Similarity Searching in Peer-to-Peer Databases

Indrajit Bhattacharya, Srinivas R. Kashyap, and Srinivasan Parthasarathy

{indrajit, raaghav, sri}@cs.umd.edu
Department of Computer Science, University of Maryland, College Park, MD 20742.

Abstract

We consider the problem of handling *similarity queries* in peer-to-peer databases. Given a query for a data object, we propose an indexing and searching mechanism which returns the set of objects in the database that are semantically related to the query. Our schemes can be implemented on a variety of structured overlays such as CAN, CHORD, Pastry, and Tapestry. We provide analytical and experimental evaluation of our schemes in terms of the search accuracy, search cost, and load balancing. Our analytical guarantees perfectly predict the experimentally observed trends for the search accuracy.

1 Introduction

Structured peer-to-peer systems such as CAN, CHORD, Pastry, and Tapestry [6, 9, 7, 11] have received a lot of attention lately. These systems implement a Distributed Hash Table (DHT) functionality on top of a self-organizing overlay of the network nodes. The main abstraction supported by these DHTs is the *lookup*. Given a query involving a particular *index*, the lookup solves the problem of finding the network node which owns the index.

Although all DHTs implement the basic lookup functionality efficiently, most real-life applications demand more. For instance, consider an Information Retrieval (IR) application where nodes export a collection of text documents. Each document is characterized by a d-dimensional vector. The field of IR is replete with vector-space methods for such document characterizations (for e.g, [2, 4]). A user query consists of a vector and the user needs all documents in the database which match this vector or which are *relevant* to it.

DHTs do not efficiently support applications like the one above. The fundamental reason which renders DHTs ineffective in these situations is that data objects in a DHT are distributed uniformly at random across the network nodes. While this ensures that no node stores too many objects, it also scatters *semantically related* objects across the network. Thus, when a query is issued, the only way the DHT can return all objects relevant to it would be to flood the entire network which leads to unacceptable network loads.

Our focus in this work is to efficiently support *similarity queries* in DHT based overlay networks. We introduce a query model where users issue queries of the form (x, δ). Here x is a data object and δ is a distance measure. The search algorithm needs to return all data objects y in the network such that $f(x, y) \leq \delta$, where f is an application specific *distance function*. The schemes presented in this paper are geared towards the Cosine distance metric which is defined as follows: $f(x, y) = \cos^{-1} \frac{x \cdot y}{|x||y|}$, where $x \cdot y$ is the dot product between the vectors and $|\cdot|$ is the Euclidean (l_2) norm. The Cosine distance is a widely used distance function in IR applications.

The key driver behind our techniques is the notion of *similarity preserving hash functions* (SPHs). SPHs provide a powerful and interesting property in the context of our work. Given a set of points which are at a small distance from each other, with high probability an SPH maps these points into a "small" set of related indices. Such a mapping of data objects onto indices leads to a simple search strategy as follows: a node u which has a query (x, δ), computes the set of indices which are relevant to object x; u then queries all the nodes which own these indices using the lookup primitive supported by the underlying DHT. The queried nodes return the set of relevant objects back to u.

Several research proposals exist for handling complex queries in peer-to-peer databases. Our work is in the same spirit as the pSearch system proposed by Tang *et al.* [10]. The pSearch system supports similarity queries based on the Cosine distance measure. It is built on top of CAN and uses LSI [4] to index documents. Object coordinates derived from these indices are used for routing and object location. However, to the best of our knowledge, the pSearch system is not extensible for overlay topologies other than CAN. Gupta *et al.* [5] and Schmidt *et al.* [8] use SPHs to distribute the objects on top of a CHORD overlay. The former supports approximate range queries in peer-to-peer databases and the latter supports exact range queries. However, neither of them support similarity queries. Also, to the best of our knowledge, their techniques do not extend beyond the CHORD overlay. Several other systems exist where the search algorithm is guided by summaries of neighbors' contents (Bloom filters), routing indices, or inverted indices.

Our work departs from these in significant ways. Specifically, we view the following as the main contributions of this work. We present a scheme for indexing vector data and a search algorithm for efficient similarity searches in peer-to-peer databases. Our techniques are specifically geared towards the Cosine distance measure which is of interest in IR applications. Our indexing scheme and search algorithm define a *broad framework which accomodates a variety of overlay topologies* and can be implemented on top of CAN, CHORD, Pastry, and Tapestry. We provide analytical guarantees which present the trade-off between the accuracy of the search results vs. the search cost (in terms of the number of nodes queried). *These guarantees are independent of the overlay topology as well as the data distribution.* The analytical guarantees are also in *perfect* agreement with the experimental results obtained through simulations.

