Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System

Harris Papadakis
Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71110 Heraklion-Crete, Greece
{adanar}@ics.forth.gr

Paolo Trunfio, Domenico Talia
DEIS Department, University of Calabria
Via P. Bucci 41C, 87036 Rende (CS), Italy
{trunfio|talia}@deis.unical.it

Paraskevi Fragopoulou
Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71110 Heraklion-Crete, Greece
{fragopou}@ics.forth.gr

CoreGRID Technical Report
Number TR-0105
August 21, 2007

Institute on Architectural issues: scalability, dependability, adaptability

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System

Harris Papadakis
Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71110 Heraklion-Crete, Greece
{adanar}@ics.forth.gr

Paolo Trunfio, Domenico Talia
DEIS Department, University of Calabria
Via P. Bucci 41C, 87036 Rende (CS), Italy
{trunfio|talia}@deis.unical.it

Paraskevi Fragopoulou
Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O. Box 1385, 71110 Heraklion-Crete, Greece
{fragopou}@ics.forth.gr

CoreGRID TR-0105
August 21, 2007

Abstract

Peer-to-peer (P2P) computing is recognized as one of most prominent paradigms to achieve scalability in key components of Grid systems. One of these components is resource discovery, whose duty is to provide system-wide up-to-date information about resources, a task inherently limited in scalability. Unlike typical P2P systems, Grid systems manage not only static resources, but also resources whose characteristics change dynamically over time. To cope with this scenario, recent P2P-based Grid resource discovery systems employ a combination of Distributed Hash Tables (DHTs) to manage static resources, and unstructured (i.e., broadcast-like) search techniques for dynamic resources. In this paper we elaborate on this approach by designing and implementing a Grid resource discovery system that employs a dynamic querying algorithm over a structured DHT-based overlay. The system has been fully implemented and deployed on the Grid'5000 platform for testing and evaluation. The experimental performance results presented here demonstrate the efficiency of the implemented system, both in terms of number of messages and time needed to complete the search.

Keywords: Grid, resource discovery, peer-to-peer, dynamic querying, distributed hash tables.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Introduction

To achieve their envisioned global-scale deployment, Grid systems need to be scalable. Peer-to-peer (P2P) techniques are widely viewed as one of the prominent ways to reach the desired scalability. Resource discovery is one of the most important functionalities of a Grid system and, at the same time, one of the most difficult to scale. Indeed, the duty of a resource discovery system (such as the Globus MDS [1]) is to provide system-wide up-to-date information, a task which has inherently limited scalability. To add to the challenge, Grid resource discovery systems need to manage not only static resources, but also resources whose characteristics change dynamically over time, making the design critical.

Recent work [2] proposed a framework that combines the use of Distributed Hash Tables (DHTs) to search static Grid resources, and an unstructured P2P search algorithm to locate dynamic resources. Differently from standard unstructured protocols, such framework searches for dynamic resources by using a "dynamic querying" algorithm, which exploits a DHT structure to distribute queries across nodes without generating redundant messages.

In this paper we elaborate on such an approach by designing and implementing a hybrid P2P-based Grid resource discovery system which supports both static and dynamic information retrieval, and push and pull models. Like recent unstructured systems, our system is based on a two-tier approach, with peers divided in two categories (*Superpeers* and *Peers*) based on the level of service they can provide. Following this approach, each Superpeer acts as a server for a number of regular Peers, while Superpeers connect to each other in a P2P fashion at a higher level [3]. Unlike unstructured systems, we organize Superpeers using a DHT-based system, namely Chord [4] and its implementation Open Chord [5].

The search for static information is performed in a structured-like fashion, while dynamic information search is performed in an efficient unstructured-like fashion tailored to the DHT structure. Thus, the proposed system is hybrid in more than one aspect. The two-tier approach couples the completely decentralized P2P paradigm with a limited degree of centrality to reduce the effort of providing a global view of the system resources. In addition, our system couples the structured topology with a brodcast-like mechanism reminiscent of unstructured systems to locate dynamic information. The support for both a push and a pull approach to resource discovery allows for a trade-off between message cost for resource discovery and staleness of provided information.

