Dynamic Querying in Structured Peer-to-Peer Networks

Domenico Talia and Paolo Trunfio

DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{talia,trunfio}@deis.unical.it

Abstract. Dynamic Querying (DQ) is a technique adopted in unstructured Peer-to-Peer (P2P) networks to minimize the number of peers that is necessary to visit to reach the desired number of results. In this paper we introduce the use of the DQ technique in structured P2P networks. In particular, we present a P2P search algorithm, named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to perform DQ-like searches over DHT-based overlays. The aim of DQ-DHT is two-fold: allowing arbitrary queries to be performed in structured P2P networks, and providing dynamic adaptation of the search according to the popularity of the resources to be located. This paper describes the DQ-DHT algorithm using Chord as basic overlay and analyzes its performance in comparison with DQ in unstructured networks.

1 Introduction

Structured Peer-to-Peer (P2P) systems like Chord [1] keep association of resource identifiers to nodes using a Distributed Hash Table (DHT), which allows to locate the node responsible for the resource with a given Id (or key) with logarithmic performance bounds. As compared to unstructured P2P systems like Gnutella [2], however, structured systems provide a limited support to complex queries. Although several extensions to basic DHT schemes have been proposed to support, for instance, range queries [3], multi-attribute search [4], and keyword-based search [5], DHT-based lookups still do not support arbitrary queries (e.g., regular expressions [6]) since it is infeasible to generate and store keys for every query expression. On the other hand, unstructured systems can do it effortless since all queries are processed locally on a node-by-node basis [7].

Even if the lookup mechanisms of DHT-based systems do not support arbitrary queries, it is possible to exploit their structure to distribute any kind of information across the overlay with minimal cost. For example, in [8] a technique for efficient broadcast over a DHT is proposed. Using such technique, a broadcast message originating at an arbitrary node in the DHT overlay reaches all other nodes without redundant messages in $O(\log N)$ steps. It can be used to broadcast arbitrary types of queries, which can be then processed locally by single nodes as in unstructured systems. We elaborate on such an approach by proposing a P2P search algorithm, named DQ-DHT (Dynamic Querying over a Distributed Hash Table), to provide efficient execution of arbitrary queries in structured P2P networks. DQ-DHT is based on a combination of the broadcast technique mentioned above with the Dynamic Querying (DQ) technique [9] used in unstructured networks.

The goal of DQ is to minimize the number of nodes that is necessary to visit in an unstructured network to obtain the desired number of results. The query initiator 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 "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 search initiator 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 there are no more neighbors to query.

Similarly to DQ, DQ-DHT performs the broadcast in an iterative way until the target number of results is obtained. At each iteration, a new subset of nodes is queried on the basis of the estimated resource popularity and the desired number of results. Differently from DQ, DQ-DHT exploits the structural constraints of the DHT to avoid message duplications and ensure higher success rate.

DQ-DHT has been particularly designed to serve as resource discovery mechanism for decentralized infrastructures like computational Grids. Large-scale Grids are typically organized into multiple administrative domains. Within each domain, one node is designated as information server to answer queries about all the resources belonging to that domain. Since information servers are highly reliable nodes, it is possible to build a P2P network of information servers having a significantly lower churn rate than typical P2P networks. Thus, we consider a scenario in which the DHT overlay is composed by information servers only, which ensures a high stability of the overlay even in large-scale networks.

The work that most relates to DQ-DHT is the Structella system designed by Castro et al. [10]. Structella replaces the random graph of Gnutella with the structured overlay of Pastry [11], while retaining the content placement of unstructured P2P systems to support complex queries. Queries in Structella are propagated using either constrained flooding or random walks. Each node receiving a query evaluates it against the local content and sends matching content back to the query originator. Beyond Structella, a few other works broadly relate to DQ-DHT for their combined use of structured and unstructured P2P techniques (see for example [12] and [13]).

In this paper we describe the DQ-DHT algorithm using Chord as DHT overlay. We analyze the performance of DQ-DHT through simulations under different algorithm configurations. We also compare the performance of DQ-DHT with that of DQ in unstructured networks. The simulation results show that DQ-DHT generates much less network overhead (i.e., number of messages) than DQ, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare.

The rest of the paper is organized as follows. Section 2 provides a background on the technique of broadcast over a DHT exploited by DQ-DHT. Section 3 describes the DQ-DHT algorithm. Section 4 analyzes its performance and compares DQ-DHT with DQ. Finally, Section 5 concludes the paper.

2 Broadcast over a DHT

This section briefly describes the Chord-based implementation of the broadcast algorithm designed by El-Ansary et al., as it is proposed in [8].

