On Using Histograms as Routing Indexes in Peer-to-Peer Systems*

Yannis Petrakis, Georgia Koloniari, and Evaggelia Pitoura

Department of Computer Science,
University of Ioannina, Greece
{pgiannis, kgeorgia, pitoura}@cs.uoi.gr

Abstract. Peer-to-peer systems offer an efficient means for sharing data among autonomous nodes. A central issue is locating the nodes with data matching a user query. A decentralized solution to this problem is based on using routing indexes which are data structures that describe the content of neighboring nodes. Each node uses its routing index to route a query towards those of its neighbors that provide the largest number of results. We consider using histograms as routing indexes. We describe a decentralized procedure for clustering similar nodes based on histograms. Similarity between nodes is defined based on the set of queries they match and related with the distance between their histograms. Our experimental results show that using histograms to cluster similar nodes and to route queries increases the number of results returned for a given number of nodes visited.

1 Introduction

The popularity of file sharing systems such as Napster, Gnutella and KaZaA has spurred much current attention to peer-to-peer (p2p) computing. Peer-to-peer computing refers to a form of distributed computing that involves a large number of autonomous computing nodes (the peers) that cooperate to share resources and services [11]. A central issue in p2p systems is identifying which peers contain data relevant to a user query. There two basic types of p2p systems with regards to the way data are distributed among peers: structured and unstructured ones.

In *structured p2p* systems, data items (or indexes) are placed at specific peers usually based on distributed hashing (DHTs) such as in CAN [13] and Chord [6]. With distributed hashing, each data item is associated with a key and each peer is assigned a range of keys and thus items. Peers are interconnected via a regular topology where peers that are close in the key space are highly interconnected. Although DHTs provide efficient search, they compromise peer autonomy. The DHT topology is regulated since all peers have the same number of neighboring peers and the selection of peers is strictly determined by the DHTs semantics. Furthermore, sophisticated load balancing procedures are required.

* Work supported in part by the IST programme of the European Commission FET under the IST-2001-32645 DBGlobe project.

W.S. Ng et al. (Eds.): DBISP2P 2004, LNCS 3367, pp. 16–30, 2005.

In *unstructured p2p* systems, there is no assumption about the placement of data items in the peers. When there is no information about the location of data items, flooding and its variation are used to discover the peers that maintain data relevant to a query. With flooding (such as in Gnutella), the peer where the query is originated contacts its neighbor peers which in turn contact their own neighbors until a peer with relevant data is reached. Flooding incurs large network overheads, thus to confine flooding, indexes are deployed. Such indexes can be either centralized (as in Napster) or distributed among the peers of the system providing for each peer a partial view of the system.

In this paper, we use a form of distributed index called routing index [3]. Each peer maintains a local index of all data available locally. It also maintains for each of its links, one routing index that summarizes the content of all peers reachable through this link within a given number of hops. We propose using histograms as local and routing indexes. Such histograms are used to route range queries and maximize the number of results returned for a given number of peers visited.

In addition, we use histograms to cluster peers that match the same set of queries. The similarity of two peers is defined based on the distance of the histograms used as their local indexes. The motivation for such clustering is that once in the appropriate cluster, all relevant to a query peers are a few links apart. In addition, we add a number of links among clusters to allow inter-cluster routing. Our clustering procedure is fully decentralized.

Our experimental results show that our procedure is effective: in the constructed clustered peer-to-peer system, the network distance of two peers is proportional to the distance of their local indexes. Furthermore, routing is very efficient, in particular, for a given number of visited peers, the results returned are 60% more than in an unclustered system.

Preliminary versions of a clustering procedure based on local indexes appears in [12] where Bloom filters are used for keyword queries on documents. The deployment of histograms as routing indexes for range selection queries, the routing procedure and the experimental results are new in this paper. As opposed to Bloom filters that only indicate the existence of relevant data, histograms allow for an ordering of peers based on the estimated results they provide to a query. This leads to a clustered p2p system in which the network distance of two peers is analogous to the estimated results.

The remainder of this paper is structured as follows. In Section 2, we introduce histograms as routing indexes and appropriate distance metrics. In Section 3, we describe how histograms are used to route queries and to cluster relevant peers. In Section 4, we present our experimental results. Finally, in Section 5, we compare our work with related research, and in Section 6 offer our conclusions.

2 Histograms in Peer-to-Peer Systems

We assume a p2p system with a set N of peers n_i. The number of peers changes as peers leave and join the system. Each peer is connected to a small number

of other peers called its *neighbors*. Peers store data items. A query q may be posed at any of the peers, while data items satisfying the query may be located at many peers of the system. We call the peers with data satisfying the query *matching* peers. Our goal is to route the query to its matching peers efficiently.