The system has been fully implemented and deployed on the Grid'5000 platform [6] for testing and evaluation. The experimental performance results presented in this paper demonstrate the efficiency of the implemented system, both in terms of number of messages and time needed to complete the search.

The remainder of the paper is organized as follows. Section 2 discusses related work on P2P-based resource discovery. Section 3 presents the system design, and describes the algorithm of dynamic querying implemented by the system. Section 4 discusses the implementation of the system and its evaluation on the Grid'5000 platform. Finally, Section 5 conclude the paper.

2 Related work

P2P-based resource discovery systems allow nodes participating in the system to share both the storage load and the query load. In addition, they provide a robust communication overlay. P2P-based Grid resource discovery mechanisms that appear in the literature can be divided into two categories: structured and unstructured [7]. Most proposed systems depend on a structured P2P underlying layer. A structured system however assumes that all pieces of information are stored in an orderly fashion according to their values in a DHT. This is the reason structured systems support efficient resource discovery. However, apart from static resources, Grids include dynamic resources whose values change over time. Whenever the value of a resource attribute stored in a structured system changes, it needs to be republished. If this occurs too often, the cost of republishing becomes prohibitively high.

Flooding is supported by those P2P-based Grid resource discovery systems that follow the unstructured approach. Flooding, however, can generate a large volume of traffic if not carefully deployed, due to the duplicate messages generated during this process. Several P2P-based Grid resource discovery algorithms appear in the literature, trying to alleviate the excessive volume of traffic produced during flooding [8]. One of the first alternatives proposed was random walks. Each node forwards each query it receives to a single neighboring node chosen at random, a method that generates very little traffic but suffers from reduced network coverage and long response time. As an alternative, multiple random walks have been proposed, where the querying node starts simultaneously k parallel random walkers. Although compared to a single random walk this method has better behavior, it still suffers from low network coverage and long response time compared to flooding.

Hybrid methods that combine flooding with random walks have been proposed in [9]. Schemes like Directed Breadth First Search (DBFS) forward queries only to those peers that have provided results to past requests, under the assumption that they will continue to do so. Interest-based schemes aim to cluster together peers with similar content, under the assumption that those peers are better suited to serve each other needs. In another family of algorithms, query messages are forwarded selectively to part of a node neighbors based on predefined criteria or statistical information. For example, each node selects the first k highest capacity nodes or the k connections with the smallest latency to forward new queries [11]. A somewhat different approach named forwarding indices builds a structure that resembles a routing table at each node [10]. This structure stores the number of responses returned through each neighbor on each one of a preselected list of topics. Other techniques include query caching, or the incorporation of semantic information in the network [13, 12].

An approach that has been used to make resource location in unstructured P2P systems more efficient is the partitioning of the overlay network into subnetworks using content characterization methods. A different subnetwork is formed per content category. Each subnetwork connects all peers that possess files belonging to the corresponding category. A system that exploits this approach is the Semantic Overlay Networks (SONs) [13]. SONs use a semantic categorization of music files based on the music genre they belong to. The main drawback of this method is the semantic categorization of the content. An approach that overcomes this semantic categorization method has been proposed in [12].

A controlled flooding mechanism, known as "dynamic querying", has been proposed to reduce the number of messages generated by unstructured P2P systems [14]. In this paper a dynamic querying-like approach is implemented and evaluated. The method exploits the benefits of dynamic querying over the overlay network of a DHT-based P2P system to eliminate duplicate messages, thus reducing significantly the total number of messages generated by every search. The dynamic querying algorithm implemented in our framework is described in the next section.