Chord uses a consistent hash function to assign each node an m-bit identifier, which 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 $\log_2 N$ [1]. 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 perform the broadcast of a data item D, a node x sends a Broadcast message to all its unique fingers. The Broadcast 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*, to 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 after exactly $N - 1$ messages, with $\log_2 N$ steps. The overall broadcast procedure can be viewed as the process of passing the data item through a spanning tree, rooted at the querying node, which covers all nodes in the network. Since the spanning tree corresponds to the lookup tree, which is a *binomial tree* in a (fully populated) Chord network [14], also the spanning tree associated to the broadcast over a fully populated Chord ring is a binomial tree.

3 Dynamic Querying over a DHT

In short, the DQ-DHT algorithm works as follows. Let x be the node that initiates the search, U the set of unique fingers not yet visited, and R_d the desired number of results. Initially U includes all unique fingers of x. Node x starts by choosing a subset V of U and sending 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 broadcast algorithm described above. When a node receives a query, it checks for local items matching the query criteria and, for each matching item, 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. Then, if the current number of received query hits R_c is equal or greater than R_d, x terminates. Otherwise, an iterative procedure takes place.

At each iteration, node x: 1) calculates the item 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 contain at least H_q nodes; 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. Note that, if the item popularity is properly estimated after the first phase of search, only one additional iteration may be sufficient to obtain the desired number of query hits.

An important point in DQ-DHT is estimating the number of nodes present in the different subtrees, and at different levels, of the spanning tree associated to the broadcast process. In the next section we discuss how we calculate such properties of the spanning tree and introduce some functions that are used in the algorithm (described in Section 3.2).

3.1 Properties of the Spanning Tree Associated to the Broadcast Process

As recalled in Section 2, the spanning tree associated to the broadcast over a fully populated Chord ring is a binomial tree. A binomial tree of order $i \geq 0$, B_i, consists of a root with i subtrees, where the j^{th} subtree is a binomial tree of order $j-1$, with $1 \leq j \leq i$. Given a binomial tree B_i, the following properties can be proven [15]: 1) The number of nodes in B_i is 2^i; 2) The depth of B_i is i; 3) The number of nodes at level l in B_i is given by the binomial coefficient $\binom{i}{l}$.

Given the binomial tree properties, we can calculate the properties of the spanning tree associated to a broadcast initiated by a node having u unique fingers (see Table 1).

Basically, in Table 1 we correct the binomial tree properties by a factor $c = N/2^u$, where N is the number of nodes in the network (which can be estimated [16]), to compensate the fact that the value of u may be different from the value of $\log_2 N$ in case of not fully populated rings. Note that, since the value of D_i may be not an integer, we use the generalized binomial coefficient to calculate N_i^l.

Table 1. Properties of the spanning tree rooted at a node with u unique fingers $F_1..F_u$.

Notation	Description	Value
N_i	Number of nodes in the subtree rooted at F_i, where $1 \leq i \leq u$	$2^{i-1} \times c$
D_i	Depth of the subtree rooted at F_i, where $1 \leq i \leq u$	$\log_2 N_i$
N_i^l	Number of nodes at level l of the subtree rooted at F_i, where $1 \leq i \leq u$ and $0 \leq l \leq D_i$	$\binom{D_i}{l}$

Based on the spanning tree properties defined in Table 1, we define in Table 2 some aggregate functions operating on a set of unique fingers. Such functions are used in the DQ-DHT algorithm presented in the next section.

Table 2. Aggregate functions operating on a set V of n unique fingers with indices $i_1..i_n \in [1, u]$.

Function	Returned result	Value
$\mathrm{N}(V)$	Total number of nodes in the subtrees associated to the unique fingers in V	$\sum_{i=i_1..i_n} N_i$
$\mathrm{D}(V)$	Depth of the subtree associated to the unique finger with highest index in V	D_i where $i = \max(i_1..i_n)$
$\mathrm{N}(V, L)$	Total number of nodes from level 0 to level L of the subtrees associated to the unique fingers in V	$\sum_{i=i_1..i_n} \sum_{l=0}^{l_i} N_i^l$ where $l_i = \min(L, D_i)$

3.2 DQ-DHT Algorithm

DQ-DHT defines two procedures: SubmitQuery, executed by a node to submit a query, and ProcessQuery, executed by a node receiving a query to process.

SubmitQuery (see Fig. 1) receives the query Q and the desired number of results R_d. It makes use of the functions defined in Table 2, and it is assumed that the procedure is executed by a node x.

