Ranking Search Results in P2P Systems

Vijay Gopalakrishnan, Ruggero Morselli, Bobby Bhattacharjee, Peter Keleher, Aravind Srinivasan
Department of Computer Science
University of Maryland, College Park
{*gvijay, ruggero, bobby, keleher, srin*}*@cs.umd.edu.*

Abstract

P2P deployments are a natural infrastructure for building distributed search networks. Proposed systems support locating and retrieving all results, but lack the information necessary to rank them. Users, however, are primarily interested in the most relevant, and not all possible results.

Using random sampling, we extend a class of well-known information retrieval ranking algorithms such that they can be applied in this distributed setting. We analyze the overhead of our approach, and quantify exactly how our system scales with increasing number of documents, system size, document to node mapping (uniform versus non-uniform), and types of queries (rare versus popular terms). Our analysis and simulations show that a) these extensions are efficient, and can scale with little overhead to large systems, and b) the accuracy of the results obtained using distributed ranking is comparable to a centralized implementation.

1 Introduction

Peer-to-peer (P2P) systems are an increasingly popular design choice for wide-area distributed systems because of the their lack of central authority and resulting flexibility. Structured systems, such as those based on distributed hash tables (DHTs), are among the most popular and useful designs because of their support for efficient storage and lookup operations. However, DHTs are relatively inefficient at searching because the randomization induced by hash-based routing destroys object locality.

The current state-of-the-art in searching on such systems is to use inverted indexes. An inverted index lists all the objects that match some property, for example, documents that include the same keyword. Inverted indexes are kept like any other object in the P2P system. Efficient mechanisms [21, 28, 10] can use inverted indexes to provide boolean queries over keywords, but queries containing popular keywords return unmanageably many, unordered results. Ideally, query results would be *ranked*, and only relevant results returned to the user. Previous systems [29, 3] for distributed similarity-based searching have been developed; while these systems can be used to locate semantically similar documents, they cannot be used to rank order results.

The advantages of ranking search results are well known: users are primarily interested only in the most relevant results rather than an unordered set of all possible results, e.g., consider the improvement in quality of search results using a ranking scheme such as PageRank [20] used by Google versus manually sorting through millions of documents that match a set of popular terms. Even seemingly specific queries can return a very large number of mostly useless results, e.g., in October 2005, the query "NSDI 2005" matched over 140,000 web pages indexed by Google. Second, collecting fewer results reduces the network bandwidth consumed, helping the system scale up—to many users, hosts, and data items— and down—to include low-bandwidth links and low-power devices.

Ranking results in a distributed manner is difficult because ranking is global: *all* documents (matching a query) have to be ranked with respect to each other. In a completely distributed system, the results returned for identical queries should ideally be the same (this is not an issue in a centralized implementation). In a large system, the lack of a central location to aggregate global knowledge makes the problem of ranking search results challenging.

In this paper, we describe a method of efficiently and consistently ranking search results in a completely distributed manner, using an approximation technique based on uniform random sampling. Our technique can be used to compute a class of ranks based on the classic Vector Space Model (VSM) [25] originally due to Gerard Salton et al. Neither VSM nor uniform sampling are new ideas; instead, the most novel contribution of our work is to demonstrate that these two techniques compose well, i.e. sampling enables distributed ranking with very little network overhead. Further, our results apply to both structured and unstructured networks.

It is possibly not too surprising that VSM can be approximated if sufficient nodes were sampled even in a large network — in the limit, if every node were sampled, the correct rank would be computed, albeit at incredible cost. However, our analysis shows that, under reasonable conditions, the cost of our sampling-based algorithm is small and *remains constant as the size of the system increases*. More precisely, if the number of documents in the system is at least as large as the number of nodes and if the documents are reasonably well distributed (i.e. the maximum number of documents at any node is at most a constant times the average), then our ranking system needs only to sample a constant number of nodes for every document insertion or search query[1]. Additionally, our distributed algorithm maintains the same level of performance when the number of documents is smaller than the number of nodes, as long as each document is replicated at a sufficient number of nodes. We present a set of simulation results that confirm our analysis. Further, the results show that the constants in the protocol are low, e.g. the protocol performs very well with samples from 20 nodes per query on a 5000 node network.

The main contributions of this paper are as follows:

- **Algorithm:** We present a set of techniques for efficiently approximating centralized VSM document ranking in a distributed manner. We present detailed algorithms for computing the SMART [4] implementation of VSM[2]; however, our scheme is general, and can easily be modified to compute other VSM-based global ranks. We analyze the overhead of our approach, and quantify exactly how our system scales with increasing number of documents, system size, non-uniform document to node mapping (as may be the case in unstructured networks) and types of queries (rare versus popular terms).

- **Evaluation:** We evaluate our ranking technique in a simulator using real document sets from the TREC collection [1]. The results show that our approach is robust to sampling errors, initial document distribution, and query location. Our results show that the distributed ranking scheme, with relatively little overhead (50 samples per query), can approximate centralized ranking ($> 85\%$ accuracy) in large systems (5000 peers) with many documents (100,000 documents, 400,000 unique terms).

The rest of the paper is organized as follows. We first present some background on ranking in classical information retrieval in Section 2. We then discuss our design for ranking results in Section 3 and analyze its properties. In Section 4, we present experimental results where we compare the performance of the distributed ranking scheme with a centralized scheme. We discuss other related work in Section 5 before concluding in Section 6.

[1] Note that this condition on document distribution will occur naturally in a structured system such as a DHT where documents are mapped uniformly at random to nodes.

[2] SMART is probably the most popular implementation of this type of global ranking, and is regularly used to rank queries on IR collections [12].

Doc 1	"He checked the **time** on his **watch**."
Doc 2	"No **time**, no **time**, said the Mad Hatter while dipping his **watch** in his tea."
Doc 3	"**Time** flies like an arrow."
Doc 4	"Did you buy a new **watch**?"
Query	***"time, watch"***

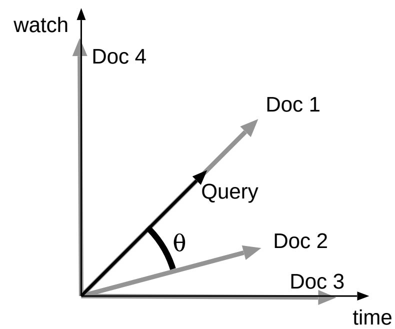

Figure 1: How VSM works on a collection of four documents. For simplicity, in computing the vectors we only consider the two terms that appear in the query. The figure shows (only qualitatively) the five vectors and the angle θ, the cosine of which is the similarity measure between the second document and the query.

2 Vector Space Model (VSM)

The Vector Space Model (VSM) is a classic information retrieval model due to Gerard Salton et al. [25]. VSM maps documents and queries to vectors in a T-dimensional term space, where T is the number of *unique* terms in the document collection. Each term i in the document d is assigned a weight $w_{i,d}$. The vector for a document d is defined as $\vec{d} = (w_{1,d}, w_{2,d}, \ldots, w_{T,d})$. A query is also represented as a vector $\vec{q} = (w_{1,q}, w_{2,q}, \ldots, w_{T,q})$, where q is treated as a document.

Vectors that are similar have a small angle between them. VSM uses this intuition to compute the set of relevant documents for a given query by looking at the angle between the vectors; relevant documents will differ from the query vector by a small angle while irrelevant documents will differ by a large angle.

Consider the example in Figure 1 which shows the vector representation of documents and query in a two-dimensional space. For simplicity, in this example we only consider the terms `time` and `watch`. In a real collection of documents, the number of unique terms would be much higher. `Doc 1`, `Doc 2` and the query contain both terms, therefore their vectors have positive components with respect to both axes. `Doc 3` only contains the term `time`; therefore its vector lies on the horizontal axis. Similarly, `Doc 4` lies on the `watch` axis. Since the word `time` appears twice in the second document, while `watch` appears only once, the vector for `Doc 2` is closer to the `time` axis. From Figure 1, it is clear the document most relevant to `Query` is `Doc 1`, followed by `Doc 2`.

Given two vectors X and Y, the angle θ between them can be computed using:

$$\cos\theta = \frac{\sum_{i=n}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}. \tag{1}$$

Equation 1, also known as the cosine similarity, has been used in traditional information retrieval to identify and rank relevant results. We will also use this measure of similarity for our ranking.

Cosine similarity has previously been used for similarity search in P2P systems by Tang et.al. [29] and Bhattacharya et.al. [3]. Given a query vector, these systems use the cosine similarity measure to identify other similar documents. In theory, [29] and [3] can also use the cosine similarity to rank results; however,

Local Weight	$\chi\left(f_{t,d}\right)$	Salton et al.[23]
	$f_{t,d}$	Salton et al.[23]
	$\ln\left(f_{t,d}+1\right)$	Frakes et al.[8]
	$\ln f_{t,d}+1$	Dumais[7]
Global Weight	$\ln\left(\frac{\sum_{\forall d} f_{t,d}}{D_t}\right)$	Dumais[7]
	$\ln\left(\frac{D-D_t}{D_t}\right)$	Frakes et al.[8]
	$\ln\left(\frac{D}{D_t}\right)$	Salton et al.[23]
	$1+\sum_{j=1}^{D}\left(\frac{p_{t,j}\log p_{t,j}}{logD}\right)$ where, $p_{t,j}=f_{t,j}/\sum_{k=1}^{D} f_{t,k}$	Dumais[7]

Table 1: Formulae to compute local and global components of term weights. $f_{t,d}$ is the frequency of term t in document d. D is the total number of documents in the system. D_t is the number of documents with term t.

their default arrangement of data would make such ranking extremely inefficient. We compare our system to these in detail in Section 5.

2.1 Generating Vector Representation

The vector representation of a document is generated by computing the *weight* of each term in the document. The goal behind assigning weights is to identify terms that capture the semantics in the document and therefore help in discriminating between the documents. Effective term weighting techniques have been an area of much research [7, 23, 8], unfortunately with little consensus. However, most methods use three components in their weighting schemes:

1. The *term frequency* factor is a local weight component and is based on the observation that terms that occur frequently in a document are keywords that represent the document. Terms that occur frequently are assigned higher weight than less frequently occurring terms. Weighting schemes differ in how much importance they give to the actual frequency.

2. The *Inverse Document Frequency (IDF)* factor, which is a global factor, takes into account the importance of a term occurring infrequently in the document collection. In other words, it looks at how many other documents contain this word. In practice, words that occur in many documents are not useful for distinguishing between documents. For example, words like “the”, “because” etc. would get assigned high term weights because they occur frequently in documents. Further, these words occur in almost all documents. Hence, these words are not useful in distinguishing documents.

3. The third component is a normalization factor. Terms that occur in longer documents get a higher weight, causing vectors of longer documents to have higher norms than shorter documents. Although cosine similarity is insensitive to the vector norms, other similarity measures are not. Normalizing the weights eliminates this advantage.

Table 1 lists a variety of different formulae proposed in the literature to compute local and global components. In this paper, we use the weighting formula used in the SMART [4] system:

$$w_{t,d}=\left(\ln f_{t,d}+1\right)\cdot\ln\left(\frac{D}{D_t}\right). \tag{2}$$

```
EXPORT-DOCUMENT(n, d)
    for each t in d
        w_t ← COMPUTE-LOCAL-WEIGHT(d, t)
        w_t ← w_t · ESTIMATE-GLOBAL-WEIGHT(t, k)
    NORMALIZE-VECTOR(w⃗)
    for each t in d
        key ← generate-key(t)
        n′ ← lookup(key)
        insert(n′, key, d.id, w⃗)
    key ← generate-key(d.id)
    n′ ← lookup(key)
    insert(n′, key, d.id, d)

EVALUATE-QUERY(n, q, size)
    for each t in q
        w_t ← COMPUTE-LOCAL-WEIGHT(q, t)
        w_t ← w_t · ESTIMATE-GLOBAL-WEIGHT(t, k)
    NORMALIZE(w⃗)
    for each t in q
        key ← generate-key(t)
        n′ ← lookup(key)
        R ← R ∪ GET-RESULTS(n′, key, w⃗, size)
        R ← TOP-K(R, size)
    return R

COMPUTE-LOCAL-WEIGHT(d, t)
    if f_{t,d} = 0
        then return 0
        else return ln(f_{t,d}) + 1

ESTIMATE-GLOBAL-WEIGHT(t, k)
    d ← 0
    d_t ← 0
    for i ← 1 to k
        j ← GET-RANDOM-NODE()
        d ← d + GET-TOTAL-DOCS(j)
        d_t ← d_t + GET-DOC-COUNT(j, t)
    return ln(d / d_t)

NORMALIZE-VECTOR(w⃗)
    for i ← 1 to T
        w_i = w_i / sqrt(Σ_{i=1}^{T} w_i × w_i)

GET-RESULTS(n, key, q⃗, size)
    for each d⃗ in Index_key
        w_d ← Σ_{∀t} d_t q_t
        R ← R ∪ (d, w_d)
    R ← TOP-K(R, size)
    return R
```

Figure 2: Pseudo-code of our ranking algorithm.

In this equation, $w_{t,d}$ is the weight of term t in document d, $f_{t,d}$ is the raw frequency of term t in document d, D is the total number of documents in the collection, and D_t is the number of documents in the collection that contain term t.

3 Distributed VSM Ranking

We assume a cooperative system where peers arrive and depart periodically, and participate in an underlying lookup protocol (either a structured protocol, like Chord [26], Pastry [22], etc. or an unstructured protocol like LMS [19] or Yappers [9]). As is the norm, each peer may *export* a set of documents; a set of keywords (by default, all words in the document) are associated with the document.

We design a distributed VSM ranking system for keyword-based queries in this setting. Users submit queries containing keywords and may specify that only the highest ranked K results be returned. Beyond the usual document export and lookup, there are three main components needed for ranking: generating a vector representation for exported documents, storing the document vectors appropriately, and computing and ranking the query results. We describe each in turn.

3.1 Generating Document Vectors

Recall Equation (2), which is used to compute the weight of each term t in a document. The equation has two components: a local component, $\ln f_{t,d} + 1$, which captures the relative importance of the term in the given document, and a global component, $\ln(D/D_t)$, which accounts for how infrequently the term is used across all documents. The local component, which is the frequency of the term in the document, is obtained locally as in procedure COMPUTE-LOCAL-WEIGHT of Figure 2.

The global component is stated in terms of the number of documents D in the system, and the number of documents D_t that have the term t. We use random sampling to estimate these measures. Other approaches are also possible; we discuss those in Section 5. In what follows, we describe our approach and analyze its properties. The pseudo-code for this step is in the procedure ESTIMATE-GLOBAL-WEIGHT in Figure 2.

Let N be the number of nodes in the system, and D and D_t be as above. Initially, we assume that a document is stored at exactly one node in the system. We will remove this assumption later.

We choose k nodes uniformly at random and independently. We then compute the total number $\tilde{D}$ of documents and $\tilde{D}_t$ of documents with term t at the sampled nodes. Choosing a random node can be done either with random walks, in unstructured systems, or routing to a random point in the namespace, in structured systems. For simplicity, we accept that the same node may be sampled more than once. It is easy to see that:

$$\mathrm{E}[\tilde{D}] = k\frac{D}{N}$$

and

$$\mathrm{E}[\tilde{D}_t] = k\frac{D_t}{N},$$

where E indicates expectation of a random variable. The intuition is that, if we take enough samples, $\tilde{D}$ and $\tilde{D}_t$ are reasonably close to their expected value. If that is the case, then we can estimate D/D_t as:

$$\frac{D}{D_t} \approx \frac{\tilde{D}}{\tilde{D}_t}.$$

We now derive a sufficient condition for this approximation to hold. We introduce two new quantities: let M be the maximum number of documents at any node and M_t the maximum number of documents at any node with term t. We call the estimate $\tilde{D}$ (resp. $\tilde{D}_t$) "good", if it is within a factor of $(1 \pm \delta)$ of its expected value and we allow for the estimate to be "bad" with a small probability (ϵ).

Theorem 1. *Let D, N, k, M be as above. For any $0 < \delta \leq 1$ and $\epsilon > 0$, if*

$$k \geq \frac{3}{\delta^2}\frac{M}{D/N}\ln(2/\epsilon). \tag{3}$$

then the random variable $\tilde{D}$ (as defined above) is very close to its mean, except with probability at most ϵ. Specifically:

$$\Pr[(1-\delta)\frac{kD}{N} \leq \tilde{D} \leq (1+\delta)\frac{kD}{N}] > 1-\epsilon. \tag{4}$$

(Proof in Appendix.)

Note that if we replace D, M, $\tilde{D}$ with D_t, M_t, $\tilde{D}_t$, the theorem also yields that, if:

$$k \geq \frac{3}{\delta^2}\frac{M_t}{D_t/N}\ln(2/\epsilon). \tag{5}$$

then the random variable $\tilde{D}_t$ is also a good estimate.

The following observations follow from Theorem 1:

- Theorem 1 tells us how many samples we need for $\tilde{D}/\tilde{D}_t$ to be a good estimate of the ratio D/D_t. Analogous to the classical problem of sampling a population, the number of samples needed does not depend on N directly, but only through the quantities D/N and D_t/N and, less importantly, on M and M_t.

  This means that as the system size grows, we do *not* need more samples as long as the number of exported documents (with term t) also increases. More samples are needed only if the system size grows without a corresponding increase in the number of documents. However, replication can be used to handle this case: our algorithm works without modification in the case where the same document may be stored at multiple (r) nodes, as long as r is the same for all of the documents. The analysis above still holds, as long as D and D_t are replaced by rD and rD_t respectively. Note that sampling performance actually *improves* with replication. Extending the algorithm to general replication is an open problem.

- The global component $\ln(D/D_t)$ that we are trying to estimate is relatively insensitive to estimation errors on D and D_t, because the algorithm uses the logarithm of the ratio.

- If we restrict our attention to the case where the number of documents D is much larger than the system size N and we focus on documents and queries consisting of popular terms ($D_t = \Omega(N)$), then our algorithm provides performance with ideal scaling behavior. Sampling a constant number of nodes gives us provably accurate results, *regardless of the system size*.

- In practice, documents and queries will contain rare (i.e., not popular) terms, for which $\ln(D/D_t)$ may be estimated incorrectly. However, we argue that such estimation error is both unimportant and inevitable. The estimation is relatively unimportant because if the query contains rare terms, then the entire set of results is relatively small, and ranking a small set is not as important. Finally, note that in centralized systems, ranking algorithms do not consider such rare terms (e.g. terms that appear only in one document) in ranking documents since they dominate the document weight. However, in a distributed setting, it is not possible to discern whether a term is truly rare since this requires global knowledge. In general, sampling is a poor approach for estimating rare properties; we describe other possible rank computation approaches that handle rare terms better in Section 3.4.

- The number of samples is proportional to the ratios $\frac{M}{D/N}$ and $\frac{M_t}{D_t/N}$ between the maximum and the average number of documents stored at a node (respectively, of all documents and of the documents with term t). This means that, as the distribution of documents in the system becomes more imbalanced, more samples are needed to obtain accurate results.

3.1.1 Special case: uniform distribution

We next restrict our attention to the special case in which the underlying storage system randomly distributes documents to the nodes, uniformly and independently. Such distribution approximately models the behavior of a DHT.[3] In this special case, a stronger version of Theorem 1 holds.

Theorem 2. *Let D, N, k be as above and assume each document is independently and randomly stored at one node. For any $\epsilon > 0$, if*

$$k \geq \frac{3}{\delta^2} \frac{1}{D/N} \ln(2/\epsilon). \tag{6}$$

[3]More precisely, in a DHT like Chord [26] or Pastry [22], documents are not exactly uniformly distributed to nodes, but the probability that a document is assigned to a specific node may be as high as $\Theta((\log N)/N)$ and as low as $\Theta(1/N^2)$. For simplicity, we ignore this detail.

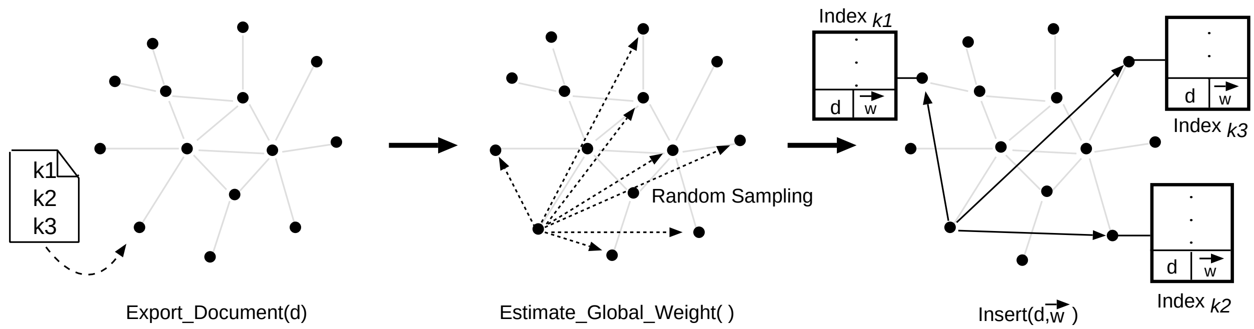

Figure 3: Various steps in exporting documents and their vector representation

then the random variable $\tilde{D}$ (as defined above) is very close to its mean, except with probability at most ϵ. Specifically:

$$\Pr[(1-\delta)\frac{kD}{N} \leq \tilde{D} \leq (1+\delta)\frac{kD}{N}] > 1-\epsilon. \tag{7}$$

Hence, for the uniform case, the number of samples does not need to be proportional to the maximum number M of documents at any node. Therefore, the cost of our sampling algorithm is significantly decreased.

3.2 Storing Document Vectors

Once the document vectors are computed, they need to be stored such that a query relevant to the document can quickly locate them. We store document vectors in distributed inverted indexes. This choice allows us to efficiently retrieve the vectors of all documents that share at least one term with the query.

Figure 3 shows the process of exporting a document. We first generate the corresponding vector by computing the term weights, which requires the previously described random sampling. Next, using the API of the underlying storage system, we identify the node storing the index associated with each term in the document and add an entry to the index. Such entry includes a pointer to the document and the document vector.

The details of storing document vectors in inverted indexes depend on the underlying lookup protocol. In structured systems, given a keyword t, the index associated with t is stored at the node responsible for the key corresponding to t. The underlying protocol allows to efficiently locate such a node. Inverted indexes have previously been used for searching [28, 11, 21, 10] in structured systems. We present the pseudo-code for exporting the document and storing the document vector in our system, over a structured network, in the procedure EXPORT-DOCUMENT in Figure 2.

Storing vectors in unstructured systems In structured systems, inverted indexes enable us to group all documents related by a term at one location. Such a scheme is not directly applicable in unstructured networks. Instead we have to resort to approaches such as Yappers [9] or LMS [19]. As with structured systems, items are mapped to keys. However, due to the lack of structure in these systems, keys are not mapped to unique nodes in the network. Instead, depending on the facilities provided by the underlying network, each index can either be partitioned or replicated and stored at multiple nodes. We can use this facility to store document vectors in keywords indexes. The indexes can subsequently be located (and if necessary, reconstituted) using the underlying lookup mechanism.

Our approach for storing document vectors is susceptible to the usual problems found in systems using distributed inverted indexes. For example, the indexes for popular terms can be large; the system is suscepti-

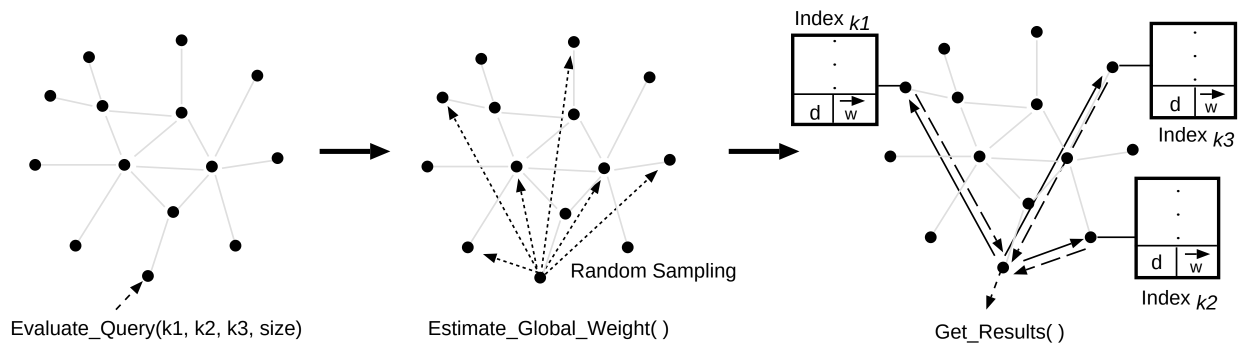

Figure 4: Process of computing query results

ble to load imbalances, both in terms of query rates and the amount of data stored on nodes; node departures and failures can disrupt query evaluation, and so on. However, solutions proposed in prior work for each of these problems [28, 11] directly apply in our setting, and we do not consider them in this paper.

3.3 Evaluating Query Results

In order to evaluate queries, they first need to be converted into their corresponding vector representations. We then compute the cosine similarity of each query with each "relevant" document vector. Figure 4 shows these different steps involved in the query evaluation.

We compute query vectors using the same technique. We first compute the local weight of the query terms, estimate the global weight by sampling, and then normalize the query vector. The next step is to locate the set of relevant documents. For each keyword in the query, we use the lookup functionality provided by the underlying system to identify the node storing the index. We then compute the cosine similarity between the query and each of the document vectors in the index. Finally, we fetch the results of the evaluation at each of the indexes and compute the union of these results. The top-K documents in this set, sorted according to the decreasing order of cosine similarities, give us our final result set $\mathcal{R}$. We outline this entire process in the procedure EVALUATE-QUERY in Figure 2.

3.4 Extensions

Reducing storage cost Our approach to storing document vectors, as discussed thus far, assumes that each unique word in the document should be treated as a keyword, and be part of the document's descriptive vector. Hence a document entry is added to the indexes of all the unique keywords in the document. This is expensive both in the size of the vectors, and in the cost of propagating the vector to all of the corresponding indexes.

We employ a heuristic to reduce the number of vectors that are stored in the system. We assume that there is a constant threshold w_{min}, that determines if the document entry is added to an index. The vector is not added to the index corresponding to the term t if the weight of t is below the threshold w_{min}. Note that the terms with weights below this threshold are still added to the vector. The intuition behind this approach is the following: a document will not appear in the top results of a query that has in common with the document only terms with low weight. Hence, discarding such entries in the index does not reduce the retrieval quality of the top results. This heuristic has previously been successfully used in eSearch [28].

Reducing sampling cost Using random sampling to estimate the global component of the term weights is generic and works well in any setting: structured or otherwise. However, the cost involved in sampling k

Parameter	Values
# of runs for each experiment	50
# of Nodes	1000, 5000
# of Documents	100K, 1033
# of unique terms	418K
# of random samples	10, 20, 50
Mapping of doc. to nodes	Uniform, Zipf
Query term popularity	Q_{pop} - terms in $> 10K$ doc. Q_{5K} - terms in $\sim 5K$ doc. Q_{rare} - terms in < 200 doc.

Table 2: Different parameters in the experiments and their values

nodes for each term is non-negligible. While we cannot eliminate the need for this estimate, we can use the information in the distributed indexes to reduce its cost.

Each inverted index contains a list of all documents that contain the keyword, and is hence an authoritative source for D_t. This eliminates the need to sample for D_t, thereby reducing the number of messages by a factor of k. We are, however, still left with estimating D, the total number of documents in the system. We discuss two approaches to estimating D. In the first, using gossip, nodes would periodically exchange estimates of system size.

The second approach is to continue using random sampling. Recall that $E[\tilde{D}]$ is dependent on N. Hence we need to estimate N in order to estimate D. In a recent result, King et al. [15] present an approach to estimate the number of nodes in a Chord-like system. Using this estimate on the number of nodes in the system, we can utilize random sampling to periodically estimate D. Techniques to estimate the number of nodes and other aggregates also exist for unstructured systems [2].

Both these approaches for reducing cost of sampling are promising, and we plan to experiment with these techniques as part of our future work.

4 Evaluation

In this section, we validate our distributed ranking system via simulation. We measure performance by comparing the quality of the query results returned by our algorithm with those of a centralized implementation of VSM. With these experiments, we demonstrate that our distributed system produces high quality results with little communication cost (of the order of 10 nodes visited per document insertion or query), for a reasonably large system (1000 nodes with 100K documents), and that such cost *is actually constant*, even as the system size increases. We also include results that show how to reduce the storage required by the search protocol without impacting quality or increasing communication cost, and results that illustrate consistent ranks obtained by different users without prior coordination.

Experimental setup We use the TREC [1] Web-10G data-set for our documents. We used a subset of 100,000 documents from this dataset for our experiments. These 100K documents contain approximately 418K unique terms. Our default system size consists of 1000 nodes. We use two different distributions of documents over nodes: a uniform distribution to model the distribution of documents over a structured P2P system and a Zipf distribution to model a skewed distribution. Such a skewed distribution is possible in unstructured systems such as Yappers.

Since our large data set (100K documents) did not have queries associated with it, we generated queries of different lengths. Our default query set consists exclusively of terms that occur in approximately 5000 documents. We denote this query set as the Q_{5K} query set in our experiments. The intuition behind picking these query terms is that they occur in a reasonable number of documents, and are hence popular. At the same time, they are useful enough to discriminate documents. We also use query sets that exclusively contain keywords that are either very popular (occur in more than 10K documents) or those that are very rare (occur in less than 200 documents). We denote these query sets as Q_{pop} and Q_{rare} respectively. In order to verify the performance against real queries, we use the smaller Medlars [4] medical dataset. This data set consists of 1033 documents and also has queries associated with it, which we use to evaluate our scheme with real queries. Each result presented (except for details from individual runs) is an average of 50 runs with different random seeds. We summarize the various parameters used in our experiments in Table 2.

We use three metrics to evaluate the quality of distributed ranking:

Coverage We define coverage as the number of top-K query results returned by the distributed scheme that are also present in the top-K results returned by a centralized VSM implementation for the same query. For example, if we're interested in the top 3 results, and the distributed scheme returns the documents (A, C, D) while the centralized scheme returns (A, B, C), then the coverage for this query is 2.

Fetch We define fetch as the minimum number K' such that, when the user obtains the set $\mathcal{R}'$ of top-K' results as ranked by the distributed scheme, $\mathcal{R}'$ contains all the top-K results that a centralized implementation would return for the same query. In the previous example, if the fourth result in the distributed case had been C, then the fetch for $K = 3$ would be 4.

Consistency We define consistency as the similarity in the rank of results, for the same query, for different runs using different samples.

We do not explicitly present network overhead measures since the cost of the ranking (without counting the cost to access the indexes) is always equal to the number of nodes sampled. For the space optimization experiments, we note the original and reduced size of the indexes.

4.1 Coverage

Network setup	Number of samples	Top-K results														
		top-10 results			top-20 results			top-30 results			top-40 results			top-50 results		
		Avg	SD	Med	Avg	SD	Med	Avg	SD	Med	Avg	SD	Med	Avg	SD	Med
1000 uniform	10	8.49	1.08	8.64	16.99	1.20	17.04	25.30	1.55	25.22	33.68	2.07	33.44	42.28	2.01	42.3
	20	8.90	0.99	9.04	17.81	1.04	17.72	26.44	1.26	26.48	35.23	1.87	35.16	44.30	1.82	44.24
	50	9.28	0.82	9.42	18.63	0.82	18.62	27.66	1.04	27.68	36.08	1.45	37.24	46.30	1.46	46.44
5000 uniform	10	6.78	1.39	6.66	13.58	1.74	13.64	20.43	2.39	20.48	27.35	2.99	27.18	34.59	3.40	34.08
	20	7.74	1.29	7.88	15.41	1.46	15.46	22.92	1.96	22.88	30.50	2.47	30.18	38.49	2.58	38.22
	50	8.52	1.09	8.62	16.96	1.18	16.88	25.20	1.56	25.06	33.59	2.11	33.36	42.34	1.98	42.40
1000 Zipf	10	8.27	1.15	8.28	16.52	1.26	16.62	24.66	1.71	24.44	32.82	2.21	32.52	41.20	2.27	41.18
	20	8.82	0.99	8.86	17.63	1.06	17.60	26.22	1.35	26.24	34.83	1.93	34.58	43.70	1.88	43.52
	50	9.26	0.80	9.38	18.54	0.88	18.54	27.52	1.12	27.72	36.71	1.49	37.08	46.12	1.56	46.22
5000 Zipf	10	6.09	1.54	6.04	12.29	1.97	12.36	18.58	2.68	18.48	25.01	3.39	25.16	31.67	3.97	31.44
	20	7.34	1.31	7.38	14.71	1.62	14.68	21.89	2.10	21.74	29.34	2.64	29.12	36.93	2.90	36.60
	50	8.41	1.13	8.48	16.73	1.22	16.78	24.92	1.61	24.72	33.22	2.08	32.68	41.71	2.03	41.54
Medlar	5	8.08	1.26	8.4	16.64	1.90	16.90	25.22	2.47	25.68	33.78	2.85	34.18	42.36	3.03	42.74

Table 3: Coverage of the distributed ranking scheme.

In the first experiment, we measure the coverage of the distributed retrieval scheme. We show that by sampling only a few nodes even on a reasonably large system, our scheme produces results very close to a centralized implementation

In our base result, we use a 1000 node network. The documents are mapped uniformly to nodes. To compute the global weight of term t, we sample 10, 20 and 50 nodes in different runs of the experiment. The queries consist of keywords from the Q_{5K} query set, i.e. the keywords occur in approximately 5000 documents.

Table 3 shows the results of the experiment in which documents were mapped uniformly at random to nodes. As is clear from the table, distributed ranking scheme performs very similar to the centralized implementation. On a 1000 node network with documents distributed uniformly, the median accuracy for the top-10 results is 95% with 50 random samples, while the median is 93% for the top-50 results. As we increase the number of results we look at, the weights assigned to the documents decrease and small errors in estimation can change the order in which the documents are retrieved. Even with 10 random samples, the results are only slightly worse: 86% accurate for top-10 results and 85% for top-50 results.

With 5000 nodes, the retrieval quality is not as high as a network with 1000 nodes. With 20 random samples, the median accuracy is 79% for top-10 results, and is a little over 76% for the top-50 results. There is a 10% increase in accuracy when we increase the sampling level and visit 1% of the nodes. This result is a direct consequence of Theorem 2. Here, the number of documents has remained the same, but the number of nodes has increased. Hence, higher number of nodes sampled leads to better estimates.

Table 3 also shows the retrieval quality for documents mapped to nodes using a Zipf distribution with parameter 0.80. With 1000 nodes and 50 samples, the retrieval quality is nearly equal to that of the uniformly distributed case. With 10 samples, however, the median accuracy drops a few percentage points to between 82-83%. With 5000 nodes and 50 samples, we see similar trends. While the quality is not as good as it is with the uniformly distributed data, it does not differ by more than 2%. With 10 samples, the results worsen by about much as 5%. Hence, we believe our scheme can directly be applied over lookup protocols on unstructured networks without appreciable loss in quality.

Finally, Table 3 also shows coverage of the distributed scheme with the Medlar dataset. Since the dataset has only 1033 documents, we use a system size of 100 nodes and sample 5 nodes. As shown in the table, the median coverage is around 85%. Distributed ranking performed well with the traditional metrics of recall and precision also. We do not present those results in this paper.

4.2 Fetch

Given the previous result, an obvious question to ask is how many results need to be fetched before all the top-K results from the centralized implementation are available (we called this measure Fetch). We ran an experiment with both 1000 and 5000 nodes with the documents uniformly distributed. We used the Q_{5K} query set for our evaluation. We plot the result in Figures 5 and 6. The x-axis is the top-K of results from the centralized implementation, while the y-axis represents the corresponding average fetch.

With a 1000 node network, we see that the fetch is quite small even if only ten nodes are sampled. For instance, sampling 10 nodes, we have to get a maximum of 13 results to match the top-10 results of the centralized case. With samples from 50 nodes, fetch is minimal even for less relevant documents: we only have to get 11 results to match the desired top-10 results and 63 to match the top-50 results from the centralized implementation.

As expected, with increasing network size, but for the same document set, the fetch increases. When we sample 1% of the 5000 nodes, we need 13 results to cover the top-10 and 88 to cover the top-50. With lower levels of sampling, however, we need to fetch a lot more results to cover the top-K. This behavior, again, is predicted by Theorem 2: when the number of nodes increases without a corresponding increase in the number of documents, the number of samples needed to guarantee a bound on the sampling error also increases.

Other experiments indicate similar results when the document distribution is skewed. We merely summarize those results here. With a 1000 node network and 10 random samples, the fetch increases by 10%

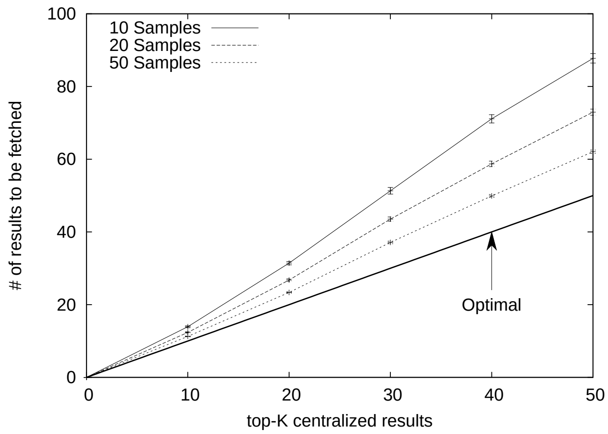

Figure 5: Average fetch of the distributed ranking scheme with 1000 nodes. The error bars correspond to 95% confidence interval.

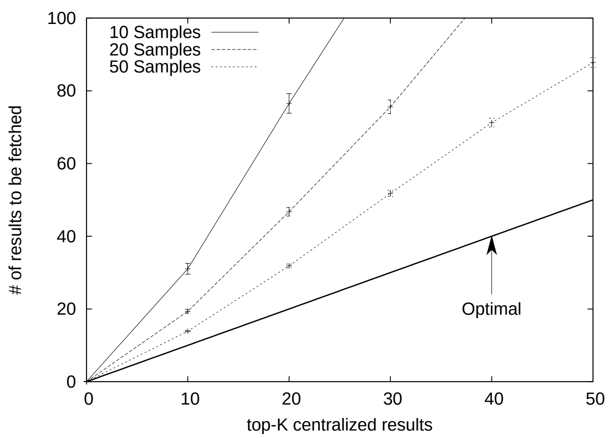

Figure 6: Average fetch of the distributed ranking scheme with 5000 nodes. The error bars correspond to 95% confidence interval.

compared to the network where documents are mapped uniformly to nodes. In a 5000 node network, this increases by 35% compared to the uniform case. The results in both the network sizes with 50 random samples, however, are comparable to the uniform case.

4.3 Consistency

In our system, a query vector is generated using independent samples each time it is evaluated. This leads to different weights being assigned to the terms during different evaluations of the same query. This can increase the variance in ranking, and potentially lead to different results for different evaluations of the query. In this experiment, we show that is not the case, and that the results are minimally affected by the different samples.

We use a network size of 1000 nodes with documents mapped uniformly to these nodes. We sample 20 random nodes while computing the query vector. We use Q_{5K} and Q_{rare} query sets for this experiment. We record the top-50 results for different runs and compare the results against each other and against the centralized implementation.

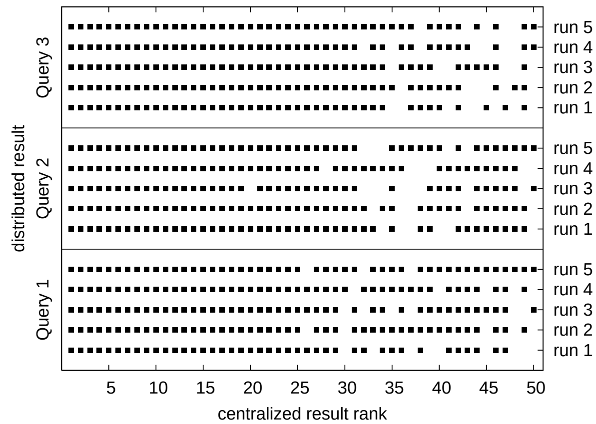

Figure 7: Consistency of top-50 results in distributed ranking for three different queries from Q_{5K} set

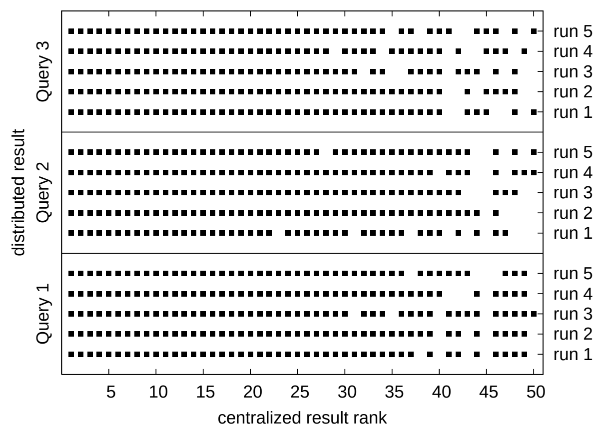

Figure 8: Consistency of top-50 results in distributed ranking for three different queries from Q_{rare} set

Figures 7 and 8 show the results obtained during five representative runs for three representative queries each. For each run, the figure includes a small box corresponding to a document ranked in the top-50 by centralized VSM if and only if this document was retrieved during this run. For example, in Figure 7, query 1, run 2 retrieved documents ranked $1 \ldots 25$, but did not return the document ranked 26 in its top 50 results. Also, note that the first 25 centrally ranked documents need not necessarily be ranked exactly in that order, but each of them were retrieved within the top-50.

There are two main observations to be drawn: first, the sampling does not adversely affect the consistency of the results, and different runs return essentially the same results. Further, note that these results show that the coverage of the top results is uniformly good, and the documents that are not retrieved are generally ranked towards the bottom of the top-50 by the centralized ranking. These observations also apply to the result with rare queries (Figure 8). In fact, a detailed analysis of our data shows that this trend holds in our other experiments as well.

Network setup	Top-K results														
	top-10 results			top-20 results			top-30 results			top-40 results			top-50 results		
	Avg	SD	Med	Avg	SD	Med	Avg	SD	Med	Avg	SD	Med	Avg	SD	Med
500 nodes	7.75	1.26	7.86	16.11	1.65	16.34	25.08	1.85	24.86	33.62	2.06	33.70	42.24	2.57	42.28
1000 nodes	7.99	1.05	8.08	16.33	1.42	16.40	24.58	1.92	24.72	32.98	2.41	33.20	41.59	2.46	42.2
2000 nodes	7.67	1.31	8.14	15.85	1.85	16.00	23.96	2.05	24.16	32.00	2.61	32.04	40.11	3.04	40.68
5000 nodes	6.95	1.34	7.04	15.21	2.17	15.64	22.99	2.86	23.70	30.66	3.26	31.16	38.85	3.45	39.36

Table 4: The mean, standard deviation and median of coverage when the number of nodes and documents scale proportionally. All the results use 20 random samples.

4.4 Scalability

In this experiment, we evaluate the scalability of our scheme with increasing system size. Theorem 2 states that the number of samples required is independent of the system size, under the condition that the size of the document set grows proportionally to the number of nodes. We demonstrate this fact by showing that coverage remains approximately constant as we increase the system size ten-fold (from 500 to 5000), while sampling the same number of nodes (20).

The number of documents in each experiment is 20 times the number of nodes in the system. For all the configurations, the terms used in queries occur in more than 10% of the total documents. For the 5000 node network, this corresponds to the Q_{pop} query set. In each case, we sample 20 random nodes to estimate the global weights.

Table 4 shows the results of our experiment. We present the mean, standard deviation and median coverage of our distributed scheme. As the table shows, the coverage of the distributed retrieval is very similar in most cases. This result confirms that our scheme depends almost entirely on the density of the number documents per node, and that it scales well as long as the density remains similar.

4.5 Reducing the number of document vectors stored on indexes

Recall our optimization (Section 3.4) to store document vectors only in the indexes of keywords whose weights are greater than a threshold w_{min}. In this experiment, we quantify the effect of this optimization. For this experiment, we used a network of 1000 nodes with documents distributed uniformly at random over the nodes. We use all the three query sets and sample 50 nodes to estimate the weights. Note that we normalize the vectors; so the term weights range between 0.00 and 1.00. We present results for thresholds ranging from 0.00 to 0.30. We compare the results retrieved from the centralized implementation with $w_{min} = 0.00$.

The results of this experiment are tabulated in Table 5. Coverage of distributed ranking is not adversely affected when the threshold is set to 0.05 or 0.10. However, larger thresholds (say 0.20 and above) discard relevant terms, and consequently decrease rank quality appreciably.

In order to understand the reduction obtained by using the threshold, we recorded the total number of index entries in the system for each threshold. The total number of index entries in our system is 15.9M when the threshold is 0.0. Our experiments show a reduction of 55.5% entries when we use a threshold of 0.05. Increasing the threshold to 0.1 leads to an additional 30% reduction in index size. A threshold value of 0.1 seems to be a reasonable tradeoff between search quality and decreased index size.

4.6 Eliminating the Random Probes

The random probes used to create query (and document) vectors are used to establish accurate values of D and D_t. We ran an experiment to test the sensitivity of the coverage metric to the accuracy with which D is

	Top-K results	Weight threshold (corresp. space reduction (%)) 0.0 (0.0)	0.05 (55.5)	0.10 (85.0)	0.20 (97.2)	0.30 (99.3)
Q_{5K}	10	9.42	9.40	9.10	8.10	4.00
	20	18.62	18.62	17.96	15.12	6.58
	30	27.68	27.68	27.08	22.04	7.96
	40	37.24	37.06	35.14	28.7	7.96
	50	46.44	46.34	45.24	34.7	7.96
Q_{pop}	10	9.1	9	8.96	7	2
	20	18.5	18.32	18.22	13.08	4.78
	30	27.9	27.74	27.72	18	5.78
	40	36.8	36.7	36.7	21.94	7.94
	50	46.85	46.36	46.36	25.46	7.98
Q_{rare}	10	8.9	8.88	8.88	8.78	8.12
	20	17.98	17.94	17.94	17.84	8.44
	30	26.86	26.88	26.88	26.06	8.44
	40	36.24	36.32	36.28	28.1	8.44
	50	45.36	45.52	45.26	28.32	8.44

Table 5: The median coverage of distributed ranking for different weight thresholds. The numbers in parenthesis show the reduction in the size of the indexes corresponding to the different thresholds.

D-factor	top-K 10	20	30	40	50
0.50	8.74	17.68	25.68	34.26	42.89
0.75	9.58	19.21	28.32	37.68	47.16
1.00	10.00	20.00	30.00	40.00	50.00
1.25	9.63	19.26	28.63	38.21	47.89
1.50	9.32	18.63	28.16	37.00	46.53
2.00	9.05	17.95	27.00	35.58	44.63
4.00	8.37	16.47	25.05	33.00	41.37
8.00	7.68	15.37	23.58	31.05	39.05
16.00	7.26	14.74	22.42	29.21	37.21

Table 6: Mean coverage with accurate D_t and inaccurate D.

known. Assuming that D_i is known accurately, the system achieves nearly 75% coverage (up to the top 50 items) even when the assumed D value is 16 times larger than the real value.

Exact values of D_t can usually be retrieved directly from the indexes of the query or document terms, of which there are usually few. These results therefore suggest eliminating the random probes in favor of a new vector creation algorithm where D is maintained through a very low-priority background process (e.g., gossiping), and D_t's are retrieved directly from term indexes.

One potential drawback is that low-weight document terms might not be inserted into system indexes because they fall below threshold, resulting in an inaccurate D_t. By definition, however, these terms are not often useful. The system could in any case default to probes when such terms are encountered.

A more serious problem is that querying indexes directly might adversely affect load balance. However, load balance issues could be addressed by caching of indexes or prior query results [11].

5 Related Work

Our paper builds on prior work on efficient lookup and storage schemes. We assume the existence of a lookup protocol provided by the underlying system. Such lookup protocols have been studied in detail both in a structured setting (e.g., Chord [26], Pastry [22], Kademlia [18], Viceroy [17] and Skipnet [13]) and in an unstructured setting (e.g., Yappers [9] and Local Minima Search(LMS) [19]).

Providing a useful search facility has been an important area of research. Prior work in searching can broadly be classified into two categories: traditional centralized approaches, and search strategies over structured P2P networks.

Classic Information Retrieval Centralized information retrieval and automatically ordering documents by relevance has long been area of much research. We discussed the Vector Space Method [25] in Section 2. Salton and Buckley [23], Dumais [7] and Frakes et al. [8] discuss different term weighting models. Salton et. al. [24] discuss the extension to the Boolean model using VSM. They show that the extended model has better result quality than the conventional boolean scheme. Latent Semantic Indexing (LSI) [6] is an extension to VSM that attempts to eliminate the issues of synonyms and polysemy. LSI employs singular value decomposition (SVD) to reduce the matrix generated by VSM. LSI then uses this reduced matrix to answer queries. LSI has empirically been shown to be very useful, especially with small collections. While techniques to compute SVD (e.g., using gossip [14]) in a distributed setting are known, it is still an open question as to how the data should be organized for effective retrieval. PageRank [20] has been deployed, with much success, to rank web pages. PageRank uses the hypertext link information to compute the rank of a web page. However, PageRank cannot be applied in a P2P setting because of the lack of links between documents.

Distributed Search over P2P systems The idea of using Vector Space Methods has been applied previously in the context of P2P similarity search. pSearch [29] uses VSM and LSI to generate document and query vectors, and maps these vectors to a high-dimension P2P system. The authors show that the query is mapped close to the documents relevant to the query and controlled flooding is employed to fetch relevant documents. The authors, however, do not compute the vectors from scratch. Instead, they use precomputed vectors and then aggregate information using combining trees. While pSearch has the information to rank search results, the way data is organized makes it inefficient to return a globally ranked result set.

Bhattacharya et.al. [3] use similarity-preserving hashes (SPH) and the cosine similarity to compute similar documents over *any* DHT. Using Hamming distance as the metric, they show that SPH guarantees that the keys generated for similar documents are clustered. Given a query and a similarity threshold, they fetch all documents whose cosine similarity is greater than the threshold. Once again, in this system, the default organization of the data makes it inefficient to return globally ranked results. Further, the authors also assume that the document vectors are given to them.

PlanetP [5] is a content-based search scheme that uses VSM for ranking search results. Each node in PlanetP stores the document vector locally. Nodes also store digests of the content stored in all other nodes. PlanetP uses gossiping to spread this information. While evaluating queries, nodes rank the *peers* using the digest and forward their queries to them. Results are evaluated locally in these peers using the cosine similarity and returned to the node that started the query. The paper, however, does not mention how the vectors are generated.

An alternative to random sampling is to use gossip to aggregate the global information. Bawa et al. [2] show techniques to estimate the number of nodes and other aggregates in unstructured systems. Similar approaches could be applied to estimate total number of documents in the system. Note, however, that gossiping would be expensive when used to compute the number of documents with the individual terms.

There have also been proposals that use distributed inverted indexes to support the boolean query model. Reynolds et al. [21] were the first to propose the use of inverted indexes for searching in structured P2P systems. The paper also presents a technique using Bloom filters for efficient intersection computation. Gnawali [10], in his thesis, proposes storing indices that are intersections of two keywords. The design is motivated by the fact that a large fraction of the queries contain two or more keywords. eSearch [28] is a boolean query system for textual data over Chord [26]. In each index, eSearch stores the document entries and all the keywords in the document. Such a design eliminates the need to contact the multiple indexes while evaluating boolean queries. Odissea [27] uses a two-tier architecture to do full-text searches using inverted indexes. Peers are arranged in a Chord-like ring and store indexes that get mapped to them. The motivation behind Odissea is to build a distributed infrastructure to perform web-search. These systems can be directly used as platforms for our ranking protocol.

Finally, Li et al. [16] question the feasibility of Web indexing and searