Sharing Supply-Demand Trends for Efficient Resource Discovery in GRID Environments

N.Antonopoulos, G.Exarchakos and K.Zhang

Department of Computing, University of Surrey, Guildford, Surrey, United Kingdom
e-mail: { n.antonopoulos, g.exarchakos, k.zhang}@surrey.ac.uk

Abstract

A critical factor for the success or otherwise of any GRID-oriented environment is the quality of tools it offers to its users to enable them to discover pertinent information and resources efficiently and accurately. The peer-to-peer model of interaction and the sheer number of resources makes the design of a GRID information/resource discovery service more challenging compared to similar services for traditional client-server environments. In this article we present the architecture of the Grid Talk (GT) agent; a pure peer-to-peer framework to facilitate resource discovery in GRID environments. In contrast to current approaches that introduce separate, inter-linked, dedicated nodes for resource discovery, our proposed framework enforces the direct interaction between requesters and providers through the sharing of supply-demand trends. The absence of any hierarchical or web-like middleware implies that our framework is firstly more efficient since queries are sent directly to appropriate providers and secondly more adaptive because any changes to the status or capacity of the providers are reported directly to the consumers through peer-to-peer table sharing.

Keywords

Grid, Peer-to-Peer, Resource discovery, Cooperation.

1. Introduction

A GRID can be thought of as a large-scale, dynamic, distributed system that manages a large volume of both software and hardware resources which belong to a mixture of participants ranging from individual end users to large-scale international organisations and companies. Regardless of their specialisation, the main concept behind any GRID environment is the efficient and collaborative use of large volumes of physically distributed resources through *peer-to-peer* interactions between users and user programs, as exemplified by the activities of the Global Grid Forum (GGF) (Global Grid Forum, 2005).

A critical factor for the success or otherwise of any GRID-oriented environment is the quality of tools it offers to its users to enable them to discover pertinent information and resources efficiently and accurately. This statement is consistent with the current GGF Open Grid Services Architecture (OGSA), which consider the process of resource discovery central to any GRID infrastructure (Tuecke *et al*, 2002). The peer-to-peer model of interaction and the sheer number of resources makes the design of a GRID information/resource discovery service more challenging compared to similar services for traditional client-server environments. Any approach to this challenge needs to take into consideration the underlying principles of GRID computing in general and as such be *scalable*, *adaptive*, *efficient* and *fault-tolerant*.

The remainder of this article is organised as follows: Section 2 provides a comprehensive survey of relevant state-of-the-art approaches and systems. Section 3 outlines the main concepts of our proposed peer-to-peer framework for resource discovery. Sections 4 and 5 show the conceptual architecture and functionality of the framework. Section 6 shows computer simulations carried out to demonstrate the framework's behaviour in terms of network traffic and finally section 7 draws the conclusions and identifies possible avenues for further work.

2. Approaches to Resource Discovery

The current literature suggests that there are three main approaches that can be or have been used to support resource discovery in GRIDs:

- *Matchmaker*. In this approach, the discovery system accepts user queries, matches them against a centralised or distributed, inter-linked set of currently known resource profiles (typically expressed in the same notation as the user queries) and reports any found matches back to the requesting user. The user is then responsible to choose the most appropriate resource and use it accordingly. A characteristic example of a system based on this approach is the CORBA object trader service (Orfali *et al*, 1997).

- *Broker*. This is an extension of the matchmaker approach whereby the discovery system not only locates the appropriate resources but also uses them according to the user specifications (queries) and returns the final results to the query originator (i.e. the actual requested files, output from an executed program, etc.). An example of such a system is the JXTA discovery service by Sun Microsystems (Waterhouse *et al*, 2002). JXTA search hubs form a web of specialised brokers, which is traversed by user queries until the appropriate resources are located and used (programs) or returned (data).

