searching space, which unavoidably increases the overhead of publishing and updating of resources. Papers [22] and [23] present DHT-based multi-attribute resource discovery approaches, but these may incur either a high traffic load for result intersection or large overhead for multiple publication and update.

In this paper, we propose a semantics-based decentralized model for grid resource discovery. It uses RDF [4, 7] to represent both resources and queries. In the framework, resource providers register their resource information to local information nodes. Information nodes connect with each other, forming a P2P overlay. Resource searching is carried out only on top of this P2P overlay. Since we focus on the query routing over the information node overlay, we use the terms "peer" or "node" to refer to an information node. To support complex RDF queries without flooding the whole network, our system uses a Resource Distance Vector (RDV) routing algorithm. The principle behind RDV routing is to use the content of a query and the knowledge of the network to drive routing decisions. The basic idea is to extract the building blocks from RDF metadata and then summarize them to form a compact structure. Based on this summarization, we create a routing table to guide the query forwarding. RDV routing is only used as a hint to find matching nodes. After potential matching nodes have been

Proceedings of the First International Conference on e-Science and Grid Computing (e-Science'05)
0-7695-2448-6/05 $20.00 © 2005 IEEE

located, the original RDF query is then used to do the final query and retrieval. When compared to unstructured P2P applications oblivious of the resource location, this routing strategy reduces both the query overhead and query latency, and guarantees a higher query hit ratio. Compared with DHTs, our approach inherently supports rich queries, and requires no explicit control over the network topology or data placement. However, our system requires extra overhead for maintaining the routing table. Fortunately, the summarized routing index is lightweight and the traffic for maintaining the routing information is low.

The remainder of this paper is organized as follows. Section 2 describes the RDF resource representation, summarization, and query. Section 3 explains the semantic routing scheme. Section 4 gives the experimental results. Related work is discussed in Section 5. Section 6 concludes the paper.

2. Resource representation and queries

2.1. RDF metadata indexing

Metadata plays an important role for complex queries that go beyond string matching. We use an RDF metadata representation to encode resources. The benefit of using an RDF representation 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. With RDF representations, the resource providers can give resources detailed descriptions and the resource requesters can customize their requirements to make queries more precise and flexible.

```
<?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">
  <dc:title>Java programming</dc:title>
  <dc:creator>Ken Arnold</dc:creator>
  <dc:date>2002-09-01</dc:date>
  <dc:type>java programming language</dc:type>
  <dc:format>text/html</dc:format>
  <dc:language>en</dc:language>
</rdf:Description>
</rdf:RDF>
```

Figure 1. An example of RDF metadata

The basic building block of RDF is the triple which includes a subject, a predicate and an object. The subject is a resource about which the statement was made. The predicate is a resource representing the specific property in the statement. The object is the property value of the predicate in the statement, which can be either a resource identifier or a literal value. Figure 1 shows a fragment of the metadata of an electronic book. It uses the "Dublin Core" metadata definition [24]. One example of a triple in this metadata is: subject: *<"http://somewhere/Java programming">*, predicate: *creator*, and object: *"Ken Arnold."*

2.2. Index summarization

We utilize a metadata index to provide improved query capabilities, and to support more sophisticated query routing. Every peer maintains a resource index table, and peers exchange their indices. Queries can then be distributed by relaying based on these indices. However, exchanging RDF indices between nodes is almost impossible because each node may maintain a large number of resources. To reduce the overhead of propagating the index information, we must make the indices lightweight. Our strategy is to extract the subject, predicate and object from the RDF metadata and summarize them in a compact structure: a triple filter, which is based on Bloom filters [8].

Bloom filters use hash functions to transform a data set into a bitmap. Membership is tested by comparing the result of the hashing on the potential numbers to the vector. A triple filter includes three Bloom filters: the subject filter, the predicate filter, and the object filter. An RDF triple can be hashed to these three filters. For example, in Figure 2 the RDF triple mentioned above is hashed to a triple filter. In this example, the vector's size is eight bits, and three hash functions (h_1, h_2, h_3) are used to map an element to the vector. In reality the size of the vector is much larger, and the number of hash functions is always more.

