

Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures

Magdalena Puceva and Karl Aberer

Department of Communication Systems,
Swiss Federal Institute of Technology (EPFL),
1015 Lausanne, Switzerland
{magdalena.puceva, karl.aberer}@epfl.ch

Abstract. We investigate the search cost in terms of number of messages generated for routing queries in tree-based P2P structured systems including binary and k-ary tree structures with different arities and different degrees of imbalance in the tree shape. This work is motivated by the fact that k-ary balanced tree access structures can greatly reduce the number of hops for searching compared to the binary trees. We study to what extent the same fact is true when the tree-like structures for access in P2P environments are unbalanced. Another important issue related to P2P environments is how to build these structures in a self-organizing way. We propose a mechanism for constructing k-ary tree based decentralized access structure in a self-organizing way and based on local interactions only. The ability to search efficiently also on unbalanced k-ary trees opens interesting opportunities for load balancing as has been shown in earlier work on P-Grid, our approach to structured P2P systems.

1 Introduction

Peer-to-peer applications attracted a lot of attention within the last few years. The reason for this fast growing interest lies in the fact that they seem to be an attractive solution for creating a network with millions of users and providing a lots of desirable properties like decentralization, self-organization, robustness, scalability, autonomy etc. A variety of approaches both from research communities and companies have been proposed. Although these approaches can have different properties and also their goals can be different to some extent, they can be classified in two main categories: unstructured and structured. In unstructured systems, for which example is Gnutella [8, 10], in principle peers are unaware of the local storages that other peers in the overlay network maintain. The search is simply flooding queries to all neighboring peers all the time without acquiring any knowledge about the others peers' storages. As a consequence, they generate a large amount of messages per query which makes the approach poorly scalable in terms of communication cost when the number of peers grows. Also, despite the large number of messages generated, there is no guarantee on the search

success. It is especially difficult to find rare data objects in an unstructured P2P system. However, unstructured P2P systems have generated substantial interest because of emergent global-scale phenomena. Gnutella overlay network exhibits properties like: small diameter and power-law distribution of node's degrees, which ensures that the time-to-live for search is relatively low. Such properties have been discovered in other systems and are result of a "preferential attachment". Gnutella is completely decentralized but also self-organizing: from local interactions of peers global structure emerge.

Structured P2P systems, for which among many others examples are FreeNet [7], Chord [9], CAN [12], Pastry [14], Tapestry [13], Kademlia [11] and P-Grid [1, 5] usually assume existence of a distributed hash table (DHT). Thus, in contrast to unstructured systems, in structured systems peers maintain information about resources stored by other peers. The DHT implementation can be done in different ways: tree-like structures, multidimensional space, XOR metric etc. All these mechanisms provide possibilities to direct queries and guarantees for locating data objects within small number of messages compared to the total population size (example: $O(\log N)$ for tree-like structures). However, this relatively small search cost may increase due to unreliable peers, so the search cost is still an important issue in structured systems as well. In this paper we focus on tree-based P2P structured systems. We are interested in studying what is the search cost in terms of messages that is associated to tree-like structures with different shapes and outdegrees i.e arities. We study this problem for P-Grid a tree-based structured DHT that allows adaptation of the structure to different data distributions, and is thus particularly suitable for data management-oriented P2P architectures.

2 Problem statement

The motivation for this study is the intuitive fact that the k -ary tree structures can substantially reduce the number of hops when searching, as k -arity increases, if trees are balanced. For balanced trees the search process requires $O(\log_k N)$ hops, where k is the arity of the tree and N corresponds to the number of leaves in the tree. This is relevant for P2P systems, that are based on tree-like structures for accessing data, since larger arities k may reduce the number of messages required for routing the query. Here, we provide theoretical and experimental results regarding the search cost, for the case with balanced trees. These results for the balanced case are also generalizable to the other tree-based DHT approaches, mentioned above. Further we extend our study to unbalanced tree structures. Unbalanced trees are of particular interest for our P-Grid. They are very likely to occur in P-Grid with non-uniform data distribution due to the constructing and storage load balancing mechanism which is explained bellow. Although intuitively it seems reasonable to expect that using k -ary tree structure with greater k will reduce the number of generated messages it is not clear how much in terms of communication cost will be saved.