The procedure starts by initializing to 0 the current number of results R_c (*line 1*). The value of R_c is incremented by 1 whenever a query hit is received. A set U is initialized to contain all unique fingers of node x (*line 2*), and H_t is set to $\mathrm{N}(U)$, which corresponds to the total number of hosts that can be queried in the network (*line 3*). The first subset V of fingers to visit is selected from U (*line 4*), and U is updated accordingly (*line 5*).

Afterwards, an integer L between 0 and $\mathrm{D}(V)$ is chosen (*line 6*). The value of L represents the last level of the subtrees associated to V from which to wait a response before to estimate the item popularity. The amount of time T_L needed to receive a response from those levels is then calculated as $T_H \times (L + 2)$, where T_H is the average time to pass a message from node to node (*line 7*). The value $L + 2$ is obtained by counting one hop to pass the message from x to the fingers,

```
procedure SUBMITQUERY($Q, R_d$)
 1: $R_c \Leftarrow 0$
 2: $U \Leftarrow$ all unique fingers of node $x$
 3: $H_t \Leftarrow \mathrm{N}(U)$
 4: $V \Leftarrow$ a subset of $U$
 5: $U \Leftarrow U \setminus V$
 6: $L \Leftarrow$ an integer $\in$ [0, $\mathrm{D}(V)$]
 7: $T_L \Leftarrow T_H \times (L + 2)$
 8: SEND($Q, V$)
 9: sleep($T_L$)
10: $H_v \Leftarrow \mathrm{N}(V, L)$
11: $T_r \Leftarrow T_H \times (\mathrm{D}(V) - L)$
12: while $R_c < R_d$ and $U \neq \emptyset$ do
13:     if $R_c > 0$ then
14:         $P \Leftarrow R_c / H_v$
15:         $H_d \Leftarrow R_d / P$
16:     else
17:         $H_d \Leftarrow H_t + 1$
18:     end if
19:     if $H_d \leq \mathrm{N}(V)$ then
20:         sleep($T_r$)
21:         $H_v \Leftarrow \mathrm{N}(V)$
22:         $T_r \Leftarrow 0$
23:     else
24:         $H_q \Leftarrow H_d - \mathrm{N}(V)$
25:         if $H_q > \mathrm{N}(U)$ then
26:             $V' \Leftarrow U$
27:         else
28:             $V' \Leftarrow$ subset of $U$ with min. $\mathrm{N}(V') \geq H_q$
29:         end if
30:         $U \Leftarrow U \setminus V'$
31:         $T_{V'} \Leftarrow T_H \times (\mathrm{D}(V') + 2)$
32:         SEND($Q, V'$)
33:         sleep($\max(T_{V'}, T_r)$)
34:         $H_v \Leftarrow \mathrm{N}(V) + \mathrm{N}(V')$
35:         $V \Leftarrow V'$
36:         $T_r \Leftarrow 0$
37:     end if
38: end while

subroutine SEND($Q, V = \{F_{i_1}..F_{i_n}\}$)
 1: for $i = i_1$ to $i_n$ do
 2:     if $i < u$ then
 3:         limit $\Leftarrow F_{i+1}$
 4:     else
 5:         limit $\Leftarrow x$
 6:     end if
 7:     send message $M = \{x, Q, limit\}$ to node $F_i$
 8: end for
```

Fig. 1. The SUBMITQUERY procedure.

L hops to propagate the message up to level L, and an additional hop to return the query hit to node x.

Then, Q is sent to all fingers in V invoking the subroutine SEND described below (*line 8*). After the wait (*line 9*), the number of nodes visited H_v is initialized to $\mathrm{N}(V, L)$ (*line 10*). While the popularity will be estimated considering only levels from 0 to L, the query continues to be forwarded up to level $\mathrm{D}(V)$. The additional amount of time T_r that would be necessary to get a response from the remaining levels is therefore proportional to $\mathrm{D}(V) - L$ (*line 11*).

After this first phase, an iterative process takes place while $R_c < R_d$ and there are more fingers to visit ($U \neq \emptyset$) (*line 12*). If at least one result has been received, node x estimates the item popularity P (*line 14*), and the estimated number H_d of hosts to obtain R_d results based on P (*line 15*). Otherwise (i.e., $R_c = 0$), H_d is set to $H_t + 1$, meaning that it is likely that *more than* all available hosts must be contacted to hit R_d results (*line 17*).

If $H_d < \mathrm{N}(V)$, it is expected to receive enough results from the fingers that have been already contacted. Note that this may happen only if $L < \mathrm{D}(V)$, because P is estimated on the basis of the results arriving from nodes up to level L of the subtrees associated to V. Thus, only in this case, the search initiator must wait for the additional amount of time T_r (*line 20*). After the wait, the value of H_v is updated to include all nodes in V (*line 21*), and T_r is set to 0 (*line 22*).

Otherwise ($H_d > \mathrm{N}(V)$), the number of nodes to be queried H_q is given by H_d minus the number of nodes already queried (*line 24*). If H_q is greater than the number of nodes available, the new set V' of fingers to visit is set to U (*line 26*). Else, V' is the subset of U with the minimum value of $\mathrm{N}(V')$ which is greater or equal to H_q (*line 28*). The elements in V' are removed from U (*line 30*), and the time $T_{V'}$ needed to receive response from all levels of the subtrees associated to V' is calculated (*line 31*).

After sending the query to all nodes in V' (*line 32*), x performs a wait (*line 33*), updates the number of hosts visited (*line 34*), and sets V to V' (*line 35*). The waiting time on *line 33* is the maximum between $T_{V'}$ and T_r, for managing the case in which the time T_r needed to visit the levels remaining from the previous phase is greater than the time $T_{V'}$ needed to receive a response from all levels in V'. As for *lines 19-22*, this may happen only on the first iteration, since after that the timeout is always set to be proportional to $\mathrm{D}(V')$, and so $T_r = 0$ (*line 36*).

The subroutine SEND forwards the query Q to a set of unique fingers V. Basically, it implements the procedure executed by a node x to perform a broadcast (see Section 2). The only difference is that we do not send the message to all unique fingers of x, but only to those in V. The message M sent by x to a node y includes the Id of the querying node (x), the query to be processed Q, and the *limit* parameter used to restrict the forwarding space of node y.

PROCESSQUERY (see Fig. 2) is executed by a node y that receives a message M containing the Id of the search initiator x, the query to process Q, and the *limit* parameter.

```
procedure ProcessQuery($M = \{x, Q, limit\}$)
 1: for $i = 1$ to $u$ do
 2:     if $F_i \in ]y, limit[$ then
 3:         if $i < u$ then
 4:             $oldLimit \Leftarrow limit$
 5:             $limit \Leftarrow F_{i+1}$
 6:             if $limit \notin ]y, oldLimit[$ then
 7:                 $limit \Leftarrow oldLimit$
 8:             end if
 9:         else
10:             $limit \Leftarrow y$
11:         end if
12:         send message $M = \{x, Q, limit\}$ to node $F_i$
13:     else
14:         exit for
15:     end if
16: end for
17: for each local item matching $Q$ do
18:     send query hit to node $x$
19: end for
```

Fig. 2. The PROCESSQUERY procedure.

The procedure broadcasts the query to all nodes in the portion of the spanning tree node y is responsible for (*lines 1-16*), following the broadcast algorithm described in Section 2. Then, it processes the query against its local resources, and for each matching item sends a query hit directly to the search initiator (*lines 17-19*).

4 Performance Evaluation

We evaluate DQ-DHT in terms of two performance parameters: *number of messages* (N_m) and *search time* (T_s). N_m is the total number of messages generated during the search process, while T_s is the amount of time needed to receive the desired number of results.

The system parameters are: the number of nodes in the network (N) and the resource replication rate (r), where r is the ratio between the total number of resources satisfying the query criteria and N. The algorithm parameters are: the initial set of unique fingers to visit (V), the initial number of levels (L), and the desired number of results (R_d). Even if it is possible to choose V to include an arbitrary subset of the unique fingers of the querying node, we consider the case in which $V = \{F_i\}$, i.e., V includes only the i^{th} unique finger, where $1 \leq i \leq u$. This permits to have, after the probe query, still $u-1$ unique fingers from which to choose the new set of subtrees to query, this way improving the granularity of search.

To analyze the message and time complexity of the algorithm we consider the following worst case scenario: at each iteration (including the probe query) the querying node chooses exactly one unique finger to contact, among those not yet contacted. Therefore, the overall search process will complete in u iterations. Since all subtrees are queried one after another, $N_m = N-1$, and so the message complexity is $O(N)$. In the same scenario the search time is the sum of the times needed to query all subtrees in sequence, i.e., $T_s = T_H \times \sum_{i=1}^{u}(D_i + 2)$, where T_H is the average time per hop. From Table 1, $D_i = \log N_i = \log(2^{i-1} \times c) = i - 1 + \log c$, where $c = N/2^u$. Thus, $T_s = T_H \times \sum_{i=1}^{u}(i + 1 + \log c) = T_H \times (\frac{1}{2}u^2 + \frac{3}{2}u + (\log c)u)$. Since on average $u = \log N$, we obtain that $T_s = T_H \times (\frac{1}{2}\log^2 N + \frac{3}{2}\log N)$. Therefore, the time complexity in the worst case is $O(\log^2 N)$.

The worst case scenario considered above is based on the very pessimistic assumptions that, at each iteration, the current estimated value of the resource popularity determines the inclusion in the next set V of exactly one unique finger among those still available. In a more typical scenario, assuming a uniform distribution of the matching resources across nodes, the popularity can be estimated with enough accuracy during the probe query, thus allowing most searches to complete in two iterations. In such two-iteration scenario the maximum search time is the sum of the probe query time (which is proportional to $L + 2$) plus the time to cover the deepest subtree that can be chosen for the second iteration (i.e., the subtree associated to F_u): $T_s = T_H \times ((L + 2) + (D_u + 2))$. Since on average $D_u = \log N - 1$, $T_s = T_H \times ((L + 2) + (\log N + 1))$ and so the time complexity in such scenario is $O(\log N)$.

4.1 Simulation Analysis

We experimentally evaluated the behavior of DQ-DHT in different scenarios using a discrete-event simulator. All the tests have been performed in a randomly-generated Chord network with $N = 50000$ nodes and a value of r ranging from

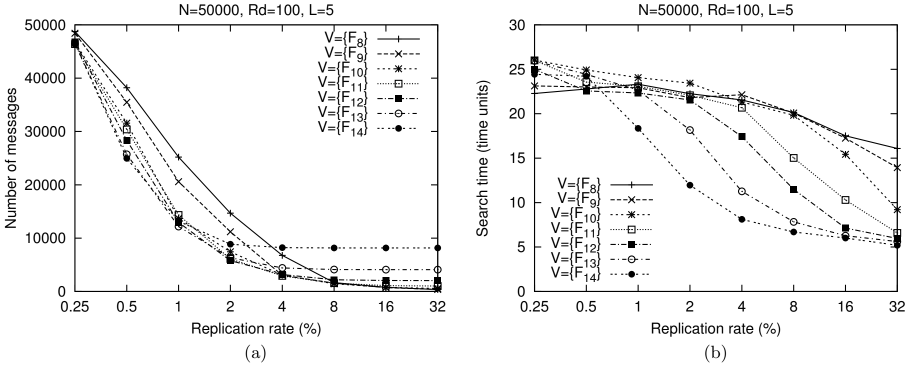

Fig. 3. Effect of varying the initial set V, with $L = 5$ and $R_d = 100$: (a) number of messages; (b) search time.

0.25 % to 32 %. Different combinations of the algorithm parameters V, L, and R_d have been experimented. All the results presented in the following are calculated as an average of 100 independent simulation runs, where at each run the search is initiated by a randomly chosen node.

We run a first set of simulations to evaluate the behavior of DQ-DHT varying the initial set V of unique fingers to contact. At each run we chose V to include one of the fingers between F_8 to F_{14}, with the initial value of L fixed to 5, and R_d set to 100. The graphs in Fig. 3 show number of messages and search time in function of the replication rate. The search time is expressed in time units, where one time unit corresponds to the average time to pass a message from node to node.

As expected, Fig. 3a shows that the number of messages decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average number of messages passes from 48735 for $r = 0.25\%$, to 360 for $r = 32\%$. In the opposite case, $V = \{F_{14}\}$, the number of messages passes from 46473 for $r = 0.25\%$, to 8159 for $r = 32\%$.

For high values of r (i.e., $r = 16 - 32\%$), in most cases the probe query is sufficient to obtain the desired number of results, and so the number of messages corresponds to the number of nodes in the subtree associated to the finger in V.

For values of r lower than 2%, typically at least one additional iteration after the probe query is needed. In these cases, the generated number of messages depends on the accuracy of the popularity estimation, which is better when a higher number of nodes is queried during the probe query (that is, when V includes a finger with a high index). For instance, when $r = 1\%$, the average number of messages is 25207 for $V = \{F_8\}$, 14341 for $V = \{F_{11}\}$, and 13169 for $V = \{F_{14}\}$.