3 System design

The proposed system is based on a two-tier architecture in which nodes belong either to the category of *Peers* or *Superpeers*, based on the level of service they can offer. Most participants act as normal Peers, while the high-bandwidth participants act as Superpeers. Superpeers participate normally in the P2P overlay and also act on behalf of Peers, which participate in the system indirectly by connecting to Superpeers. There are several reasons for this "stretching" of the P2P system definition, which dictates that all participants have equal roles and responsibilities. The first one is to improve the scalability of the system by exploiting the heterogeneity of participating nodes. This essentially means that more work can be assigned to those participants that can handle it, while at the same time removing most of the workload from the less capable peers. In addition, Peers can provide their corresponding Superpeer with static information about the resources they manage. Thus, when a Superpeer receives a query, it can forward the query only to those Peers whose resources match the static criteria. The Peers will then reply with any local resource information that also matches the dynamic part of the query.

While such two-tier approach is widely implemented by unstructured P2P systems, in our framework we organize Superpeers using Chord [4], a well known DHT-based system. There are two main reasons for this. The first one is that the Chord structure can be used to quickly resolve queries based on static information. The second reason is that the structure of Chord allows to distribute a query to all nodes in the overlay avoiding redundant (i.e., duplicate) messages. In particular, we implemented an algorithm for "dynamic querying" over a DHT, described in Section 3.1. Such algorithm allows to distribute the query to as many nodes as it is required to locate the desired information, instead of flooding the entire network for every query. This means that the cost of the lookup is further reduced, depending on the amount of matching resources that exist in the system and the number of the results required by the user.

The architecture of the system is schematically represented in Figure 1. Four types of components are defined:

- *Superpeers*: The main components of the system. They are organized in a Chord network and implement the dynamic querying algorithm. Each one is responsible for a number of Peers.
- *Peers*: Provide information about the resource they manage. The information is provided to the corresponding Superpeer following either a *push* or *pull* model (see below).

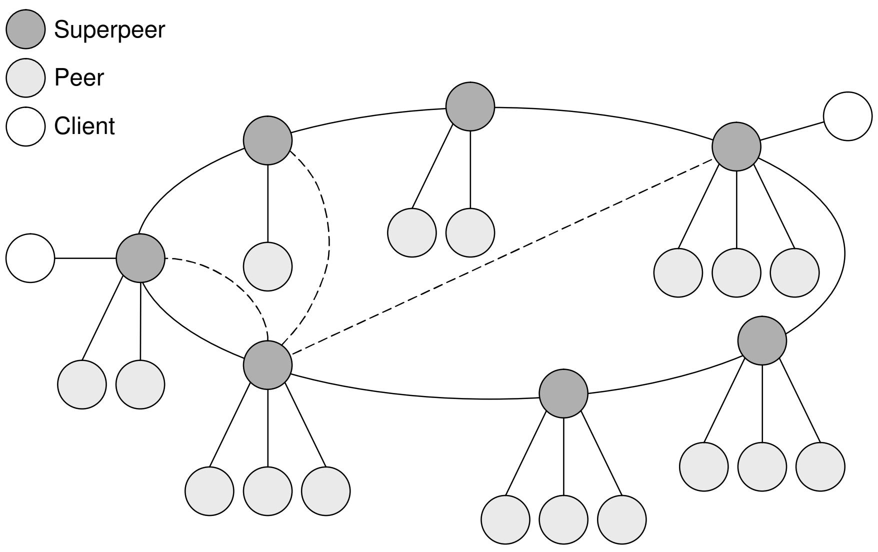

Figure 1: The architecture of the proposed system. Dashed lines indicate the fingers nodes in the Chord overlay.

- *Clients*: Connect to a Superpeer and issue queries on behalf of a user, also managing the delivery of query responses to the user.
- *Cache Servers*: The entry-points for Peers and Superpeers to the network (not shown in the figure). They hold a list of the most recently joined Superpeers and return a subset of that list to any requestor.

As mentioned above, our system provide support for both push and pull models for the dissemination of resource information (especially regarding the dynamic information). This approach is similar to the one used in the Monitoring and Discovery System (MDS) of Globus Toolkit 2, where the various GRIS modules register themselves (and their static information) to a GIIS module [1]. Each GRIS has the autonomy to decide whether it will periodically push the dynamic information to its corresponding GIIS (*push model*), or it will wait for the GIIS to query each of its registered GRISs for that information, when needed (*pull model*). In our system, each Peer can be configured to send to its Superpeer its static information only, or to periodically send up-to-date copies of its dynamic information.