2.1 Histograms as Routing Indexes

We consider a p2p system where each peer stores a relation R with a numeric attribute x and focus on routing range selection queries on x. Our approach is based on using local indexes to describe the content of each peer. In particular, each peer n maintains a summary, called local index, that describes its content. A property of the index is that we can determine, with high probability, whether the peer matches the query based on the index of the peer, that is, without looking at the actual content of the peer. We propose using histograms as local indexes.

A histogram on an attribute x is constructed by partitioning the data distribution of x into b (≥ 1) mutually disjoint subsets called *buckets* and approximating the frequencies and values in each bucket. Histograms are widely used as a mechanism for compression and approximations of data distributions for selectivity estimation, approximate query answering and load balancing [7]. In this paper, we use histograms for clustering and query routing in p2p systems. We consider *equi-width* histograms, that is, we divide the value set of attribute x into ranges of equal width and keep the percentage of x's occurrences for each bucket. In addition, we maintain the total number of all tuples (the histogram *size*).

We denote by $LI(n)$ the histogram used as the local index of peer n. Besides its local index, each peer n maintains one routing index $RI(n, e)$ for each of its links e. $RI(n, e)$ summarizes the content of all peers that are reachable from n using link e at a distance at most R. The routing indexes are also histograms defined next.

We shall use the notation $H(n)$ to denote a histogram (used either as a local index $LI(n)$ or as a routing index $RI(n, e)$), $H_i(n)$ to denote its i-th bucket, $0 \leq i \leq b-1$, and $S(H(n))$ to denote its size. Then,

Definition 1 (Histogram-Based Routing Index). *The histogram-based routing index $RI(n, e)$ of radius R of the link e of peer n is defined as follows: for $0 \leq i \leq b-1$, $RI_i(n, e) = (\Sigma_{p \in P} LI_i(p) * S(LI(p)) / \Sigma_{p \in P} S(LI(p))$ and $S(RI(n, e)) = \Sigma_{p \in P} S(LI(p))$ where P is the set of all peers p within distance R of n reachable through link e.*

An example is shown in Fig. 1. The set of peers within distance R of n is called the *horizon* of radius R of n.

As usual, we make the *uniform frequency* assumption and approximate all frequencies in a bucket by their average. We also make the *continuous values* assumption, where all possible values in the domain of x that lie in the range of the bucket are assumed to be present. However, there is a probability that although a value is indicated as present by the histogram, it does not really exist in the data (false positive). This is shown to depend on the number of buckets,

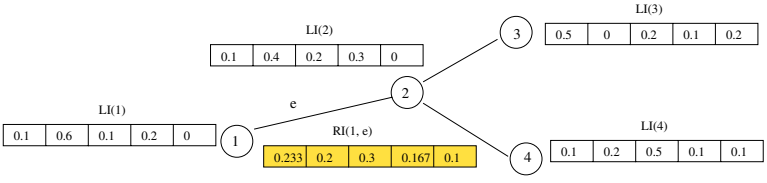

Fig. 1. The local indexes of peers 1, 2, 3, and 4 and the routing index of link e of peer 1 for radius $R = 2$, assuming that local indexes $LI(2)$, $LI(3)$ and $LI(4)$ have the same size

the number of tuples and the range of the attribute. Details can be found in the Appendix.

For a given query q, the local histogram $LI(n)$ of peer n provides an estimation of the number of results (matching tuples) of peer n, while the routing index $RI(n, e)$ provides an estimation of the number of results that can be found when the query is routed through link e. We denote by $results(n, q)$ the actual number of results to query q and by $hresults(H(n), q)$ the number of results estimated by the histogram $H(n)$. Let a query $q_k = \{x\text{: } a \leq x \leq a + k * d\}$, where d is equal to the range of each bucket, $0 \leq k \leq b$ and $a = c * d$, where $0 \leq c \leq b - 1$. We also consider the queries $q_< = \{x\text{: } x \leq a\}$ and $q_> = \{x\text{: } x \geq a\}$. Note that query $q_>$ is the same with query q_b.

We can estimate $results(n, q)$ using the histogram $H(n)$ of peer n based on the type of the query q as follows:

- q_k: $hresults(H(n), q_k) = S(H(n)) * \Sigma_{i=a/d}^{((a+k*d)/d)} H_i(n)$
- $q_<$: $hresults(H(n), q_<) = S(H(n)) * \Sigma_{i=0}^{a/d} H_i(n)$
- $q_>$: $hresults(H(n), q_>) = S(H(n)) * \Sigma_{i=a/d}^{b} H_i(n)$

We defined the query q_k as starting from the lower limit of a bucket ($a = c * d$), for simplicity.