- *Content routing*. In this approach, an administrator creates a summary of the contents/capabilities of every resource to indicate what queries the specific resource can satisfy. These profiles are then submitted to the leaves of a rigid, centrally managed and updated hierarchy of organisation nodes (brokers). Each node processes the profile and forwards it in abbreviated form to the nodes one layer up the hierarchy. In this way, the nodes of layer n can utilise the profiles of the nodes in layer n+1, process them and then forward aggregated profiles to the nodes of layer n–1 (Sheldon *et al*, 1995). The user queries are automatically forwarded down the hierarchy until they reach the server where the requested resource is physically located. Routing Indices is a variation of this approach (Crespo *et al*, 2002). In this context each peer gets aggregated profiles about the resources offered by all other neighbour peers. These are then ranked on a category basis and are then forwarded to other peers, which perform exactly the same function, etc. In this way every peer forms a hierarchy for every topic.

From the above analysis we can observe that the first two approaches rely on separate, dedicated, inter-linked nodes for discovering resources pertinent to a user query. These nodes collectively constitute a form of middleware, which acts as a single virtual server for resource discovery requests. This method scales well and can provide accurate results (subject to the query-profile matching algorithm) at the cost of adaptivity, efficiency and fault-tolerance. The end users rely on this middleware to discover pertinent resources on their behalf and as a result if it is not operational the end users cannot retrieve the required resources even if the resource provider peers are operating normally. In addition each discovery request needs to go through a number of matchmakers/brokers until it reaches its final recipient and as a result an increase in network traffic and overall latency is unavoidable. Naturally there is a linear relationship between these increases and the depth of the organisational hierarchy the queries traverse. The same relationship exists in the context of adaptivity as well, since any changes to the number, nature and availability of the GRID resources will need to propagate upwards in the organisational hierarchy before it becomes visible to the consumer peers.

The routing indices approach forms resource organisational hierarchies on a pure peer-to-peer manner. Although it doesn't rely on any kind of additional middleware, the fact that peers are organised into hierarchies means that this approach has a similar effect on adaptivity, efficiency and fault-tolerance as the aforementioned two. In addition, this type of hierarchies can introduce multiple paths of different lengths to the same resource. As a result *n* more steps, and thus *n* more messages, may be needed to retrieve a resource that could be accessed in a single step. Furthermore, each peer will have to dedicate part of its processing power to match and forward queries for which it is neither the requester nor the provider.

3. Sharing Supply-Demand Developments

As opposed to the current approaches to resource discovery in GRID environments where the aim is to index all the available resources, we propose the idea that the discovery system should enable the sharing of supply and demand trends instead. We believe that this approach will keep each peer informed with an up-to-date view of the capacity of its community customised to its own interests and requirements. A practical and simple way to share such supply-demand knowledge is through resource query sharing between peers. Collective queries of resources from a number of peers can represent a supply-demand trend between them. The supply-demand developments are shared by a combined process of ranking peer GRID participants and sharing the ranks with other peers. The ranking will determine the ability of a peer to know *what* is available *where* and the sharing to forecast what is *new* and what *has changed*.

This method in fact constitutes an adaptation and novel application of well-established principles and methods from the area of network routing. In our view there is a strong analogy between routers in computer networks and peers in GRIDs and as such we have re-used the principle of peer-to-peer routing table sharing between neighbour routers as it is found in typical distance vector routing algorithms like RIP. Our main modification to this method is that each user query is mapped directly to a ranked set of providers rather than indirectly via a path of intermediaries (as is the case with routers).

Based on the above principles, we have designed a novel system called the GridTalk (GT) agent; an automated peer-to-peer framework to support efficient resource discovery in GRIDs. Typical environments where this framework could be applied range from simple file sharing and to more complex service-oriented business computing systems.

Following widely adapted standards and to support machine readability we have chosen the Resource Description Framework (RDF) (World Wide Web Consortium, 2002) for representing resources. RDF is a W3C recommended formal notation for representing Web resources. With the OGSA's proposal for integrating GRID and Web Services, RDF can also be used to represent both Web and GRID resources. To represent RDF queries, the Hewlett Packard's Jena Toolkit (Seaborne, 2002) is used as it has evolved from previously functional activities such as SquishQL. An example using RDQL on our framework can be shown as follows.

Consider an RDF fragment representing a remote participating peer named *Turing*.