Notice that, while Figure 1 shows only one Superpeer overlay, the system can be extended to include multiple overlays, one for each type of static resource information, as proposed in [15]. This will greatly reduce the number of Superpeers contacted during a dynamic query lookup.

In the remainder of this section we briefly describe the algorithm of dynamic querying over a DHT as it has been implemented in our system.

3.1 Querying dynamic resources

Dynamic querying [14] is a technique used to reduce the traffic generated in unstructured P2P networks. In dynamic querying, the peer that initiates a search controls the query propagation by sending it only to a subset of its neighbors and with a small Time-to-Live (TTL). If such first attempt does not produce a sufficient number of results, the originating peer will send the query again to a different set of neighbors with increased TTL. This process is repeated until the expected number of results is received, or until all the neighbors have been queried.

The algorithm of dynamic querying over a DHT, as outlined in [2], uses a combination of the dynamic querying approach with an algorithm for efficient broadcast over DHTs proposed in [16], which allows to perform a broadcast operation with minimal cost in a Chord-based P2P network. In a network of N nodes, a broadcast message originating at an arbitrary node reaches all other nodes after exactly $N-1$ messages, with $logN$ steps. In order to explain how dynamic querying over a DHT works, we first recall the algorithm of broadcast over DHTs.

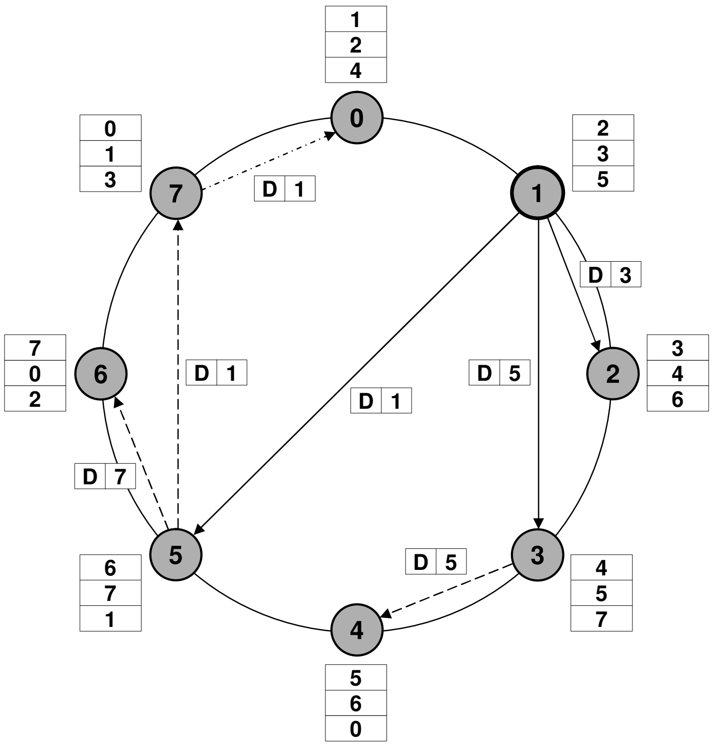

Figure 2: An example of broadcast over a Chord DHT.

Broadcast over a DHT.

Let us consider a fully populated Chord ring with $N = 2^m$ nodes and an m-bit identifier space. Each Chord node x has a finger table, with fingers pointing to nodes $x + 2^{i-1}$, where $i = 1...m$. Each of these m nodes, in turn, has its fingers pointing to other m nodes in the same way. The broadcast initiator node sends the query to all nodes in its finger table, and in turn, these nodes do the same with nodes in their finger tables. In this way, all nodes are reached in m steps. Since the same node may be pointed to by multiple fingers, the following strategy is used to avoid redundant messages. Each message sent by a node contains a "limit" argument, which is used to restrict the forwarding space of the receiving node. The "limit" argument of a message sent to the node pointed to by finger i is finger $i + 1$. Figure 2 gives an example of such Chord ring with $m = 3$ (eight nodes, three-bit identifier space and finger tables with three entries). In this example, *Node 1* initiates the broadcast of a data item D. The "limit" is sent together with the data item. Three steps of communication between nodes are shown with solid, dashed, and dashed-dotted lines. *Node 1* reaches all other nodes via $N - 1 = 7$ messages within $m = 3$ steps. The same procedure applies to Chord rings with $N < 2^m$ (i.e., not fully populated rings). In this case, the number of distinct fingers of each node is likely to be $logN$, on the average.

