IMPLEMENTING DYNAMIC QUERYING SEARCH IN K-ARY DHT-BASED OVERLAYS

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

Ali Ghodsi and Seif Haridi*
Swedish Institute of Computer Science
P.O. Box 1263, 164 29 Kista, Sweden
ali@sics.se
seif@sics.se

Abstract Distributed Hash Tables (DHTs) provide scalable mechanisms for implementing resource discovery services in structured Peer-to-Peer (P2P) networks. However, DHT-based lookups do not support some types of queries which are fundamental in several classes of applications. A way to support arbitrary queries in structured P2P networks is implementing unstructured search techniques on top of DHT-based overlays. This approach has been exploited in the design of DQ-DHT, a P2P search algorithm that combines the dynamic querying (DQ) technique used in unstructured networks with an algorithm for efficient broadcast over a DHT. Similarly to DQ, DQ-DHT dynamically adapts the search extent on the basis of the desired number of results and the popularity of the resource to be found. Differently from DQ, DQ-DHT exploits the structural constraints of the DHT to avoid message duplications. The original DQ-DHT algorithm has been implemented using Chord as basic overlay. In this paper we focus on extending DQ-DHT to work in k-ary DHT-based overlays. In a k-ary DHT, broadcast takes only $O(log_k N)$ hops using $O(log_k N)$ pointers per node. We exploit this "k-ary principle" in DQ-DHT to improve the search time with respect to the original Chord-based implementation. This paper describes the implementation of DQ-DHT over a k-ary DHT and analyzes its performance in terms of search time and generated number of messages in different configuration scenarios.

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

*Also with the Department of Electronic, Computer, and Software Systems, Royal Institute of Technology, Sweden.

1. Introduction

Distributed Hash Tables (DHTs) provide scalable mechanisms for implementing resource discovery services in structured Peer-to-Peer (P2P) networks. Structured systems like Chord [1], Tapestry [2], and Pastry [3], use a DHT to assign to each node the responsibility for a specific part of the resources in the network. When a peer wishes to find a resource with a given key, the DHT allows to locate the node responsible for that key typically in $O(logN)$ hops using only $O(logN)$ neighbors per node.

As compared to unstructured search techniques like flooding or random walks, DHT-based lookups have significant scalability advantages in terms of both time and traffic [4]. However, DHT-based lookups do not support arbitrary type of queries (e.g., regular expressions [5]) 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 [6].

A way to support arbitrary queries in structured P2P networks is implementing unstructured search techniques on top of DHT-based overlays. Following this approach, an unstructured search method can be implemented over the DHT to distribute the query to as many nodes as needed; the query 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.

The approach above has been exploited in the design of DQ-DHT [7], a P2P search algorithm that combines the dynamic querying (DQ) technique used in unstructured networks [8], with an algorithm for efficient broadcast over a DHT [9].

The goal of DQ is to reduce the traffic generated by the search process in unstructured P2P networks. 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 first phase (referred to as "probe query") is to estimate the popularity of the resource to be found. 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 until all the neighbors have been queried.

Similarly to DQ, DQ-DHT dynamically adapts the search extent 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 the DHT to avoid message duplications. Performance results presented in [7] show that DQ-DHT generates much less network overhead than the enhanced DQ algo-

rithm proposed in [10], with a comparable (and in some cases better) search time, and with a higher success rate.

The original DQ-DHT algorithm has been implemented using Chord as basic overlay. In this paper we focus on extending DQ-DHT to work in k-ary DHT-based overlays [11]. In a k-ary DHT, broadcast (as well as lookup) takes only $O(log_k N)$ hops using $O(log_k N)$ pointers per node. We exploit this "k-ary principle" in DQ-DHT to improve he search time with respect to the original Chord-based implementation. This paper describes the implementation of DQ-DHT over a k-ary DHT and analyzes its performance in terms of search time and number of messages in different configuration scenarios.

The remainder of the paper is organized as follows. Section 2 briefly describes the original DQ-DHT algorithm. Section 3 describes the implementation of DQ-DHT on top of k-ary DHT-based overlays. Section 4 discusses the performance of the algorithm. Finally, Section 5 concludes the paper.

2. Dynamic Querying over a DHT

Dynamic Querying over a DHT (DQ-DHT) uses a combination of the dynamic querying technique described above with an algorithm for efficient broadcast over DHTs proposed in [9]. In this section we first describe how the algorithm of broadcast over a DHT works, and then we briefly describe the original DQ-DHT algorithm.