2.2 Using Histograms for Clustering

Ideally, we would like to route each query q only through the peers that have the most number of results (top-k matching peers). To express this, we define *PeerRecall* as our performance measure. *PeerRecall* expresses how far from the optimal a routing protocol performs. Let V be a set of peers ($V \subseteq N$), by $Sresults(V, q)$ we denote the sum of the numbers of results (i.e., matching tuples) returned by each peer that belongs to V.

$$Sresults(V, q) = \Sigma_{v \in V} results(v, q) \tag{1}$$

Definition 2 (PeerRecall). *Let $Visited$ ($Visited \subseteq N$) be the set of peers visited during the routing of a query q and $Optimal$ ($Optimal \subseteq N$) be the set of peers such that $|Optimal| = |Visited|$ and $v \in Optimal \Leftrightarrow results(v, q) \geq results(u, q)$, $\forall\ u \notin Optimal$. We define PeerRecall as: $PeerRecall(q) = Sresults(Visited, q)/Sresults(Optimal, q)$.*

Intuitively, to increase *PeerRecall*, peers that match similar queries must be linked to each other. This is because, if such peers are grouped together, once we find one matching peer, all others are nearby. The network *distance* between two peers n_i and n_j, $dist(n_i, n_j)$ is the length of the shortest path from n_i to n_j. In general, peers that match similar queries should have small network distances. Our goal is to cluster peers, so that peers in the same cluster match similar queries. The links between peers in the same cluster are called *short-range* links. We also provide a few links, called *long-range* links, among peers in different clusters. Long-range links serve to reduce the maximum distance between any two peers in the system, called the *diameter* of the system. They are used for inter-cluster routing.

To cluster peers, we propose using their local indexes. That is, we cluster peers that have similar local histograms. For this to work, the distance (d) between two histograms must be descriptive of the difference in the number of results to any given query.

Property 1. Let $LI(n_1)$, $LI(n_2)$ and $LI(n_3)$ be the local indexes of three peers n_1, n_2 and n_3. If $d(LI(n_1), LI(n_2)) \geq d(LI(n_1), LI(n_3))$, then $|results(n_1, q)/S(LI(n_1))$ - $results(n_2, q)/S(LI(n_2))| \geq |results(n_1, q)/S(LI(n_1))$ - $results(n_3,q)/S(LI(n_3))|$.

That is, we want the distance of two histograms to be descriptive of the difference in the number of results they return for a given query workload. In the following, as a first step we consider how two well-known distance metrics perform with respect to the above property.

Histogram Distances. The L_1-distance of two histograms $H(n_1)$ and $H(n_2)$ is defined as:

Definition 3 (L_1 Distance Between Histograms). *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets, their L_1 distance, $d_{L_1}(H(n_1), H(n_2))$ is defined as:* $d_{L_1}(H(n_1), H(n_2)) = \Sigma_{i=0}^{b-1} |H_i(n_1) - H_i(n_2)|$.

Let us define as

$$L1(i) = H_i(n_1) - H_i(n_2). \tag{2}$$

then

$$d_{L_1}(H(n_1), H(n_2)) = \Sigma_{l=0}^{b-1} |L1(l)| \tag{3}$$

The histograms we study are *ordinal* histograms, that is, there exists an ordering among their buckets, since they are built on numeric attributes. For ordinal histograms, the position of the buckets is important and thus, we want the definition of histogram distance to also take into account this ordering. This property is called *shuffling dependence*. For example, for the three histograms of Fig. 2, the distance between histograms $H(n_1)$ and $H(n_2)$ that have all their values at adjacent buckets ($H_i(n_1)$ and $H_{i+1}(n_2)$ respectively) should be smaller than the distance between histograms $H(n_1)$ and $H(n_3)$ that have their values at buckets further apart. This is because, the difference of results for peers n_1

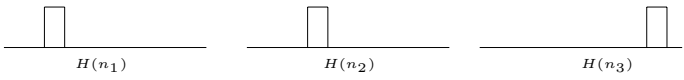

Fig. 2. Intuitively, the distance between $H(n_1)$ and $H(n_2)$ should be smaller than the distance between $H(n_1)$ and $H(n_3)$

and n_2 is smaller for a larger number of range queries than for peers n_1 and n_3. The shuffling dependence property does not hold for d_{L_1}, since the three histograms have the same pair-wise distances.