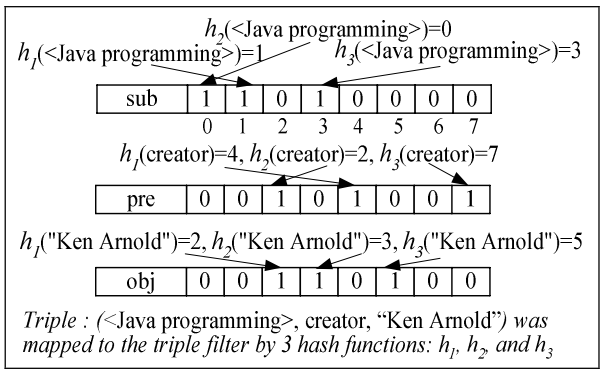


Figure 2. A triple filter example

Every node maintains a local triple filter and several aggregated neighbor triple filters. These filters form a routing table that directs query forwarding. The creation and maintenance of this routing table is discussed later in Section 3.2.

2.3. Query

Many query languages, such as RQL [25] and TRIPLE [26], have been developed for RDF. We use RDQL [27] to query the metadata. RDQL is a query language for RDF in Jena [28] models. It regards RDF as triple data without schema or ontology information unless explicitly included in the RDF source.

Some examples of RDQL queries include:

```
(1)SELECT ?x WHERE (?x, < dc:title >, "Java programming")
(2) SELECT ?x, ?types WHERE (?x, <dc:type>, ?types)
(3) SELECT ?author
     WHERE (?x, <dc:type >, "program language") ,
           (?x, <dc:author>, ?author)
```

Here, we don't cover how to resolve an RDQL query with a local RDF database, which is well understood for a local data store; we explain how to route the query to the best destination peers. It is straightforward to convert an RDQL query to a triple sequence. We then match the query triple sequence with the triple filters to determine where to forward the query. The idea is this: if the query can pass a node's filters, then it will be forwarded to that node. Consider a query *(?x, <dc:title>, "java programming")*, intended to find a resource with title *"java programming."* The resource we are looking for should have a predicate *"dc:title"* and an object *"java programming."* So if a node's predicate filter and object filter match them, the query will be forwarded to it. This filtering limits query routing traffic by forwarding queries only to a small number of related nodes. However, it cannot guarantee that the query can be answered through the forwarding path because the matched elements may belong to different resources. Matching the triple sequence with the filters relaxes the constraints of the original query. Nevertheless, an advantage of this scheme is that the filter can introduce only false positives but never false negatives – the correct nodes will not be excluded.

This design has several advantages: The storage space required is vastly reduced – compared to storing the entire RDF metadata – and it is much more efficient since queries need only to be matched with three small filters instead of large, complex RDF documents. The summarization also scales well as the number of triples in the RDF indices increases. However, these benefits are gained by sacrificing accuracy. Fortunately, strict accuracy is not necessary for the routing process, since the local RDF database is checked by the original query in the end. The resource summarization works only as a hint for forwarding the query to related nodes. Our experiments show that this lightweight routing index can effectively filter out a large number of unrelated nodes from the query. Since Bloom filters can only be applied to exact matches, they cannot be used for range queries. To perform range queries we can filter on other attributes and ran the range query on the RDF database of destination nodes.