2.1 Broadcast over a DHT

We describe the Chord-based implementation of the broadcast algorithm, as it is presented in [9]. Chord assigns to each node an m-bit identifier that represents the position of the node 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 Chord 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 nodes 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 [9], 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 that covers all nodes in the network. As an example, Figure 1 shows the spanning tree corresponding to the broadcast initiated by Node 0 in a fully populated Chord network with $N = 64$ nodes.

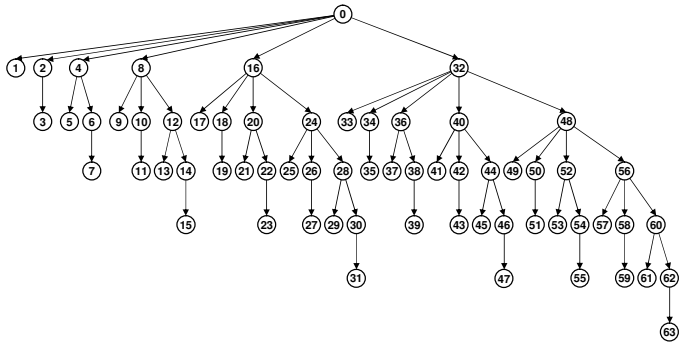

Figure 1. Spanning trees corresponding to the broadcast initiated by Node 0 in a fully populated Chord network with $N = 64$.

2.2 DQ-DHT algorithm

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 (this phase corresponds to the "probe query" of DQ). 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 probe query, only one additional iteration may be sufficient to obtain the desired number of query hits.

A key point in the implementation of DQ-DHT is estimating the properties of the spanning tree associated to the broadcast process. This can be done easily by observing that the spanning tree associated to the broadcast over a Chord network is - in the ideal case - a binomial tree [12] (see Figure 1). The basic properties of binomial trees can therefore be used to calculate with good approximation the number of nodes present in the different subtrees, and at different levels, of the spanning tree associated to the broadcast process, as shown in [7]. These values can be in turn used to calculate the number of nodes already theoretically queried, or to be queried, during the iterative dynamic querying process described above.

3. Dynamic Querying over a k-ary DHT

In a k-ary DHT, pointers are placed to achieve a time complexity of $O(log_k N)$, where N is the number of nodes in the network and k is some predefined constant. This is referred to as doing k-ary lookup or placing pointers according to the "k-ary principle" [11].

To achieve k-ary lookup, each node x keeps $n_p = (k-1)log_k(M)$ pointers (or fingers) in its finger table, where $M = k^m$ is the size of the identifier space, and m is the number of bits used for node identifiers. Each of these fingers can be chosen to be the first node that succeeds the start of every interval $f(j)$, where $f(j) = (x + c) \bmod M$, and $c = (1 + ((j-1) \bmod (k-1))) \times k^{\lfloor \frac{j-1}{k-1} \rfloor}$, for $1 \leq j \leq n_p$. It is easy to prove that for $k = 2$ intervals coincide with those of Chord.

If the identifier space is not fully populated (i.e., $N < M$), the finger table contains redundant fingers. In a network of N nodes, the number u of unique fingers of a generic node x is likely to be $(k-1)log_k(N)$.

The broadcast algorithm described in Section 2.1, which is exploited by DQ-DHT as described in Section 2.2, can also be used in a k-ary DHT. In such case, the whole broadcast process takes only $O(log_k N)$ hops.

This can be illustrated as in Section 2.1 using a spanning tree view to represent the broadcast process over a k-ary DHT. As an example, Figure 2 shows the spanning tree corresponding to the broadcast initiated by Node 0 in a fully populated k-ary DHT with $k = 4$ and $N = 64$.

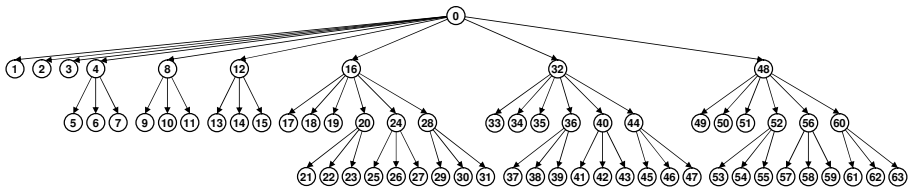

Figure 2. Spanning trees corresponding to the broadcast initiated by Node 0 in a fully populated k-ary DHT with $k = 4$ and $N = 64$.