This suggests to start the search by contacting a finger with a high index (e.g., F_{14}), when it is known that the resource is “rare.” When there is no information

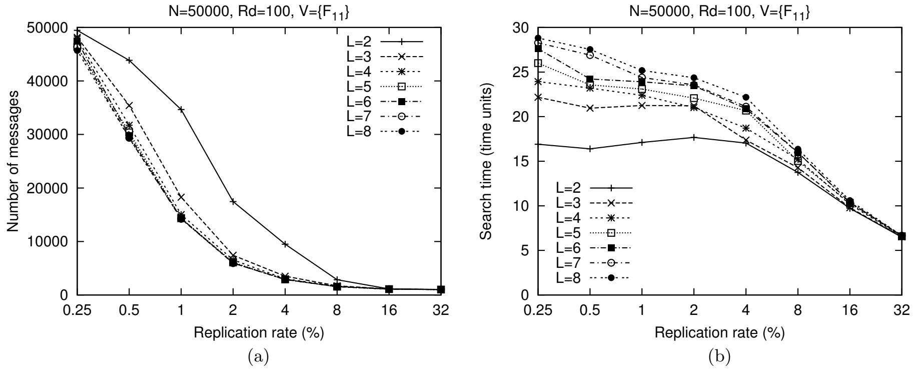

Fig. 4. Effect of varying the initial value of L, with $V = \{F_{11}\}$ and $R_d = 100$: (a) number of messages; (b) search time.

about the popularity of the resource to be found, an intermediate finger (e.g., F_{11}) should be used.

As shown in Fig. 3b, also the search time decreases as the replication rate increases, for any value of V. When $V = \{F_8\}$, the average search time passes from 22.3 for $r = 0.25\%$, to 16.1 for $r = 32\%$. When $V = \{F_{14}\}$, the search time ranges from 24.4 for $r = 0.25\%$, to 5.2 for $r = 32\%$.

The graph shows that with low values of r it is convenient to contact a finger with a high index, which leads to a lower search time with respect to fingers with a lower index. However, since the the main objective of DQ-DHT is reducing the number of messages, an intermediate finger (e.g., F_{11}) should be preferred in most cases, even if this may result to an increased search time.

We run a second set of simulations to evaluate the effect of varying the initial value of L. According to the results discussed above, we chose an intermediate finger for the probe query ($V = \{F_{11}\}$), and varied L from 2 to 8, with R_d fixed to 100. The results are presented by the graphs in Fig. 4.

Fig. 4b shows that lower values of L generate lower search times. For instance, when $r = 1\%$ the average search time passes from 17.1 with $L = 2$, to 25.2 with $L = 8$. This is mainly due to the fact that the wait after the probe phase is proportional to L, as described in Section 3.2.

On the other hand, Fig. 4a shows that very low values of L produce a significant increase in the number of messages. For example, when $r = 1\%$ the average number of messages passes from 14259 with $L = 8$, to 34654 with $L = 2$. The excess of messages in the second case is due to the reduced accuracy in the estimation of the resource popularity that is obtained considering only a few levels of the subtrees associated to V.

In general, intermediate values of L produce the best compromise between number of messages and search time. For the scenario analyzed here ($V = \{F_{11}\}$), the best result is obtained with $L = 4$, which generates a number of messages similar to that produced by higher values of L, but with a quite lower search time, as shown by the graphs in Fig. 4.

4.2 Comparison with Dynamic Querying in Unstructured Networks

In this section we compare the performance of DQ-DHT with that of DQ in unstructured networks. Since DQ-DHT is designed to work on a DHT-based network, while DQ works on unstructured networks, we adopted the following approach to compare the two systems. First, we built a random Chord network with $N = 50000$ nodes, and measured the average number of unique fingers across all nodes, which resulted to be $\bar{u} = 15.94$. Then, we built an unstructured overlay among the same nodes, in which each node is connected to $\bar{u}$ other random nodes, on the average.

As before, we measured the *number of messages* and the *search time*. In addition, we evaluated the following performance parameters: *duplication rate*, defined as the percentage of duplicate messages on the total number of messages; *success rate*, defined as the percentage of successful searches on the total number of searches performed.

For DQ in unstructured networks, we implemented the DQ+ algorithm proposed by Jiang and Jin in [17], which is an enhanced version of the original algorithm proposed by Fisk in [9]. The main difference between DQ+ and the original DQ algorithm is briefly described in the following.

In the original DQ, 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.

DQ+ adopts a “greedy” strategy. After each iteration, the querying node estimates the total number H_t of host to query, and then calculates the minimum TTL to reach H_t hosts via the next neighbor alone. To avoid overshooting of the search space, DQ+ uses a confidence interval method to estimate the popularity of the searched item. The simulation results presented in [17] show that DQ+ reduces the latency by more than four times with respect to the original DQ algorithm.

