An Experimental Evaluation of the DQ-DHT Algorithm in a Grid Information Service

Harris Papadakis, Paolo Trunfio, Domenico Talia and Paraskevi Fragopoulou

Abstract DQ-DHT is a resource discovery algorithm that combines the Dynamic Querying (DQ) technique used in unstructured peer-to-peer networks with an algorithm for efficient broadcast over a Distributed Hash Table (DHT). Similarly to DQ, DQ-DHT dynamically controls the query propagation on the basis of the desired number of results and the popularity of the resource to be located. Differently from DQ, DQ-DHT exploits the structural properties of a DHT to avoid message duplications, thus reducing the amount of network traffic generated by each query. The goal of this paper is to evaluate experimentally the amount of traffic generated by DQ-DHT compared to the DQ algorithm in a Grid infrastructure. A prototype of a Grid information service, which can use both DQ and DQ-DHT as resource discovery algorithm, has been implemented and deployed on the Grid'5000 infrastructure for evaluation. The experimental results presented in this paper show that DQ-DHT significantly reduces the amount of network traffic generated during the discovery process compared to the original DQ algorithm.

Harris Papadakis
Foundation for Research and Technology-Hellas, Institute of Computer Science (FORTH-ICS), Heraklion, Greece, e-mail: adanar@ics.forth.gr

Paolo Trunfio
Department of Electronics, Computer Science and Systems (DEIS), University of Calabria, Rende, Italy, e-mail: trunfio@deis.unical.it

Domenico Talia
Institute of High Performance Computing and Networking, Italian National Research Council (ICAR-CNR) and Department of Electronics, Computer Science and Systems (DEIS), University of Calabria, Rende, Italy, e-mail: talia@deis.unical.it

Paraskevi Fragopoulou
Foundation for Research and Technology-Hellas, Institute of Computer Science (FORTH-ICS) and Department of Applied Informatics and Multimedia, Technological Educational Institute of Crete, Heraklion, Greece, e-mail: fragopou@ics.forth.gr

1 Introduction

Information services are fundamental components of Grid systems as they allow to locate the resources needed to execute large-scale distributed applications based on application requirements and resource availability. Designing a decentralized but efficient Grid information service is a significant strand of research, with many researches demonstrating the use of peer-to-peer (P2P) models and techniques as an effective alternative to centralized and hierarchical solutions [1]. Such P2P systems are typically classified as *structured* or *unstructured*, based on the way nodes are linked each other and information about resources is placed in the resulting overlay.

Structured systems, like Chord [2], use a Distributed Hash Table (DHT) to assign to each node the responsibility for a specific part of the resources. When a peer wishes to find a resource identified by a given key, the DHT allows to locate the node responsible for that key in $O(\log N)$ hops using only $O(\log N)$ neighbors per node. In unstructured systems, like Gnutella [3], links among nodes can be established arbitrarily and data placement is unrelated from the topology of the resulting overlay. To locate a given resource, the query must be distributed through the network to reach as many nodes as needed, a method known as "flooding." Each node reached by the query processes it on the local resources and, in case of match, replies to the query initiator.

Thanks to the DHT infrastructure, searching in structured systems is more scalable - in terms of traffic generated - than searching in unstructured systems. However, DHT-based lookups do not support arbitrary types of queries (e.g., regular expressions [4]) since it is infeasible to generate and store keys for every query expression. Unstructured systems, on the contrary, can do it effortless since all queries are processed locally on a node-by-node basis [5]. Even though the lookup mechanisms of DHT-based systems do not support arbitrary queries, it is possible to exploit their structure to propagate any kind of information (including arbitrary queries) to all nodes in its overlay. Such queries can then be processed on a node-by-node basis as in unstructured systems. In this way, the DHT can be used for both key-based lookups and arbitrary queries, combining the efficiency of structured networks with the flexibility of unstructured search.

This strategy has been exploited in the design of DQ-DHT [6], a P2P search algorithm that supports arbitrary queries in a Chord DHT by implementing an efficient (i.e., bandwidth-saving) unstructured search technique on top of its overlay. In particular, DQ-DHT combines the Dynamic Querying (DQ) technique [7], which is used to reduce the amount of traffic generated by the search process in unstructured P2P networks, with an algorithm that allows to perform efficient broadcast of arbitrary data over a Chord DHT [8].