By comparing Figure 1 with Figure 2, it can be noted that the number of hops (that is, the depth of the spanning tree) needed to complete the broadcast in k-ary DHT with $N = 64$ nodes passes from 5 with $k = 2$ (i.e., with Chord), to 3 with $k = 4$. We exploit this principle by extending DQ-DHT to improve the search time with respect to the original Chord-based implementation.

3.1 Properties of the spanning tree associated to the broadcast over a k-ary DHT

As DQ-DHT iteratively calculates the number of nodes already theoretically queried, as well as the number of nodes that must be queried to reach the desired number of results, we need to estimate the number of nodes in the different subtrees, and at different levels, of the spanning tree associated to the broadcast process.

Since for $k \neq 2$ the resulting spanning tree is no more a binomial tree, we experimentally generalized the formulas presented in [7] to be applicable to the broadcast over a k-ary DHT, for any fixed k. In particular, Table 1 show how we calculate the properties of the spanning tree associated to the broadcast process in case of fully populated identifier space.

To verify the validity of the formulas also in case of not fully populated identifier spaces, we employed a custom network simulator (the same used for the performance evaluation presented in Section 4). Through the simulator we built several k-ary DHT overlays with different values of k, and compared the real properties of the broadcast spanning tree with the ideal values calculated

Table 1. Properties of the spanning tree rooted at a node with u unique fingers $F_1..F_u$ in a fully populated k-ary DHT.

Notation	Description	Value
N_i	Number of nodes in the subtree rooted at F_i, for $1 \leq i \leq u$	$N/(k^{(\lfloor \frac{u-i}{k-1} \rfloor + 1)})$
D_i	Depth of the subtree rooted at F_i, for $1 \leq i \leq u$	$log_k(N_i)$
N_i^l	Number of nodes at level l of the subtree rooted at F_i, for $1 \leq i \leq u$ and $0 \leq l \leq D_i$	$\binom{D_i}{l} \times (k-1)^l$

using the formulas above. The results of such experiments are summarized in Figure 3.

Figure 3a compares the real (i.e., measured) and ideal (i.e., calculated) values of N_i for different values of i, in a k-ary DHT with 20000 nodes and 20-bit node identifiers, considering the following values of k: 2, 3, 5, and 8. As shown by the graph, the means of the real values (represented as points) are very close to the calculated values (represented as lines) for any value of i and k.

The graph in Figure 3b considers again a k-ary DHT with $N = 20000$ and $m = 20$, but with k fixed to 3, and compares the real and ideal values of N_i^l for different values of i, with l ranging from 1 to 4. As before, the mean of the real values resulted very close to the ideal values for any value of i and l.

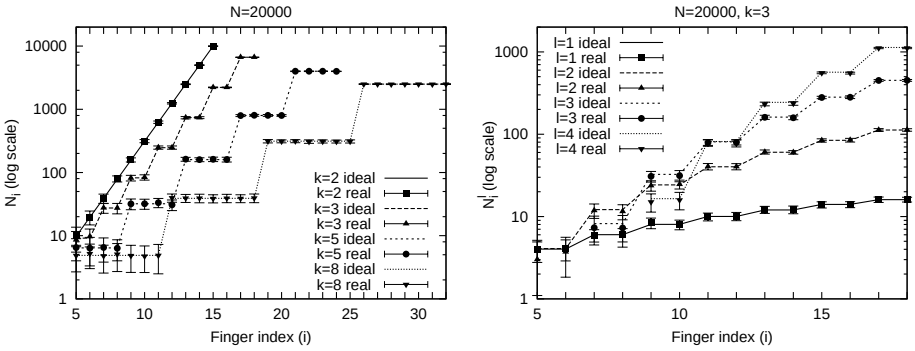

Figure 3. Comparison between ideal and real values of N_i and N_i^l for different values of k, i and l, in a simulated k-ary DHT with $N = 20000$ and $m = 20$. Lines represent the ideal values. Single points with error bars represent the real values. The error bars of the real values represent the standard deviations from the mean, obtained from 100 simulation runs. The following values of u are assumed: 15 for $k = 2$; 18 for $k = 3$; 24 for $k = 5$; 32 for $k = 8$.