P-Grid [1, 5] is our version of DHT for P2P Systems. The original version of P-Grid uses binary tree-like access structure for routing queries. Its salient features, distinguishing it from other DHT approaches are:

1. Peers do not use pre-assigned identities in order to determine the search space and thus the data items they are responsible for.
2. Peers acquire in a decentralized, self-organizing process by means of bilateral interactions the search space (or search path) they support. While doing this they aim at balancing load (more precisely storage load) among peers. During bilateral interactions peers can dynamically decide to split or join subspaces, by adding or removing a bit from the search path they support. Thus the P-Grid construction and maintenance relies on the binary structure of the tree.
3. As a result of load balancing the search paths (resp. tree structure underlying P-Grid) may be heavily unbalanced since data distribution and tree shape are tightly coupled. Realistic data distributions are frequently non-uniformly distributed, e.g. following a Zipf-like distribution. In principle, this might compromise search efficiency. However, a fundamental theoretical result [3] shows that the search cost in number of messages on average remains logarithmic, more precisely strictly bounded by $\log_e N$, where N is the number of tree leaves, due to the probabilistic nature of the approach.

Though being very desirable Feature 2 and 3 appear to be tightly connected to the binary nature of the search structure underlying a P-Grid. Our goal in this paper is to verify that it is possible to maintain the salient features of P-Grid while extending the underlying data structure to k-ary tree structures.

In this paper we propose a model for constructing k-ary tree-like access structure whose construction process is based on local bilateral interactions only, similarly as in the original P-Grid. The k-ary routing structure will be compatible to the original binary P-Grid structure. The only constraint is that the arity k , in our approach, can have values that are powers of 2. The algorithm for constructing allows both the binary and the k-ary structure to be constructed simultaneously. Thus it is possible for a peer to maintain also the binary structure in addition to the k-ary, which is necessary to ensure successful termination of the search algorithm. Also, having the two structures can lead possibly to increased robustness of the system or better load balancing of the query load per peer.

We first, show analytically what is the expected number of routing messages when using balanced k-ary tree structures. Then simulation results are provided to analyze search performance when k-ary trees are used with skewed data distributions.

3 P-Grid: binary tree like structure

3.1 Data structure

As any DHT approach P-Grid is based on the idea of associating peers with data keys from a key space \mathcal{K} . Without constraining general applicability we will only

consider binary keys in the following. In contrast to other DHT approaches we do not impose a fixed or maximal length on the keys, i.e., we assume $\mathcal{K} = \{0, 1\}^*$.

In the P-Grid structure each peer $p \in Peers$ is associated with a binary key from \mathcal{K} . We denote this key by $path(p)$ and will call it the path of the peer. This key determines which data keys the peer has to manage, i.e., the keys in \mathcal{K} that have $path(p)$ as prefix. In particular the peer has to store them. In order to ensure that the complete search space is covered by peers we require that the set of peers' keys is *complete*. The set of peers' keys is complete if for every prefix s_{pre} of the path of a peer p there exists a peer p' such that $path(p') = s_{pre}$ or there exist peers p_0 and p_1 such that $s_{pre}0$ is a prefix of $path(p_0)$ and $s_{pre}1$ is a prefix of $path(p_1)$. Naturally one of the two peers p_0 and p_1 will be p itself in that case. Completeness needs to be guaranteed by the algorithms that construct and maintain the P-Grid data structure.

We do not exclude the situation where the path of one peer is a prefix of the path of another peer. This situation will occur during the construction and reorganization of a P-Grid. However, ideally this situation is avoided and any algorithm for maintaining a P-Grid should eventually converge to a state where the P-Grid is *prefix-free*, i.e., for peers p_0 and p_1 we have $path(p_0) \not\subseteq path(p_1) \wedge path(p_1) \not\subseteq path(p_0)$, where $s \subseteq s'$ denotes the prefix relationship among strings s and s' .

We also allow multiple peers to share the same paths, in that case we call the peers replicas. The number of peers that share the same path is called the *replication factor* of the path. Replication is important to support redundancy and thus robustness of a P-Grid in case of failures and to distribute workload when searching in a P-Grid.

For enabling searches peers maintain *routing tables* which are an essential constituent of the P-Grid structure. The routing tables are defined as (partial) functions $ref : Peers \times N \rightarrow \{Peers\}$ with the properties

1. $ref(p, l)$ is defined for all $p \in Peers$ and $l \in N$ with $1 \leq l \leq |path(p)|$
2. $ref(p, l) \subseteq Peers_{s_1 s_2 \dots s_{l-1} (1-s_l)}$ with $path(p) = s_1 s_2 \dots s_{l-1} s_l \dots s_k, k \geq l$

where $Peers_t = \{p \in Peers | t \subseteq path(p)\}$ for $t \in \mathcal{K}$.