3. Overlay routing

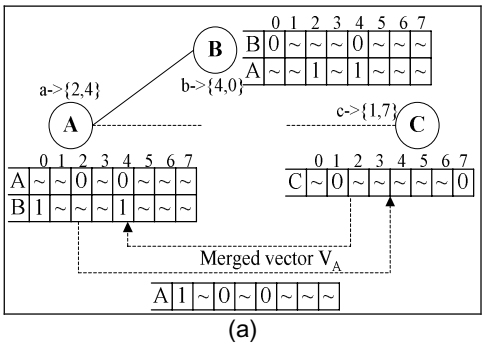
3.1. Overview

How to route over the overlay network is one of the central issues in determining the system's efficiency and scalability. We propose a so-called resource-distance-vector (RDV) routing algorithm. It uses a distance vector approach to route the query to the nearest matching nodes. The traditional distance vector approach is not scalable for locating unique nodes in an Internet-like network, but this modified version is extremely well suited for our resource discovery problem. Every peer in the overlay network maintains a resource index table. This table uses the triple filters we mentioned before, and includes distance (in number of hops) information. Peers exchange the resource indices with their neighbors, and update relevant entries in their table. The distance information is updated whenever passing through a node. To reduce false positives brought by the result of resource information aggregation, we set a hop count, which we call *radius*, to limit the number of hops the resource information can travel. When a node receives a query request, the algorithm chooses the shortest route to forward the query. So, if there is more than one provider supplying the same resource, then with high probability, the algorithm will forward the request to the nearest one. In addition, a "heuristic jump" method is used to expedite the searching process by skipping over the "barren" areas.

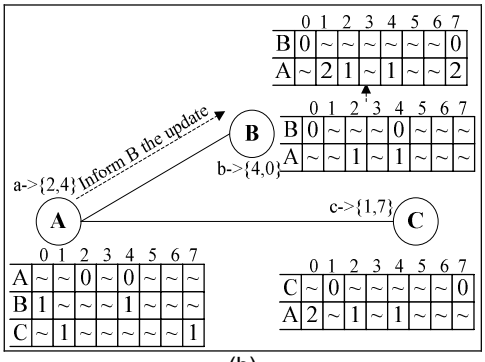
3.2. Routing table

As mentioned, each node maintains a RDV routing table (RDVT). The RDVT contains both local and neighbor triple filters. Besides resource information, the triple filters also record the distance to the resource. Figure 3 shows part of the network with the associated RDVT for each node. For brevity, only one of the three filters is shown here. Each element in the filter is associated with a distance number: the minimum

distance to a matching resource. The first row of the RDVT is the local filter containing local resource index. For example, node *A*'s local filter contains a local resource *a,* which is mapped to two positions *(2, 4)* in the filter. We set the distance number of a local resource as *0*. The rest of the rows represent resources accessible from neighbors. For example, in Figure 3(a), *A*'s second row contains resources that can be reached through the neighbor *B* (e.g., resource *b(4,0)* with *1* hop).



(a)



(b)

Figure 3. Maintaining routing indices

Figure 3 also illustrates the RDVT update process when a new node *C* joins the network. Node *C* joins the network by connecting to an existing node *A* in the network. After the connection is established, node *C* sends its resource indices to *A*. Similarly, *A* informs *C* of all the resources *A* has knowledge of. Specifically, *A* merges its local and neighbor vectors to one vector and sends it to *C*. The aggregation is done by comparing every element of the vectors, and selecting their minimum value. The merged vector of *A* represents resources accessible from *A* and their shortest distances to *A*. *A* does not need to send more information as *C* does not need to know the precise location of these resources, but only that they can be accessed through *A*. After *C* receives the merged vector from *A*, it adds *1* hop to each element of the vector, and adds an additional row in its RDVT (as shown in Figure 3b). After *A* receives *C*'s resource information and updates its routing table, it informs its neighbors (in this case, node *B*) of the update.

By exchanging the merged vector, we reduce both the amount of information transmitted and the storage used. Because this merging process chooses the minimum value of all vectors, hash positions related to a resource may have different values. It is not difficult to see that the maximum value represents the distance to that resource. For example, in *A*'s combined vector V_A in Figure 3 (a), to check a resource *b(4,0)*, we find in the related positions: $V_A(4)=0$, $V_A(0)=1$. So the distance from node *A* to resource *b* is *1*, the larger of the two values. We set a hop count limit, which we call *radius*, to limit how far the resource information can travel. In the merged vector, if an element's value equals *radius*, we reset the value to infinity ("~" in the figure), representing "not available."

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 knows about. Eventually the table stabilizes, and all resources within the range determined by *radius* are known. Nodes need to periodically "ping" their neighbors to make sure that they are still alive. To reduce the overhead of transmitting routing information, a soft-state update mechanism is used, in which routing information is exchanged periodically. At any given time, the resource routing information may potentially be stale or inconsistent, but as mentioned, this approximation will not affect the system's fidelity.

3.3. Query forwarding

This section illustrates how RDVT can be used to route queries. When a node receives a query, it converts the query into a triple sequence and matches the sequence in the RDVT. If enough matches are not found locally, the node chooses the "right" neighbors to forward the query to. A query may be transferred several hops until arriving at the matching node or the query TTL expires.