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:gt='http://description.org/schema/GRIDpeers'>
  <rdf:Description about='http://URI/Turing'>
        <gt:Name>Turing</gt:Name>
        <gt:MemoryShare>100Mb</gt:MemoryShare>
        <gt:SpaceShare>2Gb</gt:SpaceShare>
  </rdf:Description>
</rdf:RDF>
```

The RDQL query syntax:

```
SELECT ?x
FROM <http://URI/gPeers.rdf>
WHERE (?x, <http://description.org/schema/GRIDpeers#Name>, "Turing")
USING gt FOR <http://description.org/schema/GRIDpeers>, rdf FOR
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Then the output is,

```
x
=====================================================
<http://URI/Turing>
```

In this example, *gPeers.rdf* is the data source which uses the namespace *http://description.org/schema/GRIDpeers*. The user requests a contact of a peer named *Turing* by specifying the name of a remote machine.

4. The Framework

Traditionally, a node, hub or a broker exists in the system to keep track of the participating peers and these nodes handle the majority of the interaction between these peers. We focus on bypassing this process of relying on brokers and grant this function to the edge participants (i.e. peers) themselves and the benefit is an independent system where all the participating peers are self-reliant. The following two proposed concepts support this functionality:

Member Ranking: *The participating peers must be kept on a map. This map should be kept updated periodically to trace the availability of peer resources. This map is a record of peers and their capacities (9).*
Query Knowledge Sharing: *The supply and demand trends in the organisation indicate the exchange of resources in the system. The "who can provide what and when" notion.*

The functionality of the framework can be summarised as shown in Figure 1. *Virtual Overlapped Groups* are agents and their groups. Every agent in a group has its own specific group or groups that it is part of. In Figure 1, only a few groups are shown for the simplicity of presentation. For example, the two agents in Group E will have their own specific groups and still are part of other groups, Groups A and C.

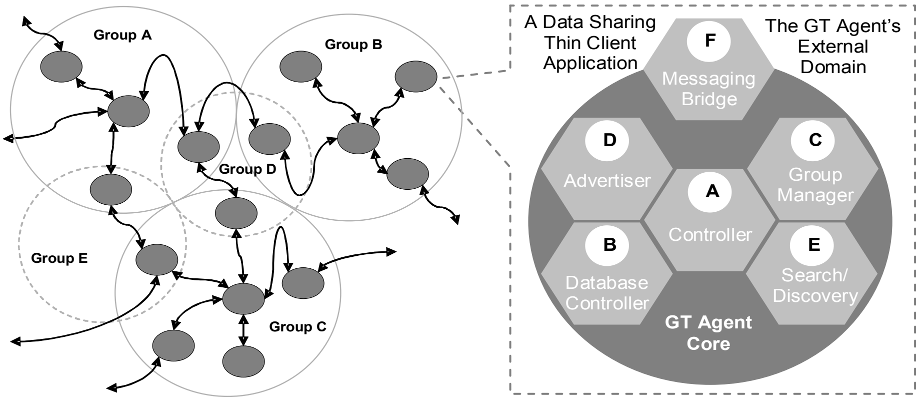


Figure 1: Virtual Overlapped Groups

Every GT agent functions as a gateway to GRID like applications in the organisation. This means that the GT agent acts as an interface to external heterogeneous applications on behalf of the owning application (i.e. as a gateway) enabling them to communicate to each other. Figure 1 also shows the enlarged logical view of a GT agent, modelling the above stated requirements with the appropriate representative components:

Controller: *A* takes care of the internal automation processes and controls other GT components.
Database Controller: *B* stores member data, policy rules, and other control information.
Group Manager: *C* maintains the profile of members and monitors member behaviour. C also controls policies of individual members depending on their status. Groups are managed dynamically and it is transparent to the user.
Advertiser: *D* is an active process, which broadcasts the owner's existence and capacity.
Search/Discovery Unit: *E* is responsible for broadcasting and querying other members for resources.
Messaging Bridge: *F* is the gateway to the GT and is a dual thread process to handle external and the owning application's requests and responses.