The definition of P-Grid does not exclude the case where the length of the paths is up to linear in the number of peers. Therefore searches can require a linear number of messages in the worst case which would make the access structure non-scalable. The following theorem that applies to P-Grids of any shape, shows that the expected average search cost is logarithmic, however.

Theorem 1. The expected search cost for the search of a specific key t using a P-Grid P , where routing tables are randomly selected among all possible candidates, starting at a randomly selected peer s is less than $\log(|Peers|)$.

A formal proof for prefix-free P-Grids without replication of paths and of references is given in [2]. For general P-Grids with replication and which are not prefix-free simulation results have shown that the property continues to hold [4].

3.2 Construction

The original P-Grid is a binary search tree that is distributed among the participating peers. In order to construct such a tree without global coordination, the construction process depends on random bilateral peer interactions. During the interactions peers decide whether to modify the distributed tree data structure by changing their paths. All decisions are made locally based on mutual information exchange. Peers decide to change their paths depending on the data they store, aiming at more balanced data distribution after the change. A peer's path change is performed as a path extension or path retraction by one bit, which corresponds to splitting the search space in two and taking over responsibility of one of the both parts by each peer or joining two parts that have been split earlier. We refer to the path extension as a peer's "specialization", since the peer who extends its path for one bit takes over the responsibility i.e "specializes" for the data whose keys in \mathcal{K} have $path(p)$ as prefix. In addition the peer p adds a reference to the other peer in its binary routing table to cover the other part of the data space. Apparently this method relies on the fact that one uses binary trees. A more detailed description of the algorithm can be found in [4].

3.3 Search

The search algorithm for locating data keys indexed by a P-Grid is defined as follows: Each peer $p \in Peers$ is associated with a location $loc(p)$ (in the network). Searches can start at any peer. Peer p knows the locations of the peers referenced by $ref(p, l)$, but not of other peers. Thus the function $ref(p, l)$ provides the necessary routing information to forward search requests to other peers in case the searched key does not match the peer identifier. Let $t \in \mathcal{K}$ be the searched data key and let the search start at $p \in P$. Then the following recursive algorithm performs the basic search.

```
search( $t, loc(p)$ ) :=  
  if  $path(p) \subseteq t$  then return( $loc(p)$ )  
  else  
    determine maximal  $l$  such that  $t_1 \dots t_{l-1}(1 - t_l) \subseteq path(p)$ ;  
     $r$  = randomly selected element from  $ref(p, l)$ ; search( $t, loc(r)$ );
```

The algorithm $search(t, loc(p))$ always terminates successfully: due to the definition of ref the function $search$ will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $search(t, loc(p))$ the length of the common prefix of $path(p)$ and t increases at least by one. Therefore the algorithm always terminates.

In case of an unreliable network it may occur that a search cannot continue since the peer r selected from the routing table is not available. Then alternative peers can be selected from the routing table to continue the search.

4 Extending P-Grid: from binary to k-ary tree-like structure

We propose an extension of the binary tree-based DHTs to k-ary tree-based ones. The arity is determined by the parameter d which corresponds to the minimal number of query bits that can be resolved by generating one message. Generating one message here means choosing a peer from a k-ary routing table and forwarding the message to it while searching. Thus the arity is $k = 2^d$ including the peer's own path. In the binary case $d = 1$ which results in $k = 2$, while for the $d \geq 2$ the possible values of k are 4,8,16...

4.1 K-ary data structure

The description of the k-ary data structure is given using notations analogous to the previous binary case. As in the previous case, each peer $p \in Peers$ is associated with a binary key from $\mathcal{K} = \{0,1\}^*$, and is denoted by $path(p)$. Assuming that $\mathcal{B} = \{0,1\}$, the k-ary routing tables are defined as (partial) functions: $kref : Peers \times N \times N \times \mathcal{B}^d \rightarrow \{Peers\}$ with the properties:

1. $kref(p, d, c, r)$ is defined for all $p \in Peers$, $d \in N$ such that $d < |path(p)|$, for all $c \in N$ such that $1 \leq c \leq \lfloor \frac{|path(p)|}{d} \rfloor$, for all $r \in \mathcal{B}^d$
2. $kref(p, d, c, r) \in Peers_{s_1 \dots s_{(c-1)*d} r_1 \dots r_d}$ where $r = r_1 \dots r_d$ and $path(p) = s_1 s_2 \dots s_{(c-1)*d} \dots s_k$, $k \geq c * d$

The parameter d determines the arity as mentioned above. Parameters c and r correspond to the levels in the routing table.