In summary, the experimental results reported in Figure 3 demonstrate that the formulas reported in Table 1 can also be used to estimate - with high accuracy - the properties of the spanning tree associated to the broadcast process in not fully populated k-ary DHTs.

3.2 Minor modifications to the original DQ-DHT algorithm

The original DQ-DHT algorithm (Section 2.2) works correctly over a k-ary DHT using the formulas defined in Section 3.1. In particular: $i)$ the N_i^l formula is used during the probe query to calculate the number of nodes theoretically queried after a predefined amount of time (which corresponds to the number of nodes up to a given depth in the subtrees rooted at the fingers queried during the probe phase); $ii)$ the N_i formula is used both to calculate the number of nodes already theoretically queried (given the set of unique fingers already contacted), and to choose a new subset of unique fingers to contact based on the theoretical number of nodes to query.

Even if the original DQ-DHT algorithm works properly for any value of k, we slightly modified it to obtain a more uniform comparison of its performance when different values of k are used. The difference between the original version and the new one is explained in the following.

As discussed in Section 2.2, to perform the probe query the original algorithm needs two parameters : 1) the initial value of V, which is the first subset of unique fingers to which the query has to be sent to; and 2) L, the last level of the subtrees associated to V from which to wait a response before to estimate the resource popularity.

In the k-ary version, we replaced the two parameters above with the following: 1) H_P, defined as the number of hosts that will receive the query as a result of the probe phase; 2) H_E, the number of hosts to query before to estimate the resource popularity.

Given H_P and the set U of unique fingers of the querying node, the algorithm calculates the initial set V of unique fingers to contact as the subset of U whose associated subtrees have the minimum number of nodes greater or equal to H_P. In other terms, while in the original algorithm the fingers to contact during the probe query are chosen explicitly, in the k-ary version they are selected automatically based on the value of H_P.

While H_P indicates the total number of nodes in subtrees that will be flooded as a result of the probe phase, H_E is the minimum number of nodes that must have received the query before to estimate the resource popularity ($H_E \leq H_P$). Given H_E and the initial set V (calculated through H_P), the algorithm calculates the minimum number L of levels of the subtrees associated to V that

contain a number of nodes greater or equal to H_E. Therefore, H_E is used in the k-ary version as an indirect way of specifying the value of L.

As H_P and H_E are independent from the actual number of unique fingers and from the depth of the corresponding subtrees, their use allows to compare the algorithm performance using different values of k, independently from the number of pointers per node they produce in the resulting overlay.

4. Performance evaluation

We evaluated the performance of the algorithm using a discrete-event simulator. Two performance parameters have been evaluated: the *number of messages* and the *search time*. The first parameter is the total number of messages generated during the search process, while the second parameter is the time needed to receive the desired number of results.

The network parameters are: the number of nodes in the network N, and the resource replication rate r, defined as the ratio between the total number of resources satisfying the query criteria and N. The algorithm parameters are: H_P and H_E, introduced in the previous section, and R_d, which is the desired number of results.

We performed all the tests in a network with $N = 50000$ nodes and a value of r ranging from 0.25 % to 32 %. Different combinations of the H_P and H_E have been experimented, while R_d was fixed to 100. All the results presented in the following have been 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 in a k-ary DHT with $k = 2$ (i.e., a Chord network), with H_P fixed to 2000, and H_E ranging from 250 to 2000. The goal of these first experiment was evaluating the behavior of the algorithm (i.e., number of messages and search time) varying the number H_E of nodes that have received the query before to estimate the resource popularity.

The graphs in Figure 4 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, Figure 4a shows that the number of messages decreases as the replication rate increases, for any value of H_E. In general, the number of messages is lower for higher values of H_E. In fact, the generated number of messages depends on the accuracy of the popularity estimation, which is better when a H_E is higher. This is particularly true in presence of low replication rates. For example, the number of messages for $r = 0.5\ \%$ passes from 25889 with $H_E = 2000$, to 31209 with $H_E = 250$.