Dynamic querying over a DHT.

In brief, the algorithm of dynamic querying over a DHT works as follows. The initiator node (that is, the Superpeer that submits the query to the network on behalf of a Client) starts by processing the query locally, and by forwarding the query to its first n unique fingers. These fingers will in turn forward the query to all nodes in the portions of the network they are responsible for, following the broadcast algorithm described above. When a Superpeer node receives a query, it checks for local resources matching the query criteria and, in case of match, it sends a query hit directly to the initiator node, which will in turn forward it to the Client. After sending the query to its first n unique fingers, the algorithm proceeds iteratively as follows.

First, the initiator waits for a given amount of time, which is the estimated time needed by the query to reach the farthest node under the n^{th} unique finger, plus the time needed to receive a query hit from that node. Then, if the number of received query hits is equal or greater than the number of query hits desired by the Client, the initiator node terminates. Otherwise, it continues the search by sending the query to other k unique fingers after the first n ones. The value of k is chosen by calculating the number of nodes that must be contacted to obtain the desired number of query hits on the basis of the estimated popularity of the resource, which is in turn calculated as the ratio between the current number of received query hits and the estimated number of nodes reached through the first n unique fingers (which is likely to be 2^n [4]).

The iterative procedure above is repeated until the desired number of query hits is obtained, or there are no more fingers to contact. Note that, if the resource popularity is properly estimated on the first iteration, two iterations - including the first one - are sufficient to obtain the desired number of query hits.

4 Implementation and evaluation

We implemented the system using Java. Basically, each one of the system components (Superpeer, Peer, Client, and Cache Server) has been implemented as a separate Java application that can be installed independently from the other components. TCP sockets have been used to let the system components communicate with each other.

For building the Superpeer overlay we used Open Chord, an implementation of the Chord algorithm by the University of Bamberg [5]. Open Chord provides an API that enables the use of a Chord DHT within Java applications. However, that API provided only methods for joining/leaving a Chord network and inserting/removing keys from it. In order to perform dynamic querying over the overlay, we extended the Open Chord API by adding the functionality to send arbitrary messages between nodes in the system. Such send operation is asynchronous, since the controlled broadcast performed by dynamic querying is executed in a Breadth-First-Search rather than a Depth-First-Search fashion. In addition, we added a functionality which allows a developer to access the (sorted) finger table of Chord, as needed by the dynamic querying algorithm.

In the implementation of the dynamic querying algorithm, the number of unique fingers contacted during the first iteration is set to 6. As the number of nodes reachable through n distinct fingers is likely to be 2^n (see Section 3.1), the number of Superpeers contacted during the first iteration is $2^6 = 64$ on the average.

While queries are distributed using the Chord overlay, results are sent directly to the query initiator. That is, if a Peer contains resources that match the query criteria, it issues a query hit message directly to the Superpeer that initiated the dynamic querying (i.e., the Superpeer to which the Client that issued the query is connected). That Superpeer will in turn forward the query hits over its open socket with the Client, as they arrive.

4.1 Experimental results

In this section we discuss the performance of the proposed resource discovery system in a real Grid scenario. We focused on the efficiency of the system in terms of number of messages and time needed to complete a search. For our experiments we used the Grid'5000 testbed, a highly reconfigurable, controllable and easy to monitor Grid platform gathering nine sites geographically distributed in France and featuring a total of 5000 CPUs [6]. Grid'5000 is an ideal testbed for our experiments, since not only it allowed us to test the system in a real Grid platform, but it also contains sites from all of France, which more closely matches the environment of a global-scale Grid.