We now consider an edit distance based similarity metric between histograms for which the shuffling dependence property holds. The edit distance between two histograms $H(n_1)$ and $H(n_2)$ is the total number of all necessary minimum movements for transforming $H(n_1)$ to $H(n_2)$ by moving elements to the left or right. It has been shown that this is expressed by the following definition [2]:

Definition 4 (Edit Distance Between Histograms). *Let two histograms $H(n_1)$ and $H(n_2)$ with b buckets, their edit distance, $d_e(H(n_1), H(n_2))$ is defined as: $d_e(H(n_1), H(n_2)) = \Sigma_{i=0}^{b-1} |\Sigma_{j=0}^{i}(H_j(n_1) - H_j(n_2))|$.*

Let us define as

$$pref(l) = \Sigma_{i=0}^{l} H_i(n_1) - \Sigma_{i=0}^{l} H_i(n_2) \tag{4}$$

then

$$d_e(H(n_1), H(n_2)) = \Sigma_{l=0}^{b-1} |pref(l)| \tag{5}$$

Let a query $q_k = \{x\colon a \leq x \leq a + k * d$, where d is equal to the range of each bucket and $0 \leq k \leq b\}$.

Given that a is chosen uniformly at random from the domain of x, then the difference in the results is equal to:

$$|hresults(H(n_1), q_k)/S(H(n_1)) - hresults(H(n_2), q_k)/S(H(n_2))| = \Sigma_{j=0}^{b-1} |pref(j+k) - pref(j-1)| \tag{6}$$

where $pref(j) = 0$ for $j \geq b-1$ and $j < 0$.

From Equation 6, for $k = b-1$ that is for queries $x \geq a$ Property 1 holds. It also holds for $x \leq a$. It does not hold however, in general.

To summarize, the L_1 distance satisfies Property 1 for q_0 (that is for queries that cover one bucket), while the edit distance satisfies Property 1 for $q_<$ and $q_>$ (which is the same with q_b).

3 Query Routing and Network Construction

We describe next how histogram-based indexes can be used to route a query and to cluster similar peers together. We distinguish between two types of links:

short-range or *short* links that connect similar peers and *long-range* or *long* links that connect non-similar peers. Two peers belong to the same *cluster* if and only if there is a path consisting only of short links between them. We describe first how queries are routed and then how long and short links are created.

3.1 Query Routing

A query q may be posed at any peer n. Our goal is to route the query q through peers that give a large number of results for q. Ideally, we would like to visit only those peers that provide the most results. To maximize $PeerRecall$, we use a greedy query routing heuristic: each peer that receives a query propagates it through those of its links whose routing indexes indicate that they lead to peers that provide the largest number of results. The routing of a query stops either when a predefined number of peers is visited or when a satisfactory number of results is located. Specifically, for a query q posed at peer n:

1. First, n checks its local index and if the index indicates that there may be matching data locally, it retrieves them.
2. Then, n checks whether the maximum number of visited peers ($MaxVisited$) has been reached or the desired number of matching data items (results) has been attained. If so, the routing of the query stops.
3. Else, n propagates the query through the link e whose routing index gives the most matches ($hresults(RI(n,e),q) > hresults(RI(n,l),q)$, $\forall$ link $l \neq e$) and e has not been followed yet. If $hresults(RI(n,e),q) = 0$, $\forall$ link e that has not been followed, query propagation stops.

By following the link e whose $hresults(RI(n,e),q)$ returns the largest value, the query is propagated towards the peers with the most results and thus $PeerRecall$ is increased.

When a query reaches a peer that has no links whose routing indexes indicate a positive number of results, or when all such links have already been followed, backtracking is used. This state can be reached either by a false positive or when the desired number of results has not been attained yet. In this case, the query is returned to the previous visited peer that checks whether there are any other links with indexes with results for the query that have not been followed yet, and propagates the query through one or more of them. If there are no such matching links, it sends the query to its previous peer and so on. Thus, each peer should store the peer that propagated the query to it. In addition, we store an identifier for each query to avoid cycles. Note that this corresponds to a Depth-First traversal.

To avoid situations in which all routing indexes indicate that there are no results, initially we use the following variation of the routing procedure. If no matching link has been found during the routing of the query, and the current peer n has no matching links ($hresults(RI(n,e),\ q) = 0\ \forall$ link e of n), which means that the matching peers (if any) are outside the radius R of n, then the long-range link of this peer is followed (even if it does not match the query). The idea is that we want to move to another region of the network, since the

current region (bounded by the horizon) has no matching peers. In the case that the peer has no long-range link or we have already followed all long-range links, the query is propagated through a short link to a neighbor peer and so on until a long-range link is found.

3.2 Clustering

We describe how routing indexes can be used for distributed clustering. The idea is to use the local index of each new peer as a query and route this towards the peers that have most similar indexes.