Figure 4 illustrates a query routing example. Like the previous example, we only show one of the three triple vectors. For simplicity, the query has only one constraint. The *radius* is set to *3*, so nodes are only aware of resources within 3 hops. In this example, node *A* receives a query for resource *e* (which is mapped to two positions: *3* and *6* in the filter). It checks its routing table and finds two matches: through *C* with *2* hops $(C_3=2, C_6=2)$ and through *D* with *3* hops $(D_3=3, D_6=3)$.

So the shortest distance to the resource is *2* through neighbor *C*. Therefore, the query is forwarded to *C*. Similarly, *C* forwards the query to *E*. *E* finds a match in its local vector, and then it checks the RDF database with the original RDQL query.

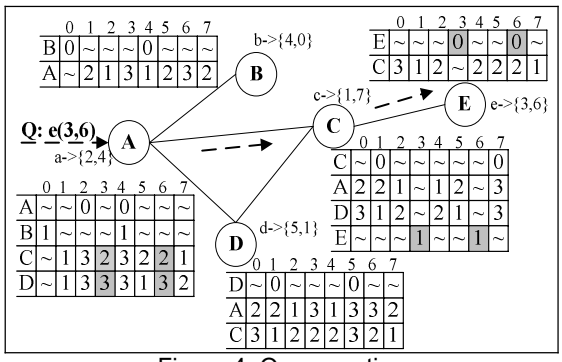


Figure 4. Query routing

Our routing algorithm works fine with networks containing cycles. Because of cycles, a node may receive a query multiple times. To avoid processing queries more than once, every query has a unique query ID and every node keeps a list of recently received query IDs. If a query has been received before, it will be discarded. Another benefit of recording the query ID is that it ensures the query does not hit the same false positive twice.

3.4. Heuristic jump and caching

By setting a *radius*, we limit the distance a node's resource information can travel. This reduces false positives, but at the same time, a node does not have global knowledge of the network but only a local view of the neighborhood. Because of this, a node may not find enough matches from its RDVT to forward queries. A naive solution is to forward the query to some random neighbors even if they have no match – hoping that these neighbors can find matches from their neighborhood. This method is inefficient since your neighbor has a neighborhood which largely overlaps your own. If the requested resources are scarce in the local area, forwarding the query to another neighbor in this area will not substantially increase the chance of resolving a query. To address this problem, we introduce a forwarding method called "heuristic jump."

This method allows the system to keep additional long-distance links as an addendum to the RDVT. When the RDVT cannot resolve the query, the query will "jump" to remote nodes the links point to. To discover those long-distance links, the system employs an aggressive caching technique. After finding the result of a query, the result travels along the reverse path to the requester. Whenever it is passed through a node, it is cached in that location. Every internal node caches the query, the destination node, and the distance to that node. We use caching to not only eliminate the need to forward a query which may be resolved locally, but also use this cached information as links for future long-distance jumps. During the query-forwarding process, when a node cannot find enough matches in its routing table, it chooses appropriate long-distance links from its cache and forwards the query accordingly. This expedites the searching process by jumping over barren areas. Candidate long-distance nodes should be located outside the neighborhood area; i.e., the distance should be greater than *radius*. In our heuristic, we also consider other metrics, for example, jump to nodes that answered more previous queries, or to nodes that answered similar queries. Our experiment in section 4.2 shows that forwarding by "heuristic jump" improves search efficiency.

4. Experiments

We performed extensive simulations to evaluate the performance of the routing scheme. In this section, we first describe our simulation methodology, and then present results for both static and dynamic network operations.

4.1. Simulation methodology

The topology of the network defines the number of nodes and how they are connected. In our model, we used BRITE [29], a well-known topology generator to create two kinds of network topologies: the random graph and the power-law graph. The resource set includes 10000 RDF triples (500 distinctive ones). We model the location of these resources using two distributions: the uniform distribution and a 70/30 biased distribution. Requesters are randomly chosen from the network. In order to simulate dynamic network behavior, we update resources, insert "on-line" nodes and remove active ones periodically. Arriving nodes start functioning without any prior knowledge. Our evaluation metrics are: (1) the recall rate which is defined as the number of results returned divided by the number of results actually available in the network; (2) the number of messages created to maintain the routing table and to resolve queries; (3) the number of hops to resolve a query.