We used several hosts from four Grid'5000 sites (Rennes, Sophia, Nancy, and Orsay) for a total of 410 nodes across six clusters of those sites. Each host was used to execute a number of independent Peer and Superpeer applications. In order to distribute the load across sites, Peers and Superpeers have been uniformly distributed across nodes. Thus, given the number of available nodes, N, and chosen the overall number of Superpeers, S, and the average number of Peers connected to each Superpeer, P, we executed an average number of S/N Superpeers and P Peers on each node.

Each Peer and Superpeer entered the system at random times. It was, thus, quite unlikely for a Superpeer to have in its finger table Superpeers running on the same host, or for a Peer to connect to a Superpeer in the same machine. We then initiated several Clients, each of which submitted the same batch of queries, each one having search criteria with a different probability to match resources in the network. Given a query, we define the probability of match, p, as the ratio between the total number of resources that match the query criteria, and the total number of Peers in the network. When submitting the query, each Client specifies the desired number of query hits R, i.e., the number of resources to be located that match the query criteria.

We measured two performance parameters:

- The number of Superpeers contacted by each query.

Table 1: Number of Superpeers contacted to perform queries with different probabilities of match (p) and different numbers of required results (R). On the left part of the table, values measured in a network with S=5000 and P=10. On the right part, values for S=5000 and P=50.

	S=5000 P=10				S=5000 P=50			
	R=1	R=10	R=100	R=1000	R=1	R=10	R=100	R=1000
p=0.001	72	1067	5000	5000	70	310	2238	5000
p=0.01	72	147	1342	5000	77	68	283	2217
p=0.1	66	73	138	1191	68	65	69	293
p=1.0	69	71	71	129	63	63	63	63

- The time required by each query hit to reach the Client.

Notice that the first parameter corresponds to the number of messages generated by the algorithm of dynamic querying over the DHT. For the second parameter, a significant value is the time required by the R^{th} query hit to reach the Client (where R is the desired number of results), since it represents the amount of time needed to reach the goal of the search.

The experiments have been performed in two network scenarios: 1) $S = 5000$ and $P = 10$; 2) $S = 5000$ and $P = 50$. For each network scenario we submitted queries with four different probabilities of match ($p = 0.001$, 0.01, 0.1, and 1.0) and four numbers of desired query hits ($R = 1$, 10, 100, and 1000), and for all these queries we measured both the number of Superpeers contacted and the time required by each query hits to reach the Client.

Table 1 reports the number of Superpeers contacted during the various experiments. As shown in the table, the search of one resource ($R = 1$) generates an average of 68.4 messages, which is close to the average number of Superpeers reachable through the first 6 fingers. In most experiments, in fact, at least one Peer responded with a query hit during the first iteration of the search.

For higher values of R, the number of Superpeers contacted depends significantly on the value of p. This is because for small values of p the first iteration does not produce enough results, and so at least another iteration must be performed. Let us consider, for example, the case of $R = 10$ in the network with $P = 10$. When $p = 0.01$, the value of 147 indicates that an average of 7 fingers have been contacted (6 of which during the first iteration), while for $p = 0.001$ the value of 1067 corresponds to 10 fingers. Similar behavior applies to the network with $P = 50$ and for other values of R.

As shown in the table, in some cases all 5000 Superpeers are contacted during the search. This happens when the algorithm, on the basis of the estimated popularity of the resource, calculates that all remaining fingers must be contacted to obtain the desired number of results. This, obviously, will or will not lead to success based on the actual presence of enough matching resources in the network. In every case, the number of messages is bound to the number of Superpeers, without redundancy.

Figure 3 shows the time needed to receive the query hits in some of the experiments described above. For space limitations, we show only the results of searches with $R = 100$ and $R = 1000$ for various values of p. For some values of p, the search did not succeed to find the desired number of resources, because they were not available. For example, with $p = 0.001$, it is impossible to find 1000 resources when $S = 5000$ and $P = 10$. That's why Figure 3a reports only results for $p = 0.01$, 0.1, and 1.0. The same applies to Figure 3c and Figure 3d.