In particular, each new peer that enters the system tries to find a relevant cluster of peers. Then, it links with a number SL of peers in this cluster through short links. Also, with probability P_l, it links with a peer that does not belong to this cluster through a long link. Short links are inserted so that peers with relevant data are located nearby in the p2p system. Long links are used for keeping the network diameter small. The motivation is that we want to be easy to find both all relevant results once in the right cluster, and the relevant cluster once in another cluster, thus increasing $PeerRecall$.

When a new peer n wishes to join the system, a join message that contains its local index $LI(n)$ is posed as a query to a well known peer in the system. The join message also maintains a list L (initially empty) with all peers visited during the routing of the join message. The join message is propagated until up to $JMaxVisited$ peers are visited.

Whenever the join message reaches a peer p the procedure is the following:

1. The distance $d(LI(n),\ LI(p))$ between local indexes $LI(n)$ and $LI(p)$ is calculated.
2. Peer p and the corresponding distance are added to list L.
3. If the maximum number of visited peers $JMaxVisited$ is reached, the routing of the join message stops.
4. Else, the distances $d(LI(n),\ RI(p,e))$ between the local index $LI(n)$ of the new peer n and the routing indexes $RI(p,e)$ that correspond to each of the links e of peer p are calculated.
5. The message is propagated through the link e with the smallest distance that has not been followed yet, because there is a higher probability to find the relevant cluster through this link. When the message reaches a peer with no other links that have not been followed, backtracking is used.

When routing stops, the new peer selects to be linked through short links to the SL peers of the list L whose local indexes have the SL smallest distances from the local index of the new peer. It also connects to one of the rest of the peers in the list through a long link with probability P_l.

An issue is how the peer that will be attached to the new peer through the long link is selected. One approach is to select randomly one of the rest of the peers within the list (that does not belong to the SL peers selected to be linked through short links). Another approach is to select one of the rest of the peers within the list with a probability based on their distances from the new

peer. Thus, we rank these peers based on their distances, where the first in the ranking is the one with the smallest distance and has $rank = 0$. The second in the ranking has $rank = 1$ and so on. The probability that a specific peer from the list is selected with respect to its ranking is: $\alpha * (1 - \alpha)^{rank}$, where $0 < \alpha < 1$. The smaller the value of α, the greater the probability to create a long link with a more dissimilar peer.

4 Experimental Results

We implemented a simulator in C to test the efficiency of our approach. The size of the network varies from 500 to 1500 peers and the radius of the horizon from 1 to 3. Each new peer creates 1 to 2 short links ($SL = 1$ or 2) and one long link with probability $P_l = 0.4$. The routing of the join message stops when a maximum number ($JMaxVisited$) of peers is visited. The routing of a query stops when a maximum number ($MaxVisited$) of peers is visited. Both numbers are set to 5% of the existing peers. Each peer stores a relation with an integer attribute $x \in [0, 499]$ with 1000 tuples. The tuples are summarized by a histogram with 50 buckets. 70% of the tuples of each peer belong to one bucket, and the rest are uniformly distributed among the remaining buckets. The tuples in each bucket also follow the uniform distribution. The input parameters are summarized in Table 1.

Table 1. Input parameters

Parameter	Default Value	Range
Number of peers	500	500-1500
Radius of the horizon	2	1-3
Number of short links (SL)	2	1-2
Probability of long link (P_l)	0.4	
Perc of peers visited during join ($JMaxVisited$)	5	
Perc of peers visited during search ($MaxVisited$)	5	
Histogram-related parameters		
Number of buckets (b)	50	
Domain of x	[0, 499]	
Tuples per peer	1000	
Range of queries	2	0-4

We compare first the two distance metrics. Then, we evaluate the clustering and the query routing procedures.

4.1 Histogram Distance Metrics

We run a set of experiments to evaluate the performance of the two histogram distance metrics (the L_1 and the edit distance). For simplicity of presentation,

in the reported experiment, we use histograms with 10 buckets and $x \in [0, 99]$. We used a workload with queries having range (k) varying from 0 (covering data in 1 bucket) to 4 (covering data in 5 buckets). We use 10 histograms $H(i)$ $0 \leq i < 10$ with 10 buckets each, that have 70% of their data in bucket i and the rest uniformly distributed among the other buckets. We compute the distance of each histogram with $H(0)$ using the two distance metrics. Our performance measure is the difference in the number of results for each histogram with $H(0)$, that is:

$$|hresults(H(n), q_k)/S(H(n)) - hresults(H(0), q_k)/S(H(0))|,\ 1 \leq n < 10$$

with respect to the distance of the respected histograms (that is, whether Property 1 is satisfied).