2 Design Details

The two main components of our design are indexing and searching. Each data object in the peer-to-peer database is associated with an index. The set of indices are distributed across all the nodes and each index is owned by a unique node. We now propose a hash function h which takes a d-dimensional data object x as input and produces a k-bit string $h(x)$ as output. The string $h(x)$ is the index of object x.

2.1 The Indexing Scheme

Let r be a d-dimensional unit vector. Corresponding to this vector, we define the function b_r as follows:

$$b_r(x) = \begin{cases} 1 & \text{if } r \cdot x \geq 0 \\ 0 & \text{if } r \cdot x < 0 \end{cases}$$

$b_r(x)$ defines the orientation of x w.r.t. r. This function was proposed by Charikar [3] for estimating cosine distances between points in high dimensional space. He also observed that if r is chosen uniformly at random from all d-dimensional unit vectors, then for any two vectors x and y, $\Pr[b_r(x) \neq b_r(y)] = \delta/\pi$, where $\delta = cos^{-1} \frac{x \cdot y}{|x||y|}$ is the angle between the two vectors in radians.

Our hash function h is parametrized by a set of unit vectors $r_1, \ldots, r_k$, each of which is chosen uniformly at random from the set of all d-dimensional unit vectors. The hash value $h(x)$ is simply the concatenation of the bits $b_{r_1}(x), \ldots, b_{r_k}(x)$. We may construct multiple hash functions for placing multiple replicas of an object. Specifically, if each object has t replicas, then we construct hash functions $h_1, \ldots, h_t$ as described above. For any object x, these hash functions yield t different indices $h_1(x), \ldots, h_t(x)$. In general, t different nodes own these indices in the DHT and each of these t nodes acquire a replica when x is published.

2.2 The Search Algorithm

The search algorithm is parametrized by a radius r, which is a non-negative integer. A node u which generates a query (x, δ) first computes the index $h(x)$. It then computes the set S of all indices whose hamming distance from $h(x)$ is at most r (i.e., the set of indices which differ from $h(x)$ in at most r bit positions; note that S always includes $h(x)$). Let V be the set of nodes in the network which own the indices in S. Node u queries each of the nodes in V. Nodes in V return all data objects which match u's query.

How is the search radius r determined? The search radius r is affected by various parameters such as k, t, the query parameter δ, and the desired search accuracy. Fixing all other variables, an increase in the value of r would in general result in more objects which match the query being returned. Of course, this increased accuracy is also achieved at an increased search cost. We examine the effect of r on the search accuracy and cost in Section 4. The search algorithm may be easily extended when there are multiple replicas of an object.

Querying the set of nodes which own the relevant indices is achieved by performing a lookup operation for each index. DHTs typically implement the lookup primitive such that the routing cost for each lookup is logarithmic in the size of the network. Although this value is typically low, several optimizations of our basic search algorithm are possible which reduce the routing overhead further. We now discuss one such optimization for Pastry and Tapestry.

3 Discussion

3.1 Routing optimizations for Pastry and Tapestry

In an idealized scenario, Pastry and Tapestry can be viewed as implementations of a hypercube network. Each node in the overlay has a unique ID which is a k-bit binary string. The number of nodes in the system is exactly $n = 2^k$. Two nodes are overlay neighbors of each other if and only if the hamming distance of their IDs is one (i.e., they differ in exactly one bit). One may view the nodes as occupying the vertices of d-dimensional unit hypercube. The ID of the node is the k-bit string obtained from the k-dimensional $\{0,1\}$-coordinates of the hypercube vertex occupied by the node. An object is stored by a node if and only if the object index matches with the node ID.

In this scenario, a node u with a query (x, δ) performs a single lookup for x which terminates in a node v whose ID is $h(x)$. Node v performs a local search by flooding the query to all its r-hop overlay neighbors where r is the search radius. These nodes return their local search results to v which gathers and returns the union of all the results to u. Note that this optimization does not reduce the number of nodes being queried. However, it reduces the routing load substantially since each lookup is now replaced by a single overlay message.

3.2 Routing Optimization with Holes

The ideal setting described in the previous section may not hold in practice. The index size k is determined during network creation when the number of nodes is not known. Further, the number of nodes will vary dynamically in the system. We get around this by fixing k such that 2^k is an upper bound on the number of nodes in the network. However, this would result in some of the hypercube vertices being unoccupied leading to *holes*.

In our scheme, any hole u in the hypercube is adopted by a randomly selected node u_a in the network. This node will be responsible for hosting the objects assigned to this hole as well as performing the routing for it. Nodes which own the neighboring hypercube vertices of the hole u, update their routing tables to point to u_a. This ensures that if vertex u was within distance r of some other vertex v, then it remains within distance r from v after being adopted by u_a. Consequently, while the r-hop neighborhood of a node may change, the set of data objects in its r-hop neighborhood does not reduce. Our scheme thus provides the nice guarantee that the retrieval accuracy does not deteriorate due to holes.