In the original DQ algorithm, which is used in unstructured systems like Gnutella, the querying node starts the search by sending the query to a few of its neighbors and with a small Time-to-Live (TTL). The main goal of this first phase (referred to as "probe query") is to estimate the popularity of the resource to be located. If such an attempt does not produce a sufficient number of results, the querying node sends the query towards the next neighbor with a new TTL. Such

TTL is calculated taking into account both the desired number of results, and the resource popularity estimated during the previous phase. This process is repeated until the expected number of results is received or all the neighbors have already been queried.

Similarly to DQ, DQ-DHT dynamically controls the query propagation on the basis of the desired number of results and the popularity of the resource to be located. Differently from DQ, DQ-DHT exploits the structural properties of a DHT to avoid message duplications, thus reducing the amount of network traffic generated by each query. A detailed description of the DQ-DHT algorithm, as well as an evaluation of its performance obtained through simulations, has been presented in a previous work [6].

The goal of this paper is to experimentally evaluate the amount of traffic generated by DQ-DHT compared to the original DQ algorithm in a real Grid infrastructure. To this end, a prototype of a Grid information service, which can use both DQ and DQ-DHT as resource discovery algorithm, has been implemented and deployed on the Grid'5000 infrastructure [9] for evaluation. The experimental results presented in this paper show that DQ-DHT significantly reduces the amount of network traffic generated during the discovery process compared to the original DQ algorithm. These results confirm that combining unstructured search techniques with structured overlays is a simple but effective way to support both DHT-based lookups and arbitrary queries using a single overlay.

The rest of the paper is organized as follows. Sect. 2 provides a background on the DQ-DHT algorithm. Sect. 3 describes the prototype of Grid information service implemented to perform the evaluation and presents the experimental results. Sect. 4 discusses related work. Finally, Sect. 5 concludes the paper.

2 Background on DQ-DHT

As mentioned above, DQ-DHT combines the DQ technique with an algorithm that allows to perform a broadcast operation with minimal cost in a Chord network. In order to better explain how DQ-DHT works, Sect. 2.1 introduces the algorithm of broadcast over a Chord DHT, proposed in [8]. Then, Sect. 2.2 briefly describes the DQ-DHT algorithm.

2.1 Broadcast over a Chord DHT

Chord assigns to each node an m-bit identifier that represents its position in a circular identifier space ranging from 0 and $2^m - 1$. Each node x maintains a *finger table* with m entries. The j^{th} entry in the finger table at node x contains the identity of the first node, s, that succeeds x by at least 2^{j-1} positions on the identifier circle, where $1 \leq j \leq m$. Node s is called the j^{th} *finger* of node x. If the identifier space is not

fully populated (i.e., the number of nodes, N, is lower than 2^m), the finger table contains redundant fingers. In a network of N nodes, the number u of unique (i.e., distinct) fingers of a generic node x is likely to be $\log_2 N$ [2]. In the following, we will use the notation F_i to indicate the i^{th} *unique finger* of node x, where $1 \leq i \leq u$.

To broadcast data item D, a node x sends a broadcast message to all its unique fingers. This message contains D and a *limit* argument, which is used to restrict the forwarding space of a receiving node. The *limit* sent to F_i is set to F_{i+1}, for $1 \leq i \leq u-1$. The *limit* sent to the last unique finger, F_u, is set to the identifier of the sender, x. When a node y receives a broadcast message with a data item D and a given *limit*, it is responsible for forwarding D to all its unique fingers in the interval $]y, limit[$. When forwarding the message to F_i, for $1 \leq i \leq u-1$, y supplies it a new *limit*, which is set to F_{i+1} if it does not exceed the old *limit*, or the old *limit* otherwise. As before, the new *limit* sent to F_u is set to y. As shown in [8], in a network of N nodes, a broadcast message originating at an arbitrary node reaches all other nodes in $O(\log_2 N)$ steps and with exactly $N-1$ messages.

Fig. 1a shows an example of broadcast in a fully populated Chord ring, where $u = m = 4$. For each node, the corresponding finger table is shown. The broadcast messages are represented by rectangles containing the data item D and the *limit* parameter. The entire broadcast is completed in $u = 4$ steps, represented with solid, dashed, dashed-dotted, and dotted lines, respectively. In this example, the broadcast is initiated by node 2, which broadcasts a message to all nodes in its finger table (nodes 3, 4, 6 and 10) (*step 1*). Nodes 3, 4, 6 and 10 in turn forward the broadcast message to their fingers under the received *limit* value (*step 2*). The same procedure applies iteratively, until all nodes in the network are reached (*steps 3* and *4*).