Figure 3(left) shows the results when the L_1 distance is used. Due to the nature of the data, all compared histograms have the same distance. The distance of the histograms has no relation with the difference in the number of results. This is because the L_1 distance compares only the respective buckets of each histogram without taking into account their neighboring buckets which however influence the behavior of queries with ranges larger than 0.

The edit distance (Fig. 3(right)) outperforms L_1. In particular, as the distance between the histograms increases, their respective differences in the results also increases. However, for each query range this occurs until some point after which the difference in the results becomes constant irrespectively of the histogram distance. This is explained as follows. The edit distance between two histograms takes into account the ordering of all buckets, while a query with range r involves only $r + 1$ buckets, and thus it does not depend on the difference that the two histograms may have in the rest of their buckets. For example, for a query with range 0, the difference in the results remains constant while the histogram distances increase. This is because the query involves only single buckets while the edit distance considers the whole histogram. Thus, the edit distance works better for queries with large ranges.

We also calculated the average performance of the two distance metrics for a mixed query workload of queries with range from 0 to 4 (Fig. 4). L_1 has the worst overall performance since although the distance between the various

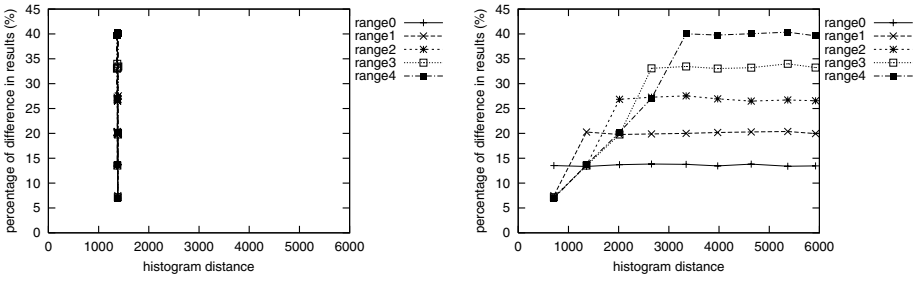

Fig. 3. Relation of the number of results returned with the histogram distance using (left) the L_1 distance and (right) the edit distance

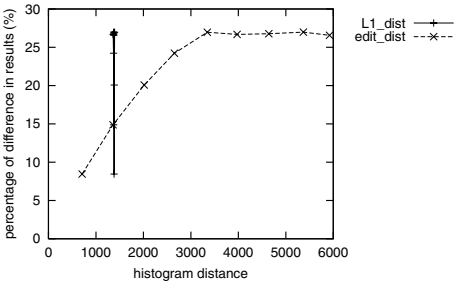

Fig. 4. Comparison of histogram distances

histograms is constant, the difference in the number of results increases. The edit distance behaves better. The difference in results increases until a point and then it becomes constant. If we continue with ranges larger than 4, this point occurs later.

4.2 Cluster Quality

In this set of experiments, we evaluate the quality of clustering. For these experiments, we assume a query workload with range 2 (whose results occupy 3 buckets). We compare the constructed clustered network with a randomly constructed p2p system, that is a p2p system in which each new peer connects randomly to an existing peer (random construction and routing) (*random*).

We measure the average histogram distance between the peers that are at various network distances from each other in the created p2p network. We use a network of 500 nodes and radius 2, and conduct the same experiment for $SL = 1$ (Fig. 5(left)) and $SL = 2$ (Fig. 5(right)). As the network distance between two peers increases, their histogram distance increases too, for both histogram distance metrics and for both 1 and 2 short links. This means that the more similar two peers are, the closer in the network they are expected to be. The rate of increase of the histogram distance is large when the network distance is small and decreases as the network distance increases, due to the

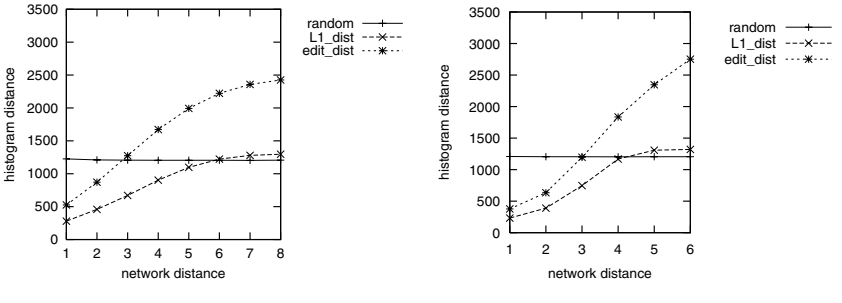

Fig. 5. Cluster quality with (left) $SL = 1$ and (right) $SL = 2$

denser clustering of similar peers in a particular area of the network (e.g., the formation of clusters of similar peers). The edit distance has a larger increase rate for large network distances (4 and above for 2 short links and 6 and above for 1 short link) than the L_1 distance (which remains nearly constant for these network distances). The conclusion is that in the network built using the edit distance, some kind of ordering among the peers in different clusters is achieved. For the random network, the histogram distance is constant for all network distances, since there is no clustering of similar peers.