For all values of R, in both network scenarios, the time needed to receive the R^{th} query hit was in the order of few hundreds milliseconds. As expected, comparing the results for a given value of P and R (for example Figure 3b) we note that times increase as p decreases. Moreover, lines for higher values of p are more flat, indicating that all query hits arrive in a small time interval. This happens because for high values of p the search is completed in only one iteration, and so there is not the additional delay of a new round of search. When a second iteration of search is performed (as for $p = 0.001$ in Figure 3b), the line shows two trends: the first part with few results arriving at different times from the first 6 fingers, the second part, more flat, with several results arriving in a more close time sequence.

Comparing the arrival times of query hits for the same values of R but different values of P (for example, Figure 3a vs Figure 3b), we find that a higher value of P leads to reduced times for any value of p. This is due to the higher number of Peers - and thus matching resources - that are reached during the first iteration of dynamic querying when P is higher. Such result confirms that the two tier-approach leads to reduced search time when there is a good proportion between the number of Superpeers and Peers in the system.

As a final remark, all the experimental results presented above demonstrate the efficiency of the implemented

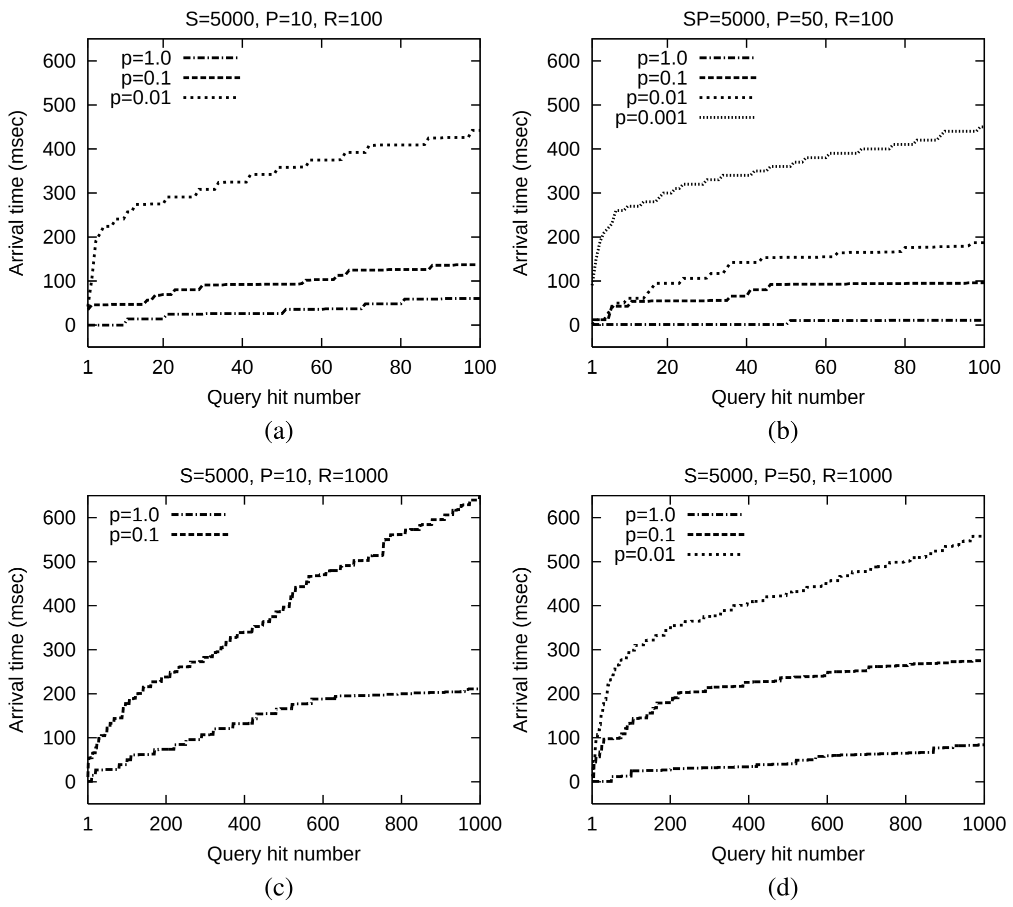

Figure 3: Time to receive query hits under different probabilities of match, in four scenarios: (a) P=10, R=100; (b) P=50, R=100; (c) P=10, R=1000; (d) P=50, R=1000. In all cases S=5000. The values on the x-axis represent the arrival numbers of the received query hits. The values on the y-axis are the times needed to receive the various query hits after the query submission.