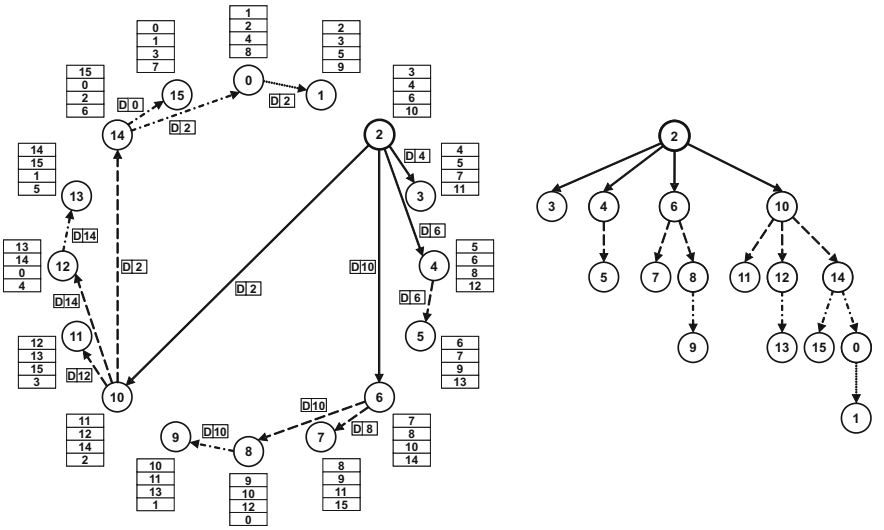


Fig. 1 (a) Example of broadcast in a fully populated Chord ring; (b) corresponding spanning tree

The overall broadcast procedure can be viewed as the process of passing the data item through a spanning tree that covers all nodes in the network [8]. Fig. 1b shows the spanning tree corresponding to the example of broadcast shown in Fig. 1a. As discussed in [6], the spanning tree associated to the broadcast over a (fully populated) Chord ring is a binomial tree.

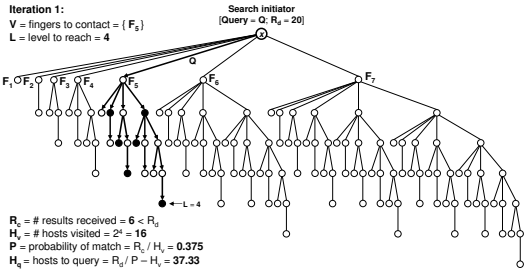
2.2 The DQ-DHT algorithm

As mentioned earlier, the goal of DQ-DHT is two-fold: allowing arbitrary queries in a Chord DHT and supporting dynamic adaptation of the search based on the popularity of the resources to be located. To support dynamic search adaptation, DQ-DHT performs the search in an iterative way, similarly to the original DQ algorithm introduced in Sect. 1. In the following, the DQ-DHT algorithm is briefly described.

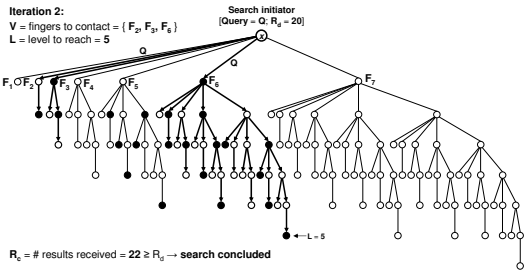
Let x be the node that initiates the search, U the set of unique fingers of x not yet visited, and R_d the desired number of results. Initially U includes all unique fingers of x. Like in DQ, the search starts with a probe query, aimed at evaluating the popularity of the resource to be located. To this end, node x selects a subset V of U and sends the query to all fingers in V. These fingers will in turn forward the query to all nodes in the portions of the spanning tree they are responsible for, following the DHT broadcast algorithm described in Sect. 2.1. When a node receives a query, it checks for local resources matching the query criteria and, for each matching resource, sends a query hit directly to x. The fingers in V are removed from U to indicate that they have been already visited.

After sending the query to all nodes in V, x waits for an amount of time T_L, which is the estimated time needed by the query to reach all nodes up to a given level L of the subtrees rooted at the unique fingers in V, plus the time needed to receive a query hit from those nodes. The value of L can be chosen to be equal to the depth of the deepest subtree associated to the fingers in V. The time needed to pass a message from level to level can be estimated taking into account the network latency measured by the application. After the waiting period, if the current number of received query hits R_c is equal or greater than R_d, x concludes the search. Otherwise, the algorithm proceeds iteratively as follows.