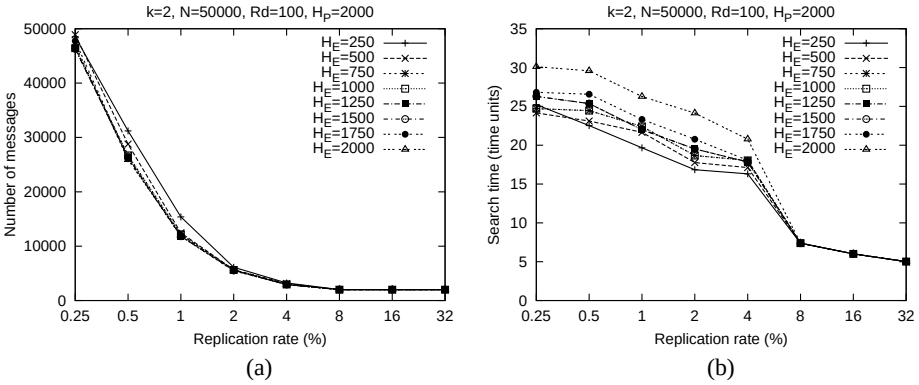

Figure 4. Effect of varying the value of H_E, with $H_P = 2000$ and $k = 2$: (a) number of messages; (b) search time.

As shown by Figure 4b, also the search time decreases as the replication rate increases. Moreover, the search time decreases as the value of H_E decreases, since lower values of H_E correspond to a lower duration of the probe query. For instance, the search time for $r = 0.5\ \%$ passes from 29.58 with $H_E = 2000$, to 22.53 with $H_E = 250$. However, since lower values of H_E generate more messages, an intermediate value of H_E should be preferred. For example, $H_E = 1000$ represents a good compromise since it generates the same number of messages of $H_E = 2000$, but with a search time close to that of $H_E = 250$.

Then, we compared the performance of the algorithm with different values of k. Based on the first set of simulations, we chosen the following algorithm parameters: $H_P = 2000$ and $H_E = 1000$. Figure 5 shows how number of messages and response time vary in this configuration with k ranging from 2 to 8.

As shown by Figure 5b, the search time strongly depends on the arity of the DHT. The maximum gain (nearly 48 %) is obtained for $r = 0.5\ \%$, with the search time passing from 24.46 with $k = 2$, to 12.74 with $k = 8$. The minimum gain (20 %) is obtained for the highest replication rate ($r = 32\ \%$), when the search time passes from 5.02 with $k = 2$, to 4.0 with $k = 8$.

The number of messages is less related to the value of k than the search time (see Figure 5a), but - in general - lower values of k generate lower number of messages. The maximum difference between $k = 2$ and $k = 8$ is reached with $r = 0.5\ \%$ (about 14 %), but it is counterbalanced by a search time gain of 48 %, as shown in Figure 5b.

We repeated the comparison above using the following configuration: $H_P = 4000$ and $H_E = 2000$. Since H_P is the minimum number of messages that will be generated during the search process, a so high value should be used when it

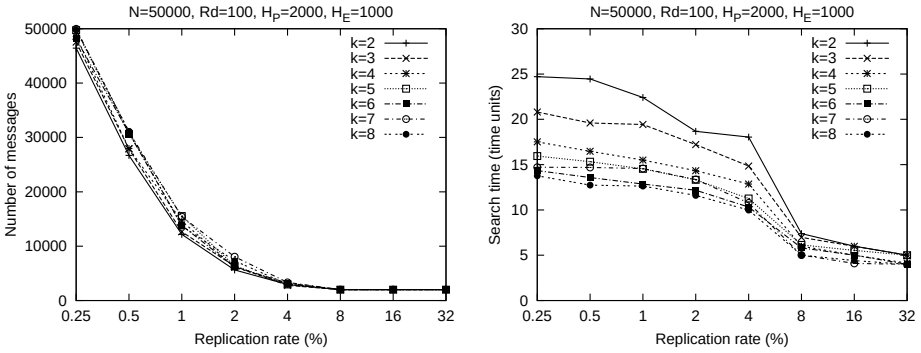

Figure 5. Effect of varying the value of k, with $H_P = 2000$ and $H_E = 1000$: (a) number of messages; (b) search time.

is fundamental to minimize the search time. The simulation results are reported in Figure 6.

The trends are similar to those shown in Figure 6. In general, the search time is lower of 1-2 units w.r.t. that measured for $H_P = 2000$ and $H_E = 1000$. For $r = 4\ \%$, the search time is significantly improved because the probe query, with $H_P = 4000$, resulted in most cases sufficient to obtain the desired number of results.

In summary, the simulation results presented above demonstrate that implementing dynamic querying over a k-ary DHT allows to achieve a significant improvement of the search time with respect to a Chord-based implementation.