The initial parameters of DQ+ are: the number of neighbors contacted during the probe phase, n, and the TTL used for the probe query, t. We experimented two configurations: *i)* $n = 3$ and $t = 2$; *ii)* $n = 3$ and $t = 3$. In both cases, the maximum value of TTL allowed after the probe query is 5.

For DQ-DHT we chose the following configurations: *i)* $V = \{F_{14}\}$ and $L = 5$; *ii)* $V = \{F_{11}\}$ and $L = 4$. The first configuration aims at minimizing the search time, but at the cost of a higher number of messages. The second configuration provides a better balance between number of messages and search time, as discussed above.

The results of the comparison between DQ-DHT and DQ+ are presented in Fig. 5. All the simulations have been conducted with a value of r ranging from

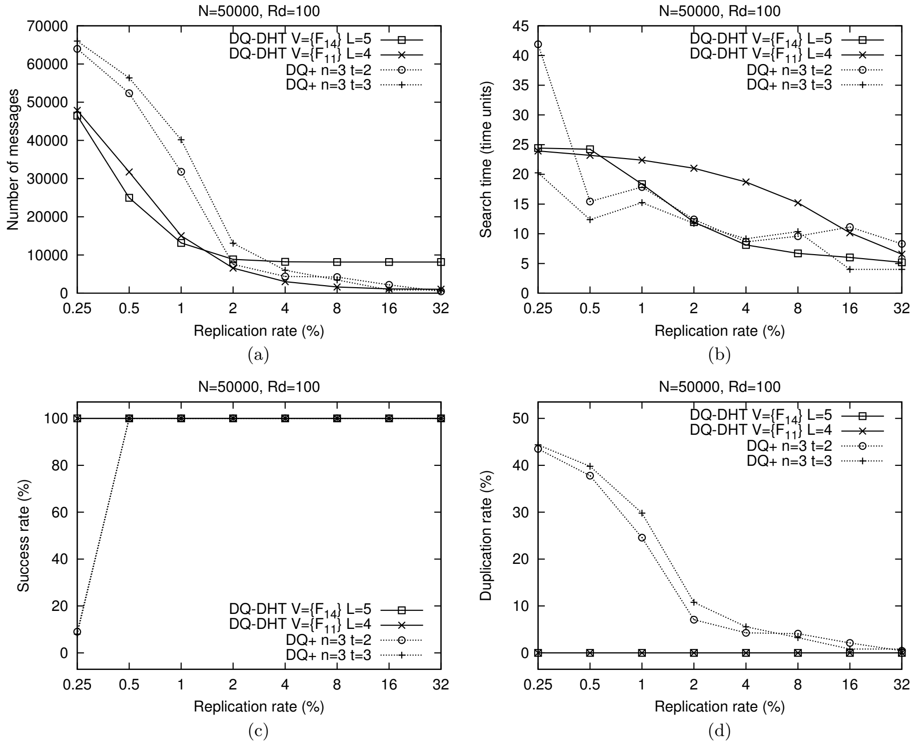

Fig. 5. Comparison between DQ-DHT and dynamic querying in unstructured networks (DQ+): (a) number of messages; (b) search time; (c) success rate; (d) duplication rate.

0.25 % to 32 %, and R_d fixed to 100. Moreover, each result is obtained as the average of 100 independent simulation runs.

As shown in Fig. 5c, the success rate for replication rates greater or equal to 0.5% is 100% with both DQ+ and DQ-DHT. However, for $r = 0.25\%$, DQ-DHT has a success rate of 100%, while DQ+ has a success rate of only 8.5%. This is due to the incomplete network coverage of the constrained flooding implemented by DQ+, which in some cases fails to find the desired number of results even if they are actually available in the network. On the contrary, DQ-DHT ensures a complete network coverage and therefore maintains a success rate of 100% even in presence of very low replication rates.

The search time in DQ+ and DQ-DHT is compared in Fig. 5b. As already discussed above, the search time in DQ-DHT strongly depends on the choice of the initial set V. With $V = \{F_{14}\}$ the search time of DQ-DHT is comparable with (and in some cases better than) the search time of DQ+. This is obtained at the cost of more messages than the case in which $V = \{F_{11}\}$, but they are much less than those generated by DQ+ for values of $r < 2\%$.

Fig. 5a shows the number of messages generated by the two algorithms. DQ-DHT with $V = \{F_{11}\}$ produces less messages than DQ+ for all values of

$r \leq 16\%$, while they generate approximatively the same number of messages for $r = 32\%$. For values of r lesser than 2% DQ-DHT outperforms DQ+ by more than a factor two. For instance, for $r = 1\%$ DQ-DHT with $V = \{F_{11}\}$ generates 15025 messages, while DQ+ with $t = 3$ produces 40155 messages.

Note that, for low replication rates, DQ-DHT generates less messages with $V = \{F_{14}\}$, while it works better with $V = \{F_{11}\}$ for high replication rates. This is due to the fact that with $V = \{F_{14}\}$ the minimum number of messages sent to the network is higher, and so more messages than needed are generated when the resource to be found is popular. On the other hand, a high number of messages ensures a better accuracy in the estimation of the resource popularity, leading to less messages when the resource to be found is rare.

The greater number of messages generated by DQ+ with respect to DQ-DHT is mainly due to the message duplication caused by flooding. The percentage of duplicate messages on the total number of messages is shown in Fig. 5d. As expected, the duplication rate of DQ+ increases as the replication rate decreases, reaching approximatively the value of 44% with $r = 0.25\%$. DQ-DHT does not suffer the message duplication problem, as each node receives the query at most once. Therefore, the duplication rate for DQ-DHT is 0% for any value of r.

In summary, the simulation results presented throughout this section show that DQ-DHT produces much less network overhead (i.e., number of messages) than DQ+, with a comparable - and in some cases better - search time, and with a higher success rate when the resource to be found is rare.

5 Conclusions

A way to support arbitrary queries in structured networks is implementing unstructured search techniques on top of DHT-based overlays. Following this approach, we proposed DQ-DHT: a P2P search algorithm that combines the dynamic querying technique with an algorithm for efficient broadcast over a DHT. DQ-DHT has been particularly designed to be used in Grid scenarios, where it is necessary to support arbitraries queries for searching resources on the basis of complex criteria or semantic features.

The behavior of DQ-DHT has been analyzed through a simulator by varying its initial configuration, in order to understand which are the best parameters to use based on user/system requirements and objectives (i.e., minimizing the number of messages or the search time). We also compared the performance of DQ-DHT with that of the enhanced dynamic querying in unstructured networks (DQ+). The simulation results show that DQ-DHT generates much less network overhead than DQ+, with a comparable (and in some cases better) search time, and with a higher success rate when the resource to be found is rare.

Acknowledgements

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

References

1. Stoica, I., Morris, R., Karger, D. R., Kaashoek, M. F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. ACM SIGCOMM'01, San Diego, USA (2001).
2. Gnutella Protocol Development. http://rfc-gnutella.sourceforge.net.
3. Andrzejak, A., Xu, Z.: Scalable, Efficient Range Queries for Grid Information Services. 2nd IEEE Int. Conf. on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden (2002).
4. Cai, M., Frank, M. R., Chen, J., Szekely, P. A.: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Journal of Grid Computing 2(1): 3-14 (2004).
5. Harren, M., Hellerstein, J. M., Huebsch, R., Loo, B. T., Shenker, S., Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks. 1st Int. Workshop on Peer-to-Peer Systems (IPTPS'02), Cambridge, USA (2002).
6. Castro, M., Costa, M., Rowstron A.: Debunking Some Myths About Structured and Unstructured Overlays. 2nd Symp. on Networked Systems Design and Implementation (NSDI'05), Boston, USA (2005).
7. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making Gnutella-like P2P Systems Scalable. ACM SIGCOMM'03, Karlsruhe, Germany (2003).
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. Fisk, A.: Gnutella Dynamic Query Protocol v0.1. http://www9.limewire.com/developer/dynamic_query.html (2003).
10. Castro, M., Costa, M., Rowstron A.: Should we build Gnutella on a structured overlay? Computer Communication Review 34(1): 131-136 (2004).
11. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Middleware 2001, Heidelberg, Germany (2001).
12. Loo, B. T., Huebsch, R., Stoica, I., Hellerstein, J. M.: The Case for a Hybrid P2P Search Infrastructure. 3rd Int. Workshop on Peer-to-Peer Systems (IPTPS'04), La Jolla, USA (2004).
13. 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).
14. Chou, J. C. Y., Huang, T.-Y., Huang, K.-L., Chen, T.-Y.: SCALLOP: A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol. IEEE Trans. Parallel Distrib. Syst. 17(5): 419-433 (2006).
15. Preiss, B. R: Data Structures and Algorithms with Object-Oriented Design Patterns in C++. John Wiley & Sons (1998).
16. Binzenhöfer, A., Staehle, D., Henjes, R.: Estimating the size of a Chord ring. Technical Report 348, Institute of Computer Science, University of Würzburg (2005).
17. Jiang, H., Jin, S.: Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks. 13th IEEE Int. Conf. on Network Protocols (ICNP 2005), Boston, USA (2005).