At each iteration, node x: 1) calculates the resource popularity P as the ratio between R_c and the number of nodes already theoretically queried; 2) calculates the number H_q of hosts in the network that should be queried to hit R_d query hits based on P; 3) chooses, among the nodes in U, a new subset V of unique fingers whose associated subtrees cumulatively contain the minimum number of nodes that is greater or equal to H_q; 4) sends the query to all nodes in V; 5) waits for an amount of time needed to propagate the query to all nodes in the subtrees associated to V. The iterative procedure above is repeated until the desired number of query hits is reached, or there are no more fingers to contact. If the item popularity is properly



(a)



(b)

Fig. 2 Example of a two-iteration DQ-DHT search in a fully populated Chord network with 128 nodes: (a) first iteration; (b) second iteration. Filled black circles represent nodes that produced a query hit after receiving the query. Empty bold circles represent nodes that received the query but did not produce a query hit. Empty circles are nodes that did not receive the query

estimated after the probe query, only one additional iteration may be sufficient to obtain the desired number of results.

Fig. 2 illustrates an example of a two-iteration DQ-DHT search in a fully populated Chord network with 128 nodes. The root of the binomial tree represents the search initiator, x, and its children represent the fingers $F_1 ... F_7$ of x. As discussed in [6], given the structural properties of binomial trees, the subtree rooted at finger F_i has depth $i-1$ and number of nodes equal to 2^{i-1}. In this example the query to process is indicated as Q and the desired number of results is $R_d = 20$.

Fig. 2a shows the first iteration of search (probe query). Node x chooses to send the query to finger F_5 (i.e., $V = \{F_5\}$) in order to probe $2^{5-1} = 16$ nodes at first. Note that in a real scenario (like that described in Sect. 3) the number of nodes to be contacted during the probe query should be larger to obtain a good estimation of the resource popularity, as discussed in [6]. After sending Q to F_5, x waits for an amount of time proportional to $L = 4$ (which is the depth of the subtree rooted at F_5) before counting the number of results received. The filled black circles represent the nodes, in the subtree rooted at F_5, which have sent a query hit to x in response to Q. Thus, after the first iteration, the number of results is $R_c = 6$. Since $R_c < R_d$, x proceeds by calculating the resource popularity P and the number H_q of hosts to query to obtain a minimum of R_d results, as shown in the figure.

The second iteration is shown in Fig. 2b. Given $H_q = 37.33$, node x chooses the new set of fingers to contact as $V = \{F_2, F_3, F_6\}$ since the total number of nodes in the subtrees associated to the fingers in V (which is equal to $\sum_{i \in \{2,3,6\}} 2^{i-1} = 38$) is the minimum value, greater or equal to H_q, that can be obtained from the fingers not previously contacted. Then, Q is sent to all fingers in V and, after a waiting period proportional to $L = 5$ (depth of the subtree associated to F_6), the search is concluded because other 16 nodes (filled black circles in the figure) have sent a query hit to x, reaching the goal of obtaining at least R_d results.

3 Experimental evaluation

In this section we evaluate the amount of traffic generated by DQ-DHT compared to the original DQ algorithm in a Grid infrastructure. We focus on the traffic generated because it is the performance parameter that mostly affects the overall efficiency of a large-scale P2P system. This is particularly true when a large number of concurrent searches are performed on the same overlay, leading to nodes overload and possible bandwidth saturation even in high-end Grid networks. Sect. 3.1 describes the prototype of Grid information service implemented to perform the experimental evaluation. Sect. 3.2 presents the experimental results obtained by deploying and using the prototype on the Grid'5000 platform.

3.1 System prototype

The prototype of Grid information service used to compare DQ-DHT with DQ is an extension of the prototype originally presented in [10]. Such prototype is based on a "superpeer" architecture in which nodes belong either to the category of *peers* or *superpeers* based on the level of service they can offer. Most nodes act as peers, while nodes with higher capacity act as superpeers. Superpeers form the P2P overlay and also act on behalf of peers, which participate in the system indirectly by connecting to superpeers.

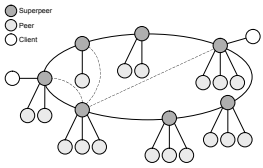


Fig. 3 The architecture of the original prototype presented in [10], with superpeers organized to form a Chord overlay

Several Grid information services proposed so far are based on superpeer architectures (see [1] for some examples) since they are naturally appropriate to the organizational nature of current Grids. Moreover, the superpeer model allows to improve the scalability of large-scale 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 nodes.

The original prototype organizes superpeers to form a Chord overlay (see Fig. 3). This overlay can be used to perform both key-based lookups using the Chord algorithm, and arbitrary queries using the DQ-DHT algorithm. As the goal of this work is to compare DQ-DHT with the original DQ algorithm (which works only on unstructured overlays) we extended the original prototype by building also an unstructured overlay among superpeers. In this way, an arbitrary query can be processed using either DQ-DHT over the Chord overlay, or DQ over the unstructured overlay.

For the construction of the Chord overlay we used Open Chord, an implementation of the Chord algorithm by the University of Bamberg [11]. The Open Chord API provides only methods for joining/leaving a Chord network and inserting/removing keys from it. To perform DQ-DHT searches over the overlay, we extended that API by adding the functionality to send arbitrary messages among nodes in the system.

The unstructured overlay among superpeers is built using a custom implementation of the Gnutella protocol [3], appropriately extended to implement the original DQ algorithm [7]. DQ works iteratively as described in Sect. 1. In particular, after each iteration, the querying node calculates the total number H_t of hosts to query to reach the desired number of results. Then, it calculates the number H_n of hosts to query per neighbor as H_t/n, where n is the number of neighbors that have not yet received the query. Finally, it calculates the minimum TTL to reach H_n hosts through the next neighbor, and sends the query towards that neighbor.

To get a fair comparison of the number of messages generated by DQ and DQ-DHT, the prototype ensures that the average number of connections (i.e., neighbors) in the unstructured overlay is equal to the average number of connections (i.e., unique fingers) in the Chord overlay. As mentioned in Sect. 2.1, this number can be calculated as $\log_2 N$, where N is the number of superpeers in the overlay.

Furthermore, for uniformity purposes, while queries are distributed using either the Chord overlay or the unstructured 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 search (i.e., the superpeer to which the client/peer that issued the query is connected).

3.2 Experimental results

The goal of our experiments is to evaluate the efficiency of DQ-DHT by comparing the amount of traffic generated by it with that generated by the original DQ algorithm. The comparison has been performed by executing the two algorithms in the same network scenario (i.e., same number of superpeers and peers) and with the same query parameters (desired number of results and probability of match). The amount of traffic has been measured as the number of messages exchanged across the superpeer overlay to complete a search process. The experiments have been performed by deploying our Grid information service prototype on the Grid'5000 platform [9].

We used hosts from four Grid'5000 sites (Rennes, Sophia, Nancy, and Orsay) for a total of around 400 hosts across those sites. Each host has been used to execute a number of independent superpeer and peer applications. To distribute the load across sites, superpeers and peers have been uniformly and randomly distributed across all hosts. In order to build the largest possible infrastructure, we first fixed $P_S = 10$ as the number of peers per superpeer, and then we performed some experiments to find the maximum number of superpeers (with associated peers) that was possible to deploy using all the available hosts, based on the memory availability of the hosts. Based on the results of these experiments, we managed to build a network including $S = 8500$ superpeers, each one connected to P_S peers, on average. Each superpeer in the unstructured overlay has been connected to $\log_2 S \simeq 13$ other superpeers, in order to obtain approximatively the same number of connections that superpeers have in the Chord overlay, as explained in Sect. 3.1.

After initializing the overlays, we started several clients, each of which submitted the same batch of queries. Each query had a different probability to match the resources available 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_d, i.e., the number of resources to be located that match the query criteria. Notice that, for the purpose of our experiments, the only relevant information associated to a query is its probability of match. Thus,

in our prototype, each query carries its value of P and each node receiving that query will reply with a query hit with probability P.

All the searches have been performed in the same network with $S = 8500$ superpeers and $P_S = 10$ peers per superpeer. Each query submitted to the system had a probability of match, P, that varied in the following values (expressed as percentages): 0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8 and 1.6. The desired number of results, R_d, was varied in the following values: 10, 20, 40 and 100.

The parameters of the original DQ algorithm are: the number of neighbors contacted during the probe phase, N_p, and the TTL used for the probe query, TTL_p. We used the following values: $N_p = 3$ and $TTL_p = 2$, as suggested in [7]. After the probe query, the maximum allowed value of TTL is increased to 5, to ensure complete network coverage.

The parameter of DQ-DHT is the set of fingers to contact during the probe query, V. We used $V = F_8$ and set consequently the value of L to 7, to reach all superpeers under finger F_8. Since the number of nodes reachable through the i^{th} finger is likely to be 2^{i-1}, the number of superpeers reached during the first iteration was $2^7 = 128$ on average.

Table 1 lists the main parameters introduced above and all the values used to carry out the experiments.

Table 1 Parameters values used to carry out the experiments

Symbol	Definition	Values
S	Number of superpeers	8500
P_S	Number of peers per superpeer	10
P	Probability of match (%)	.0125, .025, .05, .1, .2, .4, .8, 1.6
R_d	Desired number of results	10, 20, 40, 100
N_p	Neighbors contacted during probe phase (DQ)	3
TTL_p	TTL used during probe phase (DQ)	2
V	Fingers contacted during probe phase (DQ-DHT)	$\{F_8\}$

The graphs in Fig. 4 show the number of messages generated by DQ and DQ-DHT to process queries with different probabilities of match, for increasing values of R_d. Each value in the graphs is calculated as an average of 25 search executions, where at each search the same query is submitted to a randomly chosen superpeer.

As expected, for all values of R_d, the number of messages decreases when the probability of match increases, with both DQ and DQ-DHT. This is due to the fact that both DQ and DQ-DHT are able to reduce the query propagation when the popularity of the resource to be located increases. In particular, for the highest probabilities of match (i.e., 0.8-1.6%) the lines associated to DQ and DQ-DHT are always superimposing, since with both algorithm it is possible to obtain the desired number of results by contacting just a small number of superpeers.

The behaviors of DQ and DQ-DHT diverge significantly in presence of low probabilities of match. For example, with $R_d = 10$ and $P = 0.0125\%$ (see Fig. 4a), DQ-DHT generates around 8200 messages, while DQ generates almost 14000 messages.

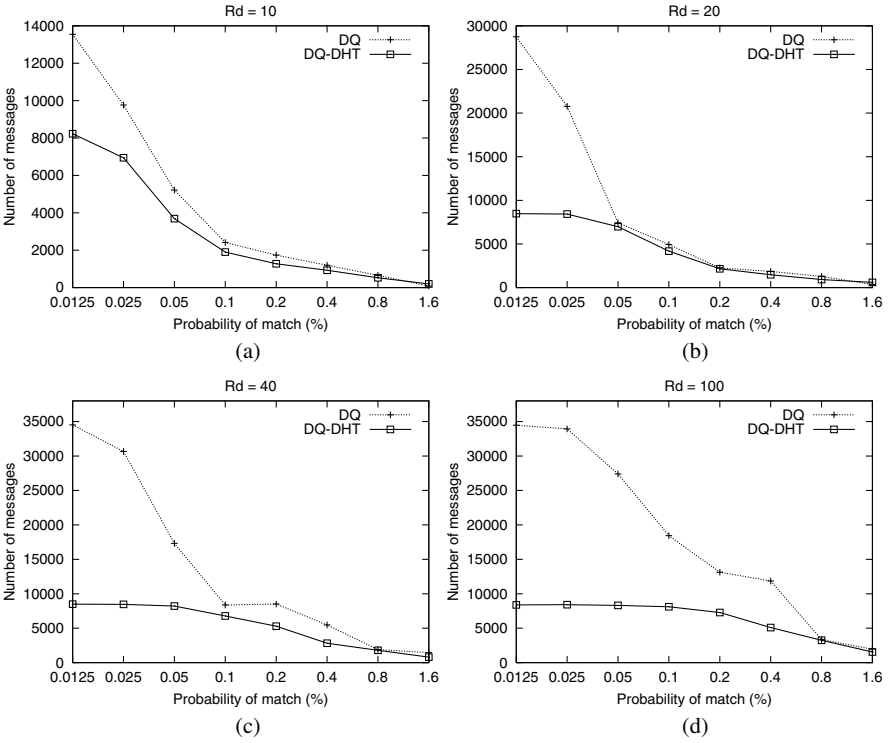


Fig. 4 Number of messages generated by DQ and DQ-DHT to process queries with different probabilities of match, with increasing values of R_d. (a) $R_d = 10$; (b) $R_d = 20$; (c) $R_d = 40$; (d) $R_d = 100$

This difference can be explained taking into account that with DQ-DHT each superpeer is contacted at most once (i.e., there is not messages duplication), while in DQ a message can reach a superpeer more than once due to the randomness of connections.