In order to retain searchability we decide that each peer in addition to the k-ary routing table also maintains the binary routing table because:

- The k-ary structure is not guaranteed to be complete and thus it doesn't guarantee to cover the whole search space. For example assume that a peer has a path 0000 and builds a k-ary structure with $d = 2$ and a binary one. Since the binary structure has the completeness property there must be a peer with path 001. If the peer maintains only the k-ary structure he is supposed to have references to the peers with paths 0010 and 0011 whose existence is not guaranteed.
- The k-ary structure doesn't guarantee to address the highest granularity of the search space. In that case, it is not possible to address a query with a key larger than the largest existing k-ary reference. For example assume a peer has a path 00000 and maintains a k-ary structure with $d = 2$, thus the query 00001 cannot be addressed with the k-ary structure. This means, if the length of the peer's path is not a power of 2, in addition to the k-ary routing table each peer also must maintain a routing table as in the binary case that corresponds to at least the last remaining bits $\lfloor \frac{|path(p)|}{d} \rfloor * d \leq l \leq |path(p)|$.

4.2 Construction algorithm

This method is a natural extension of the P-Grid approach that uses a binary tree structure. The construction process assumes that exchanges between random peers occur. Exchanges always happen between two peers as in the previous version with binary tree.

In order to highlight the difference to the original construction process we discuss here only the mechanism used to construct the k-ary routing table entries. To that end we assume that a binary P-Grid is already constructed, but the k-ary routing tables are empty.

We observe that actually the k-ary routing tables will not be able to completely cover the search space when the P-Grid is unbalanced, a situation that results naturally from the P-Grid construction when the data distributions are skewed. In the extreme case, when the tree underlying the P-Grid is a linear tree, it is not possible to construct a k-ary routing table at all.

In Figure 1 we see an example. Assume the tree associated with peer $P1$ also corresponds to the tree underlying the construction of the P-Grid, i.e. there exist only peers with paths 00000, 00001, 0001, 001, 01, 10, and 11 and the tree is unbalanced. Then peer $P1$ can maintain a 2-ary routing table at the top level only as illustrated. For all other levels it maintains only binary routing tables. The second peer $P2$ is associated with path 11.

For filling the k-ary routing tables peers exchange information during bilateral interactions. With a passive strategy they exchange information on their own path and on their routing table entries in order to obtain new entries for their k-ary routing tables. An example of some actions taken in such a step is illustrated in Figure 1 when peer $P1$ encounters peer $P2$. $P1$ can enter immediately the address of peer $P2$ into its 2-ary routing table. In addition it can copy an entry $P3$ from $P2$'s (binary) routing table.

We also considered a proactive construction method which takes place in addition to the passive construction. When a peer reaches certain level determined by d , it sends information about the newly obtained references up the hierarchy till the level just after previous level for which k-ary level was constructed. This speeds up the construction of k-ary routing tables at the expense of additional messages.

Regarding the maintenance of the entries in the routing tables, the correcting of the entries in the routing tables both k-ary and binary can be made as a result of the querying process. The query message contains the searched key t and the number of already resolved bits. Thus the peer who receives the message can immediately determine whether the peer who sends the message has the correct routing entry. A similar mechanism was proposed for a k-ary extension of the Chord-based P2P system [6]. The efficiency of such an approach will depend on the frequencies of queries and updates and is yet to be evaluated.