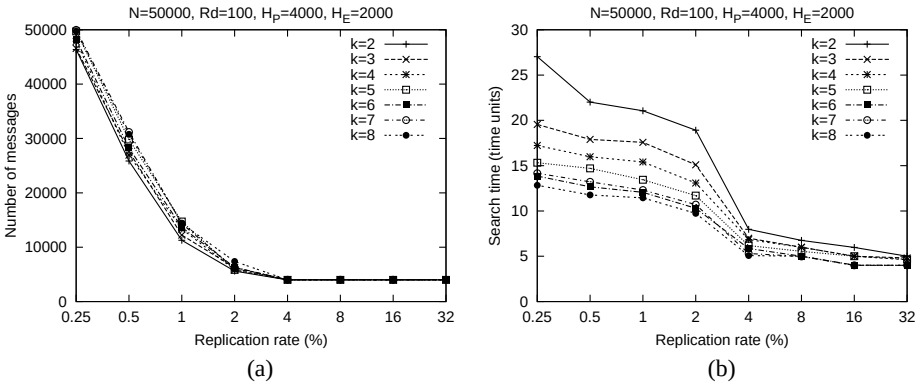

Figure 6. Effect of varying the value of k, with $H_P = 4000$ and $H_E = 2000$: (a) number of messages; (b) search time.

5. Conclusions

Implementing unstructured search techniques on top of DHT-based overlays is an efficient way to support arbitrary queries in structured P2P networks. This approach has been followed in the design of DQ-DHT [7], a P2P search algorithm that combines the dynamic querying technique used in unstructured networks with an algorithm for efficient broadcast over DHTs.

The original DQ-DHT algorithm has been implemented using Chord as basic overlay. This paper focused on extending DQ-DHT to work in k-ary DHT-based overlays [11]. As demonstrated by the experimental results presented in this paper, the "k-ary principle" allowed DQ-DHT to achieve a significant improvement of the search time with respect to the original Chord-based implementation.

Dynamic querying over a DHT can be effectively used to implement a resource discovery service in large distributed environments, as demonstrated by the dynamic querying-based Grid resource discovery system proposed in [13]. The k-ary DQ-DHT algorithm proposed in this paper could be therefore used to implement a more efficient version of that Grid system. Another application of this work could be adding the capability to perform dynamic querying search to existing distributed k-ary systems like DKS [14].

References

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

[2] B. Y. Zhao, J. D. Kubiatowicz, A. D. Joseph. Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[3] A. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Middleware 2001, Heidelberg, Germany, 2001.

[4] 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, pp. 864-878, 2007.

[5] M. Castro, M. Costa, A. Rowstron. Debunking Some Myths About Structured and Unstructured Overlays. 2nd Symp. on Networked Systems Design and Implementation (NSDI'05), Boston, USA, 2005.

[6] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker. Making Gnutella-like P2P Systems Scalable. ACM SIGCOMM'03, Karlsruhe, Germany, 2003.

[7] D. Talia, P. Trunfio. Dynamic Querying in Structured Peer-to-Peer Networks. *Submitted for publication*.

[8] A. Fisk. Gnutella Dynamic Query Protocol v0.1, May 2003. http://www9.limewire.com/developer/ dynamic_query.html

[9] S. El-Ansary, L. Onana Alima, P. Brand, S. Haridi. Efficient Broadcast in Structured P2P Networks. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, USA, 2003.

[10] H. Jiang, S. Jin. Exploiting Dynamic Querying like Flooding Techniques in Unstructured Peer-to-Peer Networks. 13th IEEE Int. Conf. on Network Protocols (ICNP 2005), Boston, USA, 2005.

[11] A. Ghodsi. Distributed k-ary System: Algorithms for Distributed Hash Tables. Ph.D. Thesis, Dept. of Electronic, Computer, and Software Systems, The Royal Institute of Technology (KTH), Stockholm, Sweden, 2006.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest. Introduction to Algorithms. MIT Press, 1990.

[13] H. Papadakis, P. Trunfio, D. Talia, P. Fragopoulou. Design and Implementation of a Hybrid P2P-based Grid Resource Discovery System. Tech. Rep. TR-0105, Institute on Architectural issues: scalability, dependability, adaptability, CoreGRID Network of Excellence, 2007.

[14] L. O. Alima, S. El-Ansary, P. Brand, S. Haridi. DKS (N, k, f): A Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications. 3rd Int. Symp. on Cluster Computing and the Grid (CCGrid 2003), Tokyo, Japan, 2003.