The difference between DQ and DQ-DHT in presence of low probabilities of match increases when the total number of resources matching the query criteria is lower than R_d. In our experiments this happens in three cases: 1) $R_d = 20$ and $P < 0.025\%$; 2) $R_d = 40$ and $P < 0.05\%$; 3) $R_d = 100$ and $P < 0.8\%$. In all these cases, both DQ and DQ-DHT cannot find the desired number of results even if they have distributed the query to all superpeers. Therefore, DQ-DHT generates exactly S messages (i.e., each superpeer is contacted exactly once), while DQ generates a high number of unnecessary (duplicated) messages.

As a final remark, all the experimental results presented above demonstrate the ability of DQ-DHT to control the number of messages generated by the search in function of the probability of match and the number of desired results, also com-

pared to the original DQ algorithm implemented on an unstructured overlay built on top of the same nodes.

4 Related work

The most related work to DQ-DHT is the Structella system designed by Castro et al. [12]. Structella replaces the random graph of Gnutella with the structured overlay of Pastry [13], while retaining the content placement and discovery mechanisms of unstructured P2P systems to support complex queries. Two discovery mechanisms are implemented in Structella: constrained flooding and random walks.

Constrained flooding is based on the algorithm of broadcast over Pastry presented in [14]. A node x broadcasts a message by sending the message to all the nodes y in the Pastry's routing table. Each message is tagged with the routing table row r of node y. When a node receives a message tagged with r, it forwards the message to all nodes in its routing table in rows greater than r. To constrain the flood, an upper bound is placed on the row number of entries to which the query is forwarded.

Random walks in Structella are implemented by walking along the ring formed by neighboring nodes in the identifier space. When a node receives a query in a random walker, it uses the Pastry's leaf set to forward the query to its left neighbor in the identifier space. It also evaluates the query against the local content and sends matching content back to the query originator. A random walker is terminated when it finds matching content. Multiple concurrent random walkers can be used to improve search time.

DQ-DHT and Structella share the same goal of supporting complex queries in structured network. However, DQ-DHT has been designed to find an arbitrary number of resources matching the query criteria, while Structella is designed to discover just one of such resources. In Structella in fact, with both constrained flooding and random walks, a node stops forwarding a query if it has matching content.

A few other research works broadly relate to our system for their combined use of structured and unstructured P2P techniques. Loo et al. [15] propose a hybrid system in which DHT-based techniques are used to index and search rare items, while flooding techniques are used for locating highly-replicated content. Search is first performed via conventional flooding techniques of the overlay neighbors. If not enough results are returned within a predefined time, the query is reissued as a DHT query. This allows fast searches for popular items and at the same time reduces the flooding cost for rare items.

A critical point in such system is identifying which items are rare and must be published using the DHT. Two techniques are proposed. A first heuristic classifies as rare the items that are seen in small result sets. However, this method fails to classify those items that have not have been previously queried and found. Another proposal is to base the publishing on well-known term frequencies, and/or by maintaining and possibly gossiping historical summary statistics on item replicas.

Another example is the work by Zaharia and Keshav [16], who focus on the problem of selecting the best algorithm to be used for a given query in a hybrid network allowing both unstructured search and DHT-based lookups. A gossip-based algorithm is used to collect global statistics about document availability and keyword popularity that allow peers to predict the best search technique for a given query.

Each peer starts by generating a synopsis of its own document titles and keywords and labels it as its "best" synopsis. In each round of gossip, it chooses a random neighbor and sends the neighbor its best synopsis. When a node receives a synopsis, it fuses this synopsis with its best synopsis and labels the merged synopsis as its best synopsis. This results in every peer getting the global statistics after $O(\log N)$ rounds of gossip.

Given a query composed by a set of keywords, a peer estimates the expected number of documents matching that set of keywords using the information in its best synopsis. If this number is over a given threshold, many matches are expected, so the peer floods the query. Otherwise, it uses the DHT to search for each keyword, requesting an in-network join, if that is possible. The flooding threshold is dynamically adapted by computing the utility of both flooding and DHT search for a randomly chosen set of queries.

It is worth noticing that the last two systems do not support arbitrary queries, since information about resources is published and searched using DHT-based mechanisms. DQ-DHT, on the contrary, supports arbitrary queries in an easy way since content placement is unrelated from the DHT overlay and query processing is performed on a node-by-node basis.