4.3 Search using the k-ary and the binary routing tables

The search algorithm presented here uses a combination of both the k-ary and binary routing tables. The idea is to guarantee searchability which is not possible

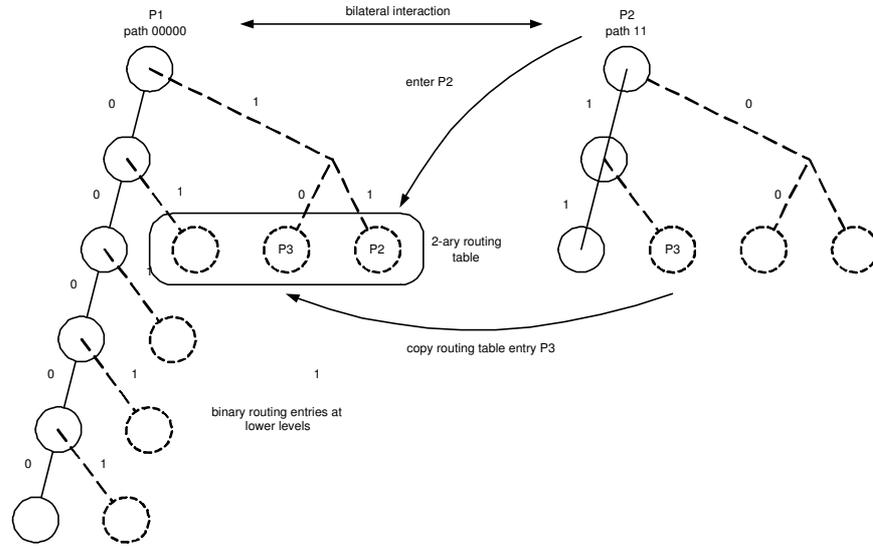


Fig. 1. P-Grid data structure example

by using the k -ary routing tables only, as discussed above. The search algorithm tries first to find an appropriate reference in the k -ary routing table if such a reference exist. In case it doesn't exist a reference from the binary routing table is selected. The function $prefix(t, d_1, d_2)$ extracts from string t the bits from position d_1 to d_2 .

```

ksearch( $t, loc(p)$ ) :=
if  $path(p) \subseteq t$  then  $return(loc(p))$ 
else
  determine maximal  $l$  such that  $t_1 \dots t_{l-1}(1 - t_l) \subseteq path(p)$ ;
   $r_k = kref(p, d, \lceil l/d \rceil, prefix(t, d * (\lceil l/d \rceil - 1), d * \lceil l/d \rceil))$ ;
  if  $r_k$  is not empty then  $r =$  randomly selected element from  $r_k$ 
   $ksearch(t, loc(r))$ 
else
   $r =$  randomly selected element from  $ref(p, l)$ 
   $search(t, loc(r))$ ;

```

In the following we give a result on the expected number of messages generated by using this algorithm for the balanced case. Whether this result generalizes to

the unbalanced case, as it has been shown for binary P-Grids by Theorem 1, is the question we will evaluate experimentally in the succeeding section.

Theorem 2. The expected number of messages for a balanced P-Grid, generated by the search algorithm that uses both k-ary and binary routing tables, when peers have paths of lengths l_p and the k-ary routing tables, defined by the parameter d , are complete is : $Div(l_p, d) * (1 - (\frac{1}{2})^d) + Mod(l_p, d) * \frac{1}{2}$

Proof: For showing the claim we first show that in a k-ary tree based P2P system that consists of N peers with different paths and each has complete k-ary routing tables and paths and queries are uniformly distributed the expected number of messages per query is $log_k N * (1 - \frac{1}{k})$. It is assumed that the keys for both the peer's paths and queries are based on a set consisting of k different symbols. When a query arrives at a peer the probability that the peer itself has the answer is $\frac{1}{k}$ and thus the probability that the message is needed is $(1 - \frac{1}{k})$. Since the depth of the tree is $log_k N$ the expected number of messages per query is $log_k N * (1 - \frac{1}{k})$. Now, the first part of the claim regarding the combined search method corresponds to search using the k-ary routing tables. $Div(l_p, d)$ is the depth of the k-ary tree routing table. The probability that peer who receives the query has the answer itself means that d bits from the query should match the corresponding d bits from the peer's path which is $(\frac{1}{2})^d$. The second part corresponds to searching for the remaining bits of the query using the binary tree structure.

5 Experiments

In order to gain a better understanding on how much communication cost in terms of messages can be saved by using the proposed approach with routing tables of higher arities, we made a simulation of the algorithms in Mathematica 4.2. Many experiments were conducted to evaluate the search performances under different conditions.

The following list of parameters and settings are considered to have an influence on the final results in terms of communication cost savings:

- Initial data distribution: we examine uniform and non-uniform distributions including Zipf-like distribution.
- Parameter d : defines the arity of the k-ary routing table
- Query distribution: we assume that the query distribution is coupled to the data distribution, i.e. all data items are queried with the same probability.