The main features of the GT agent architecture are as follows:

1. Group(s) per GT agent: Each peer manages its own group (i.e. a collection of other peers). A peer may also be part of various disciplines, hence it can be a member of other groups. The owner decides whom to invite or whom to grant access to the group. If the owner of the group requires a service, either a broadcast can be sent out or the GT can seek already registered members. The first point of contact will be the registered members and, if they cannot provide the requirements, a broadcast is sent out.

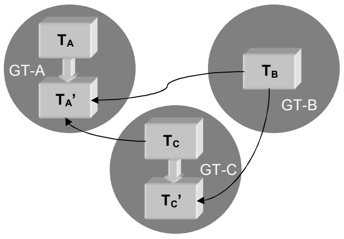
2. Policy management by the originating GRID: As opposed to hierarchical management, peers manage their own rules of interaction. As every group is based on individual peers, policies will be bi-directional, i.e. each of the participating GRID peers will hold its own policies during the interaction.

3. Total peer-to-peer interaction: Peers contact each other directly and not through a route of mediators. A broadcast must be sent out when a GT agent first comes into existence. Each GT agent gradually builds up contacts as it becomes more functional in its environment.

5. Query Knowledge Sharing

n the domain of GT agents, we concentrate on the sharing of query information (i.e. as processed data, as knowledge or as metadata queries) with registered members. This implies the sharing of query rank tables so that each member can modify its own rank table based on other member's tables.

As the Figure 2 illustrates, GT *A*, *B* and *C* have query tables T_A, T_B and T_C respectively. Provided that GT *B* and *C* are members of *A*, query tables of T_B and T_C together with T_A will provide GT *A* with a more up-to-date query table T_A'. The same scenario may also apply to *B* or *C*. This method of sharing results in an *environment aware community* and therefore in more *accurate matching of queries to resources*.



$$T_A + T_B + T_C \Rightarrow T_A'; \quad B \in A,\ C \in A$$

$$T_C + T_B \Rightarrow T_C'; \quad B \in C$$

Figure 2: Sharing of Query Rank Tables

I To merge two query tables, we match each entry of the first table with the contents of the second. Each query table entry consists of a query specification (expressed in the RDF Query language) and an associated ranked set of relevant resource providers. More details on how query tables are constructed and updated can be found in (Virupatchan *et al*, 2002). The merging process depends on the matching of query specifications. Queries are compared against each other with respect to specific RDF Query tags. Matching is performed in the following order.

Namespace: *Two queries to be matched should have the same namespace. The namespace is a dictionary of RDF and RDF Query elements and attributes.*
Select: *Values to be returned by the query*
From: *The RDF source file, specifying resources.*
Condition: *The condition for selecting resources.*

Each RDF Query must specify a source collection and may specify a *Property* name or a *Condition* for resource retrieval. The query table separates a query by its *Source collection*, *Object* (i.e. *Property*) and *Condition*. The precedence is given in the order of *Object*, *Source* and *Ranking* respectively. The Condition is included for clarity purposes (i.e. specifying the condition may restrict the retrieval of the match because the *Condition* is for more specific matches). *Condition* is checked in the first instance upon a formation of a request for potential exact matches. This can be clarified as follows: *who* (i.e. a member GT) requires or supplies *what* (i.e. *Object* data) is of more importance than *how* (i.e. *Condition* of the request) the request is made or supplied (i.e. the weight of *who* and *what* in this context is higher than the weight of *how*). Knowing *how* is an added bonus but is not of any strong advantage. The rules for query tables merge are as follows:

<u>Rule 1:</u> Object: *If the Object of the query being shared is required by the owner (i.e. the peer which requests the sharing), but is from a different Source (i.e. a namespace not used by this owner) then a <u>new entry</u> is made regardless of the Condition or the Ranking (i.e. here we give precedence to the actual Object, as knowing a new Source for the required Object is an addition to a GT's knowledge repository).*

<u>Rule 2:</u> Source: *If the Object of the query being shared is not required by the owner, but is from the same Source (i.e. a namespace used by this owner) then a <u>new entry</u> is made regardless of the Condition or the Ranking (i.e. here we give precedence to a potential Source collection, as knowing new Objects from a required Source is an addition to a GT's knowledge repository).*

Rule 3: Rank:

a. *If the Object and the Source of the query being shared are not required by the owner and if the Ranking is high then a new entry is made. (i.e. it is extremely valuable to know who supplies what in the community regardless of instantaneous requirements).*

b. *If the Object of the query being shared is required by the owner and is from the same Source then a merge is performed.*

When similar queries exist on member tables, the ranked peers in the tables for that specific query have to be sorted before being committed into the owner GT's table. For sorting ranked peers, two attributes of a peer are required to be compared: the ranking and the number of responses provided by that specific peer.