system. The dynamic querying algorithm was able to limit and control the number of messages generated by the search in function of the probability of match and the number of desired results, while its coupling with the two-tier architecture allowed to ensure very low search times in all the experimental scenarios.

5 Conclusions

We designed and implemented a Grid resource discovery system that combines the flexibility of unstructured P2P systems and protocols, such as the two-tier architecture and the dynamic querying approach, with the efficiency of structured DHT-based systems like Chord.

The performance of the implemented system has been evaluated in a real Grid environment using the Grid'5000 testbed. The experimental results presented in the paper demonstrated the efficiency of the algorithm of dynamic querying over a DHT to control the number of messages generated by the resource discovery tasks, and its coupling with the two-tier architecture ensured very low search delays in all experimental scenarios.

Acknowledgement

We would like to thank the Grid'5000 team for allowing us to use their platform for experimenting our system. This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission

(Contract IST-2002-004265).

References

[1] Globus MDS. http://www.globus.org/toolkit/mds.

[2] D. Talia, P. Trunfio, J. Zeng. Peer-to-Peer Models for Resource Discovery in Large-scale Grids: A Scalable Architecture. Int. Conf. on High Performance Computing in Computational Sciences (Vecpar 2006), Rio de Janeiro, Brazil, 2006.

[3] B. Yang, H. Garcia-Molina. Designing a Super-peer Network. Int. Conf. on Data Engineering (ICDE 2003), 2003.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. ACM SIGCOMM'01, San Diego, USA, 2001.

[5] Open Chord. http://open-chord.sourceforge.net.

[6] Grid'5000. http://www.grid5000.fr.

[7] P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, S. Haridi. Peer-to-Peer resource discovery in Grids: Models and systems. Future Generation Computer Systems, vol. 23, n. 7, 2007.

[8] Q. Lv, P. Cao, E. Cohen, K. Li, S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. Int. Conf. on Supercomputing (SC 2002), Baltimore, USA, 2002.

[9] C. Gkantsidis, M. Mihail, A. Saberi. Hybrid Search Schemes for Unstructured Peer-to-Peer Networks. IEEE INFOCOM 2005, Miami, USA, 2005.

[10] A. Crespo, H. Garcia-Molina. Routing Indices for Peer-to-peer Systems. Int. Conf. on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 2002.

[11] D. Tsoumakos, N. Roussopoulos. A Comparison of Peer-to-Peer Search Methods. Int. Workshop on the Web and Databases (WebDB 2003), San Diego, USA, 2003.

[12] K. Sripanidkulchai, B. Maggs, H. Zhang. Efficient Content Location using Interest-based Locality in Peer-to-Peer Systems. IEEE INFOCOM 2003, San Franciso, USA, 2003.

[13] A. Crespo, H. Garcia Molina. Semantic Overlay Networks for P2P Systems. Int. Conf. on Agents and Peer-to-Peer Computing (AP2PC 2004), New York, USA, 2004.

[14] A. A. Fisk. Gnutella Dynamic Query Protocol v0.1.
http://www.the-gdf.org/ wiki/index.php?title=Dynamic_Querying.

[15] H. Papadakis, P. Fragopoulou, E. P. Markatos, M. Dikaiakos, A. Lambrinidis. Divide et Impera: Partitioning Unstructured Peer-to-Peer Systems to Improve Resource Location. 2nd CoreGRID Integration Workshop, Krakow, Poland, 2006.

[16] S. El-Ansary, L. Alima, P. Brand, S. Haridi. Efficient Broadcast in Structured P2P Networks. Int. Symp. on Cluster Computing and the Grid (CCGRID'05), Cardiff, UK, 2005.