For each experiment, we present the following values:

- Number of messages generated per query using the binary routing tables only.
- Number of messages generated per query using both the k-ary and the binary routing tables.

- Expected number of messages generated per query using both the k-ary and the binary routing tables, calculated as an average value of expected values for all peers according to Theorem 2.
- The imbalance of the tree that represents the peer’s paths distribution.

Regarding the imbalance measure, we give a definition that is used in presenting the results:

Definition. The *imbalance measure of a node* is defined as the absolute difference between the numbers of leaves in its left and right subtrees. The *imbalance measure of a tree* is the sum of imbalance measures of all its nodes.

For examining the impact of the data distribution, we performed experiments with three different data distributions: uniform and two types of Zipf-like distributions. For the Zipf-like distributions, in the first case, the parameter Θ was chosen to be 0.8614 which is very frequently observed value in many situations. In the second case, higher value for Θ was used to produce more skewed data distribution. In the first case the data distribution is uniform. The size of the peer’s population was set to 256 and the number of peer interactions required for constructing the P-Grid with all k-ary routing tables. The stable state was determined by performing a large set of experiments and observing the changes in the k-ary routing structures. The system is considered to be in a stable state when no more changes occur. For each type of data distribution, the parameter d received values of 2, 3, 4 and 5 which corresponds to arities: 4, 8, 16 and 32 respectively. Using higher values of d doesn’t make sense as in that case each peer will have to store routing information about a large fraction of peers compared to the size of the peer’s population. The number of different inserted data objects was 4096. Each experiment consisted of 1000 queries with the query distribution that matches the initial data distribution. Queries were sent to randomly chosen peers.

As can be observed from the results, each data distribution results in a different imbalance of the tree of the path’s distribution. While for the uniform data distribution this value is pretty low, less than 10, for the Zipf-like distributions is much higher.

Table 1 shows the results obtained by using the uniform data distribution. It can be observed that the number of messages generated per query with the combination search method matches very well the expected number of messages calculated according to the Theorem 1. It can be observed that it drops with increasing d (arity) and reaches its minimum value for $d = 4$. However, when d is increased to 5 both the expected and the observed from the simulation values for generated messages increase. This is due to the fact that most of the peers reach path lengths of 8 in this experiment. Thus, when $d = 4$ they are able to construct 2 levels in their k-ary routing tables, while for $d = 5$ at most 1 level can be constructed. When using $d = 4$ approximately 43% of communication cost can be saved compared to the case when only binary routing tables are used for search.

Table 2 and Table 3 show the results obtained by using Zipf-like data distributions. In this case we have no theoretical result to predict the message cost.

Table 1. Influence of changing the initial data distribution

Θ_d^a	n^b	cycles ^c	delta ^d	im ^e	mb ^f	mk ^g	e ^h
0	256	15000	2	7	3.441	2.667	2.649
0	256	15000	3	3	3.318	2.269	2.252
0	256	25000	4	8	3.495	1.992	1.938
0	256	25000	5	2	3.502	2.046	2.099

^a Zipf distribution parameter for data

^b total number of peers

^c number of random exchanges

^d determines k-ary fanout

^e imbalance tree measure

^f number of messages generated using the binary routing tables

^g number of messages generated using the k-ary and binary routing tables

^h expected number of messages generated using the k-ary and binary routing tables

From the binary case we know that the expected cost should remain logarithmic. With this experiment we determine how this result generalizes when using k-ary routing tables.

Since the tree structure is unbalanced the k-ary routing tables are incomplete in these experiments. The number of levels constructed in a peer's k-ary routing table and its presence in other peers' k-ary routing tables depends on its path length. The k-ary references can be constructed only by peers with longer paths and only those can be stored as k-ary references by other peers. However, the simulation results show that the resulting number of messages generated by using both types of routing tables is pretty close to the number of messages generated with the uniform data distribution. This means that it is enough to construct k-ary routing tables only for longer paths. The saved cost, when using k-ary routing tables, is for $d = 4$ approximately 39%.