In our experiments, some default values are used for the following arguments unless declared specifically: topology=power-law, resource distribution=uniform,

network size=2000, average node degree=5, number of walkers=5, TTL= 100 and *radius*=3.

4.2. Static experiments

Our goal with the static experiments is to examine the characteristics of RDV routing with a static network and to show its efficiency and scalability. To make comparisons, we simulate RDV in conjunction with learning-based routing [14, 20], which routes queries to neighbors according to past experience. We deploy 5 walkers for both routing algorithms: the original requesting node forwards the query to 5 neighbors, while the rest of the nodes forward only to 1 neighbor.

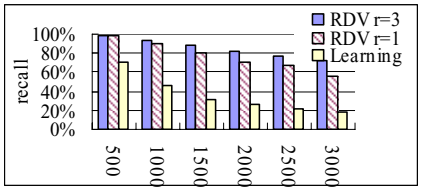


Figure 5. Recall versus network size

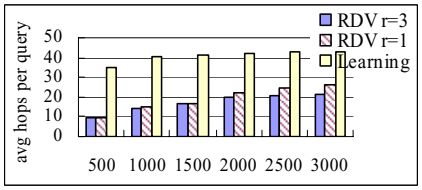


Figure 6. Hops versus network size

Figures 5 and 6 compare RDV (with *radius*=1, and 3 respectively) with learning-based forwarding in terms of query recall rate and query hop consumption. The network size increases from 500 to 3000. As expected, our RDV routing algorithm outperforms learning-based routing on both metrics. RDV always forwards the query to the right direction, so it can find more results and thus enjoy a higher recall. In addition, RDVT records the resource distance information, so it can forward queries to the nearest resource providers. That's why RDV needs fewer hops to resolve a query. Another observation is that RDV with *radius=1* achieves pretty good performance. When *radius=1*, the RDV routing becomes very simple: nodes only exchange local resource index with neighbors, and they do not need to manage the index aggregation. Under this condition, the system can save lots of computing power for the routing, but the routing accuracy will be affected a little bit. This scheme fits for systems having more concerns for the simplicity than the accuracy.

Figure 7 illustrates the relationship of the query recall rate with the query TTL. Note that the recall rate is achieved by only 5 walkers – increasing the number of walkers will increase the recall rate under certain TTL.

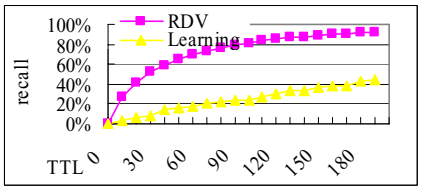


Figure 7. Recall versus TTL

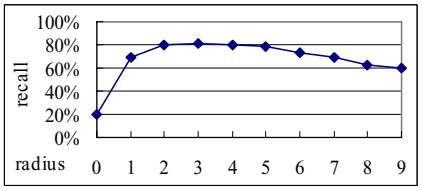


Figure 8. Recall versus radius

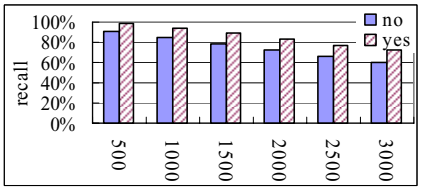


Figure 9. Effect of heuristic jump

How far the resource information propagates determines how much a node learns about the network. Figure 8 shows the influence of the routing *radius* on query recall. When *radius=0*, the RDV algorithm degrades to a random walk algorithm. Initially, increasing the *radius* increases the nodes' knowledge of the network, thus improving the query performance. When the *radius* grows to three, nodes have a good knowledge of the network; further increasing the *radius* does not bring more benefit. On the contrary, that deteriorates the recall rate because the large amount of resource index aggregation causes more false positives of the Bloom filters.

Figure 9 compares the performance of routing with and without using the "heuristic jump." We can see "heuristic jump" improves the overall recall rate.

4.3. Dynamic experiments

In this section, we present simulation results for a changing network to show that our routing scheme is robust and effective under this situation. The dynamic network behaviors are simulated like this: in every unit simulation time, an active node has a twenty percent possibility to create a query, one percent possibility to update its resources, and one percent possibility to

leave the system. Some offline nodes, whose number is the same as the leaving nodes, join the system and start functioning without any prior knowledge. Nodes propagate their updated routing information (if any) every three unit time.

Figure 10 illustrates the recall rate in the dynamic environment mentioned above. We can see that RDV continues to exhibit a much better recall rate than the learning-based routing. According to Figure 11, the recall rate in the dynamic environment shows only a small decrease, compared with the static environment.

Figure 12 compares the aggregate overhead of routing table update and query during the whole simulation period. In this experiment, the update message was propagated in the format of compressed full-table update. But in practice, a system may have two types of updates: full-table updates and incremental updates. The latter would occur more frequently and its size is much smaller, therefore the update overhead can be even smaller.

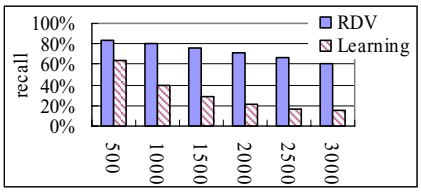


Figure 10. Recall rate versus network size

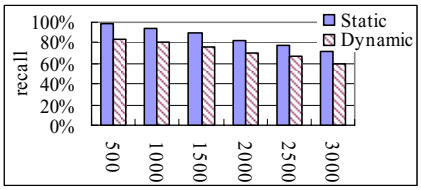


Figure 11. Static and dynamic recall rate

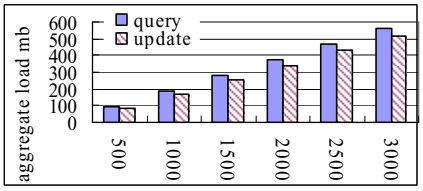


Figure 12. Update overhead and query overhead

5. Related work

Many recent P2P-based searching techniques relate to our research. Flooding is the predominant search method in unstructured P2P networks. This method, though simple, does not scale well in terms of message overhead. There have been numerous attempts [1, 2, 3] to enhance its scalability. Random Walks [14, 20] are an alternative to flooding for unstructured searches. They can reduce the amount of network traffic, but it is at the cost of query latency. The learning-based forwarding used in our experiments, in fact, is an advanced version of random-work. Recently, hierarchical super-peer systems [13] have been proposed to improve searching efficiency.

DHTs [16–19] have received a lot of attention in the last few years. These systems have been shown to be scalable and efficient. However, a missing feature of DHTs is keyword searching and support for more advanced queries. Another hurdle to DHTs deployment is their tight control of both data placement and network topology, which makes them more sensitive to failures, and difficult to keep the content and path locality [9].

More recently, a few studies [10, 21, 22] extend the DHT scheme to support keywords or multi-attribute queries. The basic idea is to map each keyword to a key. A query with multiple keywords then uses the DHT to lookup each keyword and returns the intersection. In order to do that, large amounts of data must be transferred from one peer to another, and the traffic load may be high [6]. Systems like [23] avoid this multiple lookup and intersection by storing a complete keyword list of an object on each node, but this may incur more overhead on publishing and storing the keywords.

Some applications have used RDF to represent resources and queries. Edutella [5] is a well known example. It uses a super-peer structure. To resolve a query, it broadcasts the query to the super peer overlay network. Cai et al. propose a DHT-based P2P architecture called RDFPeers [11], which maps RDF triples to the Chord overlay. This approach, however, suffers from the same inherent shortcomings of the DHTs mentioned previously.

Bloom filters have been used as a succinct summary technique for query filtering and routing. For example, OceanStore [32] uses attenuated Bloom filters to store objects information. PlanetP [15] also uses Bloom filters to distribute a summary of the contents of each peer.

6. Conclusion

As more and more resources appear in grids, there is a compelling need to find an effective and efficient way to discover and query these resources. In this paper, we present a novel design for resource discovery in large-scale grids. The system is based on the P2P model and provides a complex query interface. It supports rich resource descriptions and complex queries by encoding resources and queries with RDF. To avoid flooding

queries to irrelevant nodes, a semantics-based routing scheme is proposed to route queries only to related nodes. This system has been evaluated by a group of simulations, which show that the proposed routing schemes are both efficient and scalable.