5 Conclusions

Providing efficient resource discovery mechanisms in Grids, Clouds and large-scale distributed systems is fundamental to build and execute distributed applications involving geographically dispersed resources. In this paper we experimentally evaluated the efficiency of DQ-DHT, a resource discovery algorithm that combines the Dynamic Querying (DQ) technique used in unstructured P2P networks with an algorithm for efficient broadcast over a Distributed Hash Table (DHT). Similarly to DQ, DQ-DHT dynamically controls the query propagation on the basis of the desired number of results and the popularity of the resource to be located. Differently from DQ, DQ-DHT exploits the structural constraints of a DHT to avoid message duplications, thus reducing the amount of network traffic generated by each query.

This paper experimentally evaluated the amount of traffic generated by DQ-DHT compared to the original DQ algorithm in a large-scale Grid infrastructure. A prototype of a Grid information service, which can use both DQ and DQ-DHT as resource discovery algorithm, has been implemented and deployed on the Grid'5000 infrastructure for evaluation. The experimental results presented in this paper showed that DQ-DHT significantly reduces the amount of network traffic generated during the resource discovery process compared to the original DQ algorithm. These results

confirm that combining unstructured search techniques with structured overlays is a simple but effective way to support both DHT-based lookups and arbitrary queries using a single overlay.

Acknowledgement

We would like to thank the Grid'5000 team for providing us the platform for deploying and experimenting our system.

References

1. Trunfio, P., Talia, D., Papadakis, H., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., Vlassov, V., Haridi, S.: Peer-to-Peer Resource Discovery in Grids: Models and Systems. Future Generation Computer Systems **23**(7), 864-878 (2007)
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. SIGCOMM'01, San Diego, USA (2001)
3. Gnutella Protocol Development. http://rfc-gnutella.sourceforge.net. Cited 24 Sep 2009
4. Castro, M., Costa, M., Rowstron, A.: Debunking Some Myths About Structured and Unstructured Overlays. 2nd Symposium on Networked Systems Design and Implementation (NSDI'05), Boston, USA (2005)
5. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P Systems Scalable. SIGCOMM'03, Karlsruhe, Germany (2003)
6. Talia, D., Trunfio, P.: Dynamic Querying in Structured Peer-to-Peer Networks. 19th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2008), Samos Island, Greece, LNCS **5273**, 28-41 (2008)
7. Fisk, A.: Gnutella Dynamic Query Protocol v0.1. http://www9.limewire.com/developer/dynamic_query.html. Cited 24 Sep 2009
8. El-Ansary, S., Alima, L., Brand, P., Haridi, S.: Efficient Broadcast in Structured P2P Networks. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, USA (2003)
9. Bolze, R., Cappello, F., Caron, E., Dayd, M., Desprez, F., Jeannot, E., Jgou, Y., Lantri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E-G., Touche, I.: Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. Int. Journal of High Performance Computing Applications **20**(4), 481-494 (2006)
10. Papadakis, H., Trunfio, P., Talia, D., Fragopoulou, P.: Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System. In: Danelutto, M., Fragopoulou, P., Getov, V. (eds.) Making Grids Work, pp. 89-101. Springer, USA (2008)
11. Open Chord. http://open-chord.sourceforge.net. Cited 24 Sep 2009
12. Castro, M., Costa, M., Rowstron, A.: Should we build Gnutella on a structured overlay? Computer Communication Review **34**(1), 131-136 (2004)
13. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Middleware 2001, Heidelberg, Germany (2001)
14. Castro, M., Jones, M. B., Kermarrec, A.-M., Rowstron, A., Theimer, M., Wang, H., Wolman, A.: An Evaluation of Scalable Application-Level Multicast Built Using Peer-to-Peer Overlays. IEEE INFOCOM'03, San Francisco, USA (2003)
15. Loo, B.T., Huebsch, R., Stoica, I., Hellerstein, J.M.: The Case for a Hybrid P2P Search Infrastructure. 3rd Int. Work. on Peer-to-Peer Systems (IPTPS'04), La Jolla, USA (2004)
16. Zaharia, M., Keshav, S.: Gossip-based Search Selection in Hybrid Peer-to-Peer Networks. 5th Int. Workshop on Peer-to-Peer Systems (IPTPS'06), Santa Barbara, USA (2006)