The following example explains the process of a table merge. Consider member rank tables, T_A, T_B and T_C with queries, Q_1, Q_2 and Q_3 which are identical with respect to the *Object* data and the *Source* collection. Members contributing to these queries (i.e. Q_1, Q_2 and Q_3) in the ranked order are:

$$T_A = Q_1 \{GT_A, GT_B, GT_C, GT_H\}$$
$$T_B = Q_2 \{GT_M, GT_G, GT_A, GT_H, GT_C\}$$
$$T_C = Q_3 \{GT_B, GT_N, GT_A\}$$

The comparisons are done separately for responses and ranks. If the ranks are the same then the number of responses is considered for ordering the members. Responses provide a way to measure the long-term reliability and capability of a member. The comparison can be done as shown below:

Comparison based on ranking	Comparison based on responses
$Q_{Compare\ Ranks}$ {	$Q_{Compare\ Responses}$ {
$\sum(Rank\ GT_A)/3,$	$\sum(Responses\ GT_A)/3,$
$\sum(Rank\ GT_B)/2,$	$\sum(Responses\ GT_B)/2,$
$\sum(Rank\ GT_C)/2,$	$\sum(Responses\ GT_C)/2,$
$\sum(Rank\ GT_H)/2,$	$\sum(Responses\ GT_H)/2,$
$Rank\ GT_M,$	$Responses\ GT_M,$
$Rank\ GT_G,$	$Responses\ GT_G,$
$Rank\ GT_N,$	$Responses\ GT_N,$
}	}

Thus the newly constructed query as a result of a merge, Q_{Merger} is given as follows:

$$Q_{Merger} = Sorted\ \{\ Q_{Compare\ Ranks}\ ,\ Q_{Compare\ Responses}\ \}$$

This Q_{Merger} will be a new entry in the owner GT's table for *Rules, 1 and 2* or a modification to the existing query, $T_A(Q_1)$ for *Rule 3.b.*

6. Evaluation

Our assessment illustrates the following functionalities:

1. Reduction of multicast traffic in a peer-to-peer environment.
2. Improved accuracy on routing queries due the ranking of members.
3. The average number of members that a GT agent should have in order to realise all the other GTs in the community (i.e. since this number is important to share a member's knowledge on the supply and demand of resources).

6.1. Reduction in Multicast Messages

To illustrate that the multicast messages sent out by a GT reduce over time, we simulate this setup by defining the following parameters: number of GT agents, probability of requests from a GT agent, average number of resources per GT agent, probability of responses from external GT agents, probability of responder also wishing to become a member and the probability of responses from member GT agents.

As time increases, a GT agent's members will gradually increase and the probability of responses by members will increase. At the same time, the probability of external GT agents promising to become members will decrease due to the reduction of external non-members in the environment.

The outcome is shown in Figure 3.a, initially for 200 GT agents, with an average of 50 resources, 20 members and a probability of requests of 0.5.

6.2. Member Ranking

Member ranking is the process where members are weighed and labelled based on the potential of answering a particular query. The concept of "*who knows what*" in the community is understood by this process. The subject of ranking members is demonstrated in this section and the average number of messages sent out by a GT agent to its members over a period of time is calculated.

In order to conduct this simulation, we have constructed a model community of 10 GT agents. Here, the measurements are with respect to a GT agent with 10 connected members. The ranking is simulated as follows: In the event of a response over a request query, the rank of the responders are calculated and reordered. These ranks are used when routing another similar query to potential members. In the event of another similar request, the average of the member ranks are considered for routing these requests. Only the GTs with a rank higher than the average of the member ranks are selected for answering new queries of that type.