7. References

[1] Chawathe, Y., Ratnasam, S., Breslau, L. Lanhan, N. Shenker, S. "Making Gnutella-like P2P Systems Scalable", In *Proceedings of ACM SIGCOMM'03*.

[2] B.Yang, H.Garcia-Molina, "Efficient search in peer-to-peer networks", *Proc. of CDCS'02*, Vienna, Austria, July 2002

[3] Banaei-Kashani, F. and C. Shahabi. "Criticality-based analysis and design of unstructured peer-to-peer networks as complex systems." *Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 351-358.

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

[5] W. Nejdl, M. Wolpers, W. Siberski, A. Loser, I. Bruckhorst, M. Schlosser, and C. Schmitz. "Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks." *In Proc. of the Twelfth International World Wide Web Conference,* 2003.

[6] Globus Toolkit: http://www.globus.org/toolkit/

[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,* 1970.

[9]N. J.A. Harvey, M.B. Jones, S. Saroiu, M. Theimer, and A. Wolman. "SkipNet:A Scalable Overlay Network with Practical Locality Properties." In *Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems* (USITS '03), Mar. 2003.

[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] M. Cai and M. Frank. RDFPeers: "A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network." *In International World Wide Web Conference (WWW)*, 2004.

[12] H. Lican, W. Zhaohui, and P. Yunhe. "A scalable and effective architecture for Grid Services discovery." *In Proc.of the First Workshop on Semantics in Peer-to-Peer and Grid Computing*, 2003.

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

[14] Lv, C., Cao, P., Cohen, E., Li, K., Shenker, S. "Search and replication in unstructured peer-to-peer networks." In: *ACM, SIGMETRICS* 2002.

[15] F. M. Cuenca-Acuna, C. Peery,R. P. Martin, andT. D. Nguyen. "PlanetP: Infrastructure Support for P2P Information Sharing", *Technical Report Department of Computer Science, Rutgers University, Nov. 2001*.

[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*, 2001

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

[20] Adamic, L., Huberman, B., Lukose, R., Puniyani, A.: "Search in power law networks." *Physical Review* (2001)

[21] P.Reynolds and A. Vahdat. "Efficient Peer-to-Peer Keyword Searching." *In Proceedings of ACM/IFIP/USENIX Middleware*, 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] C.Tang and S.Dwarkadas. "Hybrid Gloablal-Local Indexing for Efficient Peer-to-Peer Information Retrieval." *In Proceedings of USENIX NSDI,* March 2004.

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

[25] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. "RQL: A Declarative Query Language for RDF." *In Proceedings of The Eleventh International World Wide Web Conference* (WWW'02), Hawaii, May 2002.

[26] Michael Sintek and Stefan Decker. "TRIPLE - An RDF query, inference, and transformation language." *In Deductive Databases and Knowledge Management*, October 2001.

[27] A. Seaborne. "RDQL: A Data Oriented Query Language for RDF Models." www-uk.hpl.hp.com/people/afs/RDQL/, 2001.

[28] Brian McBride. "Jena: Implementing the RDFmodel and syntax specification." *Technical report*, Hewlett Packard Laboratories, Bristol, UK, 2000.

[29] A.Medina, A. Lakhina, I.Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," *Proc. The International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems-* MASCOTS, Cincinnati, Ohio, August 2001.

[30] Iamnitchi A, Foster I, "On Fully Decentralized Resource Discovery in Grid Environments", *Proc. The 2nd IEEE/ACM International Workshop on Grid Computing* 2001, Denver, November 2001.

[31] Chander, A., S.Dawson, P.Lincoln and D.Stringer-Calvert. "NEVRLATE: Scalable Resource Discovery." *Proc. Second IEEE/ACM internation Symposium on CLuster Computing and the Grid (CCGRID2002).*

[32] S. C. Rhea and J. Kubiatowicz, "Probabilistic location and routing," *in Proc. INFOCOM, vol. 3,* New York, NY, June 2002, pp. 1248–1257.