4 Analysis

Consider a query (x, δ). Let S be the set of all points in the database which matches this query.

Let S' be the set of objects returned by the search algorithm. Recall that t is the number of replicas, k is the the number of bits in the index and r is the search radius. We define the accuracy of the search to be $|S'|/|S|$, i.e., the fraction of objects in the database which match the query and which are returned by the search. $\mathbf{E}[|S'|/|S|]$ denotes the expected accuracy. The following theorems hold.

Theorem 4.1

$$\mathbf{E}[|S'|/|S|] \geq 1 - \left(1 - \sum_{i=0}^{r} \begin{pmatrix} k \\ i \end{pmatrix} \left(\frac{\delta}{\pi}\right)^{i} \left(1 - \frac{\delta}{\pi}\right)^{k-i}\right)^{t} \tag{1}$$

Theorem 4.2 *Let the search cost (the number of nodes being queried) be C. Then,*

$$C = t \sum_{i=0}^{r} \begin{pmatrix} k \\ i \end{pmatrix} \tag{2}$$

We note that, in comparison with uniform hash functions, the use of SPHs generally results in an uneven distribution of data objects across indices and hence across nodes. Obtaining analytical load balancing guarantees without sacrificing accuracy is an involved issue and is a subject of future research.

5 Experimental Evaluation

We have implemented a prototype simulator for Pastry to evaluate our algorithms. We use the CHORD simulator which is publicly available [1]. The main goal of our simulations is to evaluate the search accuracy and the storage load as a function of various system parameters. Recall the definition of accuracy from Section 4. To evaluate storage load, we sort the nodes in decreasing order of number objects they store. The sorted list of nodes is bucketed such that each bucket contains 5% of the total number of nodes. For each of the 20 buckets, we plot the percentage of total number of objects that are stored in the nodes of the bucket. The baseline for comparsion is the perfectly uniform distribution where each bucket stores 5% of the objects.

5.1 Experimental Setup

The data samples for our experiments are drawn as follows: each coordinate of the data vector is sampled uniformly and independently at random from a standard normal distribution. For each query (x, δ), the parameter x is sampled from the same distribution as that of the data. We observe the effect of the number of replicas t, the size of the index k, the dimensionality of the data d, the number of nodes n, the number of data objects N, and the query parameter δ on the search accuracy and storage load. The default values of these parameters are in the table below. Results are averaged over 100 trials.

N	d	k	t	r	δ	n
50,000	15	10	1	1	0.75	2^k=1024

Table 1: Default values for network parameters used in the experiments

5.1.1 Observations

Figures (a)-(g) plot the effect of the various system parameters on the accuracy for the Pastry system. We plot both the experimentally observed values as well as the analytically predicted ones. The accuracy results from our CHORD simulations are identical to those of Pastry as expected. The accuracy increases as a function of the number of replicas t and the search radius r. It does does not vary much as a function of the data dimension d or the number of nodes n or the number of data objects N in the system. However, the accuracy decreases with the size of the index k as well as the the query parameter δ. Amazingly, our analysis predicts the experimental trends extremely accurately in all the trials. This suggests that the accuracy guarantees provided by our analysis do not only hold in expectation, but also with high probability. Also note that the experimentally observed values are always higher than the analytically predicted ones. This is explained by the fact that our analysis always yields a lower bound on the expected accuracy rather than the exact value.

Figures (h)-(l) plot the effect of the system parameters on the storage load across nodes. Figures (j) and (i) respectively indicate that increasing the size of the index k adversely affects the storage load balance while increasing the number of replicas t aids load balance. Varying other parameters does not seem to change the storage distribution across the nodes. The observed load balancing trends are similar for both CHORD and Pastry.

6 Conclusion and Future Work

We have presented a framework for indexing and searching data objects in peer-to-peer information retrieval systems. Our schemes use SPHs to map semantically related data objects to a small set of indices leading to a simple and efficient search algorithm. This framework can be implemented on a wide variety of structured overlays such as CAN, CHORD, Pastry and Tapestry.

We plan to extend our work in the future through extensive experimental evaluation. In particular, we plan to evaluate our schemes with data sets obtained from real applications and compare the performance of our scheme with existing systems such as pSearch. We also plan to evaluate the performance of our schemes under dynamic network conditions. Finally, load balancing mechanisms which evenly distribute the indices and query load across nodes without sacrificing accuracy will be a major focus of future studies.

References

[1] http://www.pdos.lcs.mit.edu/chord/.

[2] Michael W. Berry, Zlatko Drmac, and Elizabeth R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[3] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM Press, 2002.