4.3 Query Routing

In this set of experiments, we evaluate the performance of query routing using as our performance measure *PeerRecall* (as defined in Def. 2). We compare the constructed clustered network with a randomly constructed p2p, that is a p2p system in which each new peer connects randomly to an existing peer (random construction and routing) (*random*). We also consider a randomly constructed p2p system that uses histograms only for query routing (*random_join*).

We use a network of 500 peers and examine the influence of the horizon in the query routing performance for $SL = 1$ (Fig. 6(left)) and $SL = 2$ (Fig. 6(right)). The radius varies from 1 to 3; we use queries with range = 2. Using histograms for both clustering and query routing results in much better performance than using histograms only for routing or not using histograms at all. For radius 2 and for 2 short links, we have the best performance. For 1 short link, *PeerRecall* increases as the radius of the horizon increases, since each peer has information about the content of more peers. For 2 short links, *PeerRecall* decreases for radius greater than 2. The reason is that there are more links, and thus, much more peers are included within the horizon of a particular peer (when compared with the network built using 1 short link). Thus, a very large number of peers correspond to each routing index. This results in losing more information than when using radius 2. Thus, for each type of network there is an optimal value of the radius that gives the best performance.

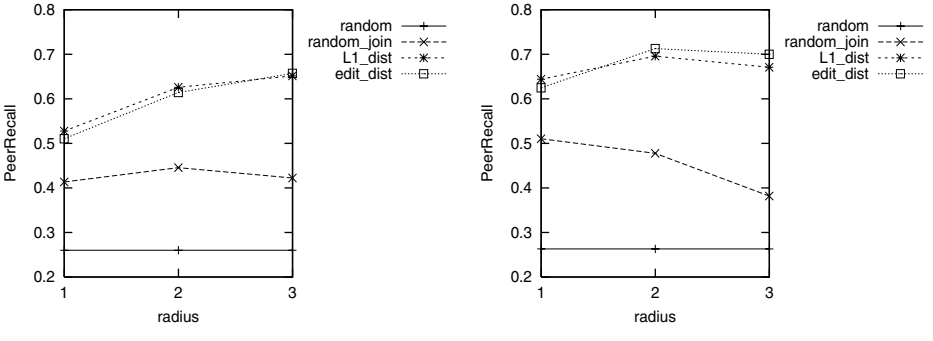

Fig. 6. Routing for different values of the radius and with (left) $SL = 1$ and (right) $SL = 2$

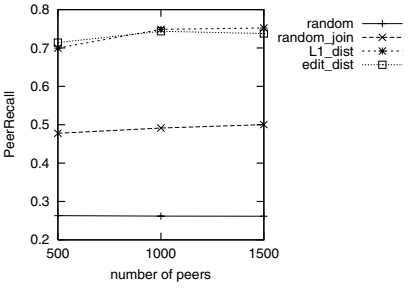

Fig. 7. Varying the number of nodes

Next, we examine how our algorithms perform with a larger number of peers. We vary the size of the network from 500 to 1500. Radius is set to 2 and we use 2 short links and queries with range = 2. As shown in Fig. 7, *PeerRecall* remains nearly constant for both histogram distance metrics and outperforms the *random_join* and the *random* networks.

5 Related Work

Many recent research efforts focus on organizing peers in clusters based on their content. In most cases, the number or the description of the clusters is fixed and global knowledge of this information is required. In this paper, we describe a fully decentralized clustering procedure that uses histograms to cluster peers that answer similar queries. In [1], peers are partitioned into topic segments based on their documents. A fixed set of C clusters is assumed, each one corresponding to a topic segment. Knowledge of the C centroids is global. Clusters of peers are formed in [17] based on the semantic categories of their documents; the semantic categories are predefined. Similarly, [4] assumes predefined classification hierarchies based on which queries and documents are categorized. The clustering of peers in [10] is based on the schemes of the peers and on predefined policies provided by human experts. Besides clustering of peers based on content, clustering on other common features is possible such as on their interests [8].

In terms of range queries, there has been a number of proposals for supporting them in structured p2p systems. In [15], which is based on CAN, the answers of previous range queries are cached at the appropriate peers and used to answer future range queries. In [16], range queries are processed in Chord by using an order-preserving hash function. Two approaches for supporting multidimensional range queries are presented in [5]. In the first approach, multi-dimensional data is mapped into a single dimension using space-filling curves and then this single-dimensional data is range-partitioned across a dynamic set of peers. For query routing, each multi-dimensional range query is first converted to a set of 1-d range queries. In the second approach, the multi-dimensional data space is broken up into "rectangles" with each peer managing one rectangle using a kd-tree whose leaves correspond to a rectangle being stored by a peer.

Routing indexes were introduced in [3] where various types of indexes were proposed based on the way each index takes into account the information about the number of hops required for locating a matching peer. In the attenuated Bloom filters of [14], for each link of a peer, there is an array of routing indexes. The i-th index summarizes items available at peers at a distance of exactly i hops. The peer indexes of [9] use the notion of horizon to bound the number of peers that each index summarizes.

6 Conclusions and Future Work

In this paper, we propose using histograms as routing indexes in peer-to-peer systems. We show how such indexes can be used to route queries towards the peers that have the most results. We also present a decentralized clustering procedure that clusters peers that match similar queries. To achieve this, we use the histograms of each peer and test how the L_1 and the edit histogram distances can be used to this end. Our experimental results show that our clustering procedure is effective, since in the constructed clustered peer-to-peer system, the network distance of two peers is proportional to the distance of their histograms. Furthermore, routing is very efficient, since using histograms increases the number of results returned for a given number of peers visited.

This work is a first step towards leveraging the power of histograms in peer-to-peer systems. There are many issues that need further investigation. We are currently working on defining more appropriate distance metrics and multi-attribute histograms. We are also developing procedures for dynamically updating the clusters. Another issue is investigating the use of other types of histograms (besides equi-width ones).

References

1. M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search Enhanced by Topic Segmentation. In *SIGIR*, 2003.
2. S-H Cha and S. N. Sribari. On Measuring the Distance Between Histograms. *Patern Recognition*, 35:1355–1370, 2002.
3. A. Crespo and H. Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *ICDCS*, 2002.
4. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, 2002. Submitted for publication.
5. P. Ganesan, B. Yang, and H. Garcia-Molina. One Torus to Rule Them All: Multidimensional Queries in P2P Systems. In *ICDE*, 2004.
6. R. Morris I. Stoica, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *IEEE/ACM Trans. on Networking*, 11(1):17–32, 2003.
7. Y. Ioannidis. The History of Histograms. In *VLDB*, 2003.
8. M.S. Khambatti, K.D. Ryu, and P. Dasgupta. Efficient Discovery of Implicitly Formed Peer-to-Peer Communities. *International Journal of Parallel and Distributed Systems and Networks*, 5(4):155–164, 2002.

9. Y. Wang. S. R. Jeffrey L. Galanis and D. J. DeWitt. Processing Queries in a Large Peer-to-Peer System. In *Caise*, 2003.
10. A. Loser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic Overlay Clusters within Super-Peer Networks. In *International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.
11. D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Technical Report HPL-2002-57, HP Laboratories Palo Alto, 2002.
12. Y. Petrakis and E. Pitoura. On Constructing Small Worlds in Unstructured Peer-to-Peer Systems. In *EDBT International Workshop on Peer-to-Peer Computing and Databases*, 2004.
13. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.
14. Sean C. Rhea and J. Kubiatowicz. Probabilistic Location and Routing. In *INFOCOM*, 2002.
15. O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A Peer-to-peer Framework for Caching Range Queries. In *ICDE*, 2004.
16. P. Triantafillou and T. Pitoura. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In *DBISP2P*, 2003.
17. P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. In *CIDR*, 2003.

Appendix

False Positive Probability for a Histogram. Let H be a histogram for an integer attribute $x \in [Dmin, Dmax]$ (x can take $D = Dmax - Dmin + 1$ distinct values). H has b buckets. Let a query $x = A$. We assume that each peer has n tuples that follow uniform distribution. Then in each bucket we have n/b tuples. The probability that we do not have a query match, that is, there does not exist a tuple with value $x = A$ in the data summarized by H is $P(query\_no\_match) = ((D-1)/D)^n$. The probability that the histogram indicates a match is: $P(hist\_match) = 1 - ((b-1)/b)^n$ (it is sufficient that one tuple falls into the bucket that A falls into as well). The range of each bucket is D/b. Thus the probability of having a *query_no_match* while we had a histogram match is: $P_1 = ((D/b-1)/(D/b))^{n/b} = ((D-b)/D)^{n/b}$. Thus, the false positive probability is according to the formula of Bayes:
$P(fp) = P(hist\_match \ / \ query\_no\_match) = P_1 * P(hist\_match)/P(query\_no\_match) \Rightarrow$

$$P(fp) = (((D-b)/D)^{n/b} * (1 - ((b-1)/b)^n))/((D-1)/D)^n. \quad (7)$$