Table 2. Influence of changing the initial data distribution

Θ_d	n	cycles	delta	im	mb	mk
0.8614	256	15000	2	42	2.893	2.375
0.8614	256	15000	3	33	2.861	2.122
0.8614	256	25000	4	41	2.922	1.776
0.8614	256	25000	5	38	2.903	1.885

These experiments show that the amount of saved messages is in line with the results obtained for uniform data distributions with balanced search structures. Thus the result from Theorem 1 is most likely also to generalize when using k-

Table 3. Influence of changing the initial data distribution

Θ_d	n	cycles	delta	im	mb	mk
0.9614	256	15000	2	55	2.838	2.313
0.9614	256	15000	3	76	2.809	2.078
0.9614	256	25000	4	62	2.812	1.709
0.9614	256	25000	5	56	2.819	1.917

ary routing structures. It is worth to note that the saving in communication cost in the experiments appears to be not very substantial. However, we performed the experiments with relatively short paths, just sufficient to study the effects of using k-ary routing tables. From Theorem 2 it is clear that for longer paths the savings will be substantial with increasing values of d .

6 Conclusions and Future Work

We have proposed algorithms for constructing a k-ary access structure for P2P system in a self-organizing way and based on local bilateral interactions. We show analytically what is the expected communication cost when query distribution is uniform and the k-ary tree structure for routing is complete. Due to the skewed data distribution and our construction mechanism the k-ary structures can be incomplete. We performed a set of experiments that justify the expected cost with the complete k-ary structures. The simulations show only minor degradation in searching performances when Zipf-like data and query distributions are used. The simulated and the expected numbers of messages generated show that the increasing arity doesn't always lead to improving performances. Thus taking into account other parameters like maintenance cost it will be possible to find an optimal value for the arity of the access structure. Here we give a simulation with the P-Grid system. However, the results with the balanced tree structures are generic and applicable to other structured P2P systems.

References

1. Aberer, K.: P-Grid: A Self-organizing Access Structure for P2P Information Systems. Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy (2001)
2. Aberer, K.: Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Ecole Polytechnique Fédérale de Lausanne (EPFL), (2002). <http://www.p-grid.org/Papers/TR-IC-2002-79.pdf>
3. Aberer, K.: Scalable Data Access in P2P Systems Using Unbalanced Search Trees. Workshop on Distributed Data and Structures (WDAS 2002), Paris, France (2002).

4. Aberer, K., Datta, A., Hauswirth, M.: The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems. Technical Report IC/2003/32, Ecole Polytechnique Fédérale de Lausanne (EPFL), (2003). <http://www.p-grid.org/Papers/TR-IC-2003-32.pdf>
5. Aberer, K., Hauswirth, M., Puceva, M., Schmidt, R.: Improving Data Access in P2P Systems. IEEE Internet Computing, 6(1), Jan/Feb. (2002)
6. Alima, L.O., El-Ansary, S., Brand, P., Haridi, S.: A Framework for Peer-to-Peer Lookup Services Based on k-ary Search. SICS Technical Report T2002-06 (2002). <http://www.sics.se/~seif/Publications/SICS-T-2002-06-SE.pdf>.
7. Clarke, I., Sandberg, O., Wiley, B., Hong, W.T.: Freenet: A Distributed Anonymous Information Storage and Retrieval System. Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, number 2009 in LNCS (2001). <http://freenetproject.org/cgi-bin/twiki/view/Main/ICSI>.
8. Clip2. The Gnutella Protocol Specification v0.4. Document Revision 1.2, Jun (2001). http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
9. Dabek, F., Brunskill, E., Kaashoek, F.M., Karger, D., Morris, R., Stoica, I., Balakrishnan, H.: Building Peer-to-Peer Systems with Chord, a Distributed LookupService. Eighth Workshop on Hot Topics in Operating Systems (HotOS), (2001).
10. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and Replication In Unstructured Peer-to-Peer Networks. International Conference on Supercomputing (2002).
11. Maymounkov, P., Mazieres, P.: Kademlia: a Peer-to-Peer Information System based on XOR Metric. First International Workshop on Peer-to-Peer Systems (IPTPS)(2002).
12. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content Addressable Network. (ACM SIGCOMM)(2001).
13. Rhea, S., Wells, C., Eaton, P., Geels, D., Zhao, B., Wearherspoon, H., Kubiawitz, J.: Maintenance-Free Global Data Storage. IEEE Internet Computing, 5(5), (2001).
14. Rowstron, A., Druschel, P.: Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. ACM International Conference on Distributed Systems Platforms (Middleware), (2001).