[4] S.C. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, and R.A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[5] Abhishek Gupta, Divyakant Agrawal, and Amr El Abbadi. Approximate range selection queries in peer-to-peer systems. In *CIDR*, 2003.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *ACM SIGCOMM*, 2001.

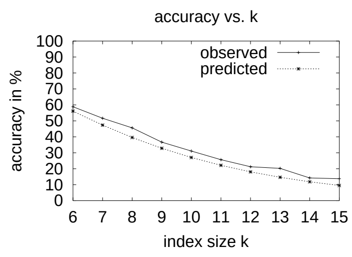
[7] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[8] Christina Schmidt and Manish Parashar. Flexible information discovery in decentralized distributed systems. In *IEEE International Symposium on High-Performance Distributed Computing (HPDC-12)*, 2003.

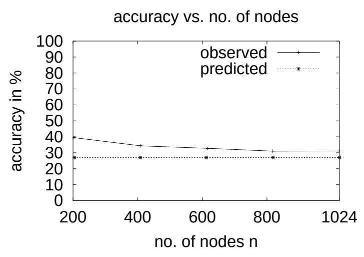
[9] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[10] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.

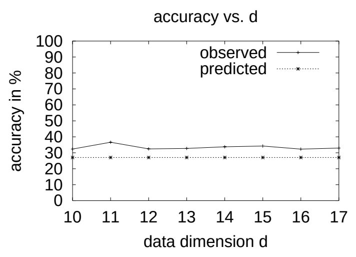
[11] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. To appear in IEEE Journal on Selected Areas in Communications.



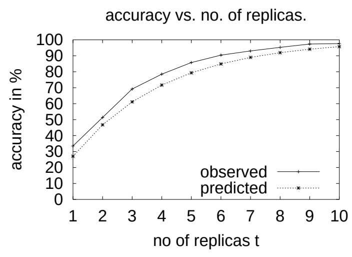
(a)



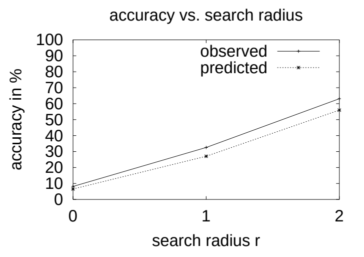
(b)



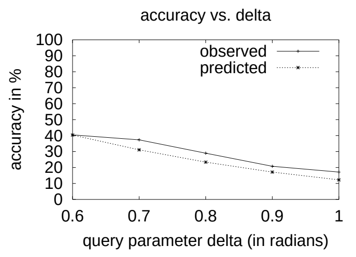
(c)



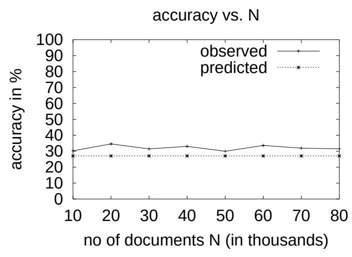
(d)



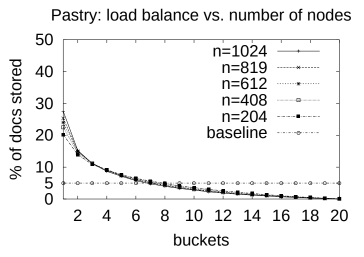
(e)



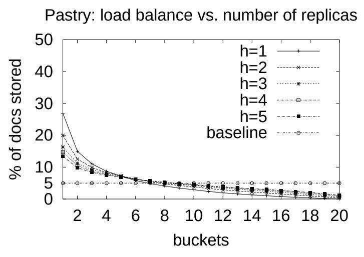
(f)



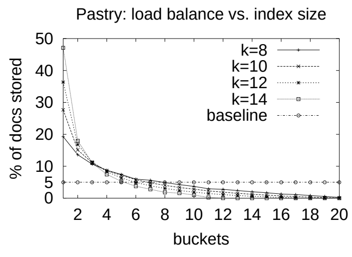
(g)



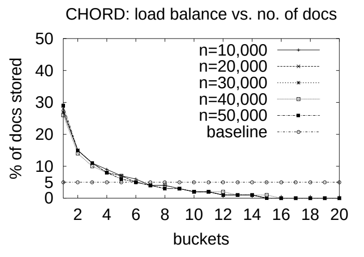
(h)



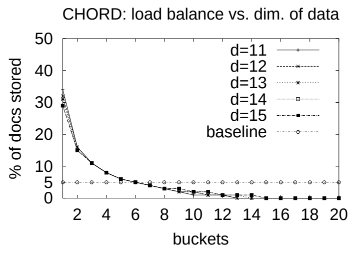
(i)



(j)



(k)



(l)

Figure 1: Experimental Results for Accuracy and Load Balance