Figure 3.b illustrates a scenario of a GT agent sending 100 separate queries to its members. The plot shows the number of request messages sent out until each query is successfully answered. In that figure, the three prominent readings are the maximum number of requests, 10, the average number of requests, 5, and the minimum number of requests, 1. The maximum reading represents a general multicast to all the members. The multicasting occurs when there are no potential members from the ranking for answering a particular request and the request has to be announced to all the members. The minimum reading represents a targeted routing to a specific member for answering a request. The average reading represents the average number of messages required to be sent out based on the member rankings. This simulation shows that in a community of 10 GTs the multicasting is reduced since the average number of messages is 5 rather than 10.

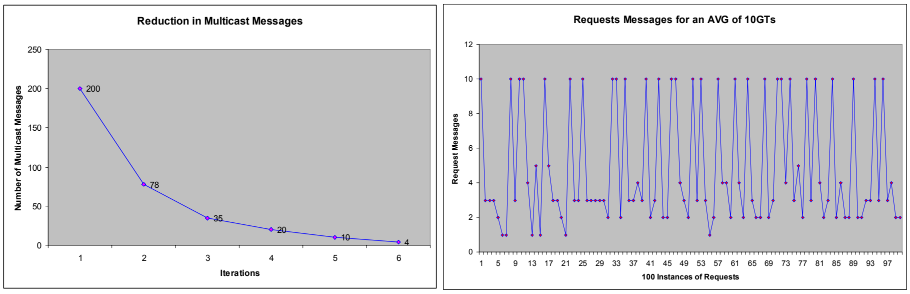


Figure 3: a) Multicasting Behaviour & b)Request Messages for 100 Instances of Request

6.3. Query Table Sharing

Query table sharing is the process by which GTs in a community exchange their request information (i.e. resource queries) with their members. The aim of this simulation is to calculate the average number of query table sharing required for a GT in the community to realise the available number resources.

This simulation again assumes a community of 10 GTs each containing an average of 10 resources. Here, we vary the number of members contained by GTs to estimate the average time required by any one GT to realise all the available resources. The reading will provide the minimum number of members required by a GT agent to realise all the resources owned by the remaining 9 GTs in the community. The greater the number of members, the less sharings will be required by each GT agent. If the number of members is low, the number of sharings will increase and will reach a saturation point where the sharing cannot continue further and the resources in the community cannot be fully realised.

Figure 4 shows a scenario of query table sharing by 10 GTs with an average of 10 (—□—) and 5 (—Δ—) members per GT. The simulation is executed for 100 snapshot instances.

In the 10 member scenario, a GT only requires an average of 5 table sharing with its members to realise all the resources contained in the community. Here, the maximum number of sharing is 9 and the minimum number of sharing is 2.

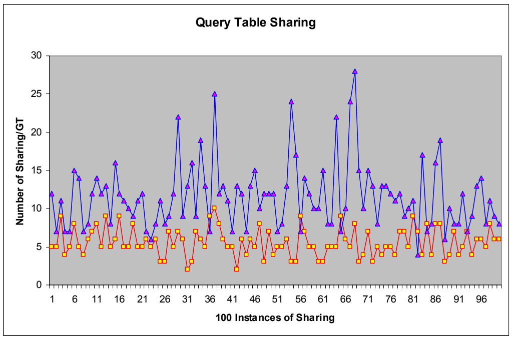


Figure 4: Query Table Sharing Variations

The maximum value indicates that during this 100 run period, one of these 10 GTs has performed 9 query table sharings with its members in order to realise the resources in the community. On the other hand, the minimum value shows that a GT has taken only 2 sharing with its members to realise the resources.

The behaviour is due to the capacity of members in the rank table of a GT. If the members of a GT have complete rank tables, which expose most of the resources in the community then the number of sharing required will be less. On the other hand, if the members have incomplete rank tables, which expose only a few resources in the community, then the number of sharing required will be high.

In the 5-member scenario, the number of sharing required has increased from an average of 5 to an average of 11. In this case the maximum number of sharing is 28 and the minimum number of sharing is 4. As the number of members is low compared to the previous situation, the chance of members exposing most of the other resources in the community is also low. In this situation, to realise all the resources, the number of sharing has to increase in order to go through the cycle of learning about the succession of members.

When the number of members is reduced to 4, we have found out that the simulation cannot be continued further. This leads us to the conclusion that, in a community of 10 GTs, a minimum of 4 members is required by each GT to fully realise the resources contained in that community. This phenomenon is similar to that of *file overfragmentation*. Our simulation program can be extended to generate this minimum threshold for a community of *n* GTs in general.

7. Discussion & Conclusions

In this article we presented the architecture of the Grid Talk (GT) agent; a pure peer-to-peer framework to facilitate resource discovery in GRID environments. In contrast to current approaches that introduce separate, inter-linked, dedicated nodes for resource discovery, our proposed framework enforces the direct interaction between requesters and providers through the sharing of supply-demand trends. Each peer learns from its own experience by constructing and managing a group of its own where all the members are ranked according to their capacity, availability and prior performance in answering specific queries. Every peer also learns from their environment through the sharing of its rank tables with all the members of its group.

Because our solution does not rely on building and maintaining some form of hierarchical or web-like middleware (as is the case with the current state-of-the-art), it exhibits the following benefits over existing approaches:

- *Better efficiency since queries are sent directly to appropriate providers.* Thus every user query generates one message rather than $n+1$ as in the case with n-level hierarchies between peers (routing indices) or brokers (JXTA search hubs). Our framework is expected to generate higher network traffic initially compared to the other systems since it relies on multicasting. The amount of this traffic is nonetheless comparable to that of standard unstructured peer-to-peer networks and more importantly our simulations have shown that this extra traffic is reduced sharply over time due to the built-up of sufficient providers per query type. In addition this process guarantees that every peer maintains a list of providers for the query types and topics its users are interested in rather than indexing a-priori the full capabilities of every peer as in the case of routing indices (*data representation efficiency*).

- *Higher adaptivity and fault-tolerance.* Any changes to the status or capacity of the providers are reported directly to the consumers through peer-to-peer table sharing (a typical peer in a group can act both as consumer and provider).

- *Better scalability.* The rank tables of every peer in our framework could grow steeply over time. However it is trivial to re-use various *cache memory control* methods to ensure that there is an upper bound to the size of these tables.

The simplicity and re-usability of our method in conjunction with the generic definition of resources make our framework applicable to areas such as Knowledge Management and Information Retrieval in specialised or generic computing environments. Our primary aim for further work is to provide a prototype implementation of our framework and make it available to the GRID community for evaluation purposes. In addition we plan to apply the same concepts to manage and share specialist knowledge such as access control policies in peer-to-peer environments.

8. References

Crespo, A. and Garcia-Molina, H., (2002), "Routing indices for peer-to-peer systems". *In 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July.

Global Grid Forum, (2005), http://www.gridforum.org/.

Orfali, R., Harkey, D. and Edwards, J., (1997), *Instant CORBA*, John Wiley & Sons, Inc., USA.

Seaborne, A., (2002) “RDQL – RDF Data Query Language” (part of the Jena RDF Toolkit), HP Labs Semantic Web activity, http://hpl.hp.com/semweb/.

Sheldon, M., Duda, M., Weiss, R. and Gifford, D, (1995), “Discover: A resource discovery system based on content routing” *In Proc. 3rd International World Wide Web Conference*, Darmstadt, Germany, April.

Tuecke, S., Czajkowski K., Foster I., Frey J., Graham J., Kesselman C. and Vanderbilt P., (2002), “Grid Service Specification” (Draft 4), http://www.globus.org/ogsa/, October.

Virupatchan, M. and Antonopoulos, N, (2002), “Towards an Automated Interactive Architecture for Thin GRIDs”. *International Conference on Communications in Computing*, Las Vegas, USA, June.

Waterhouse, S., Doolin, D., Kan, G. and Faybishenko, Y., (2002), “JXTA Search: a distributed search framework for peer-to-peer networks”, *IEEE Internet Computing*, vol 6, 2002, pp 68-73.

World Wide Web Consortium (W3C), (2002), “RDF Primer”, http://www.w3.org/TR/2002/WD-rdf-primer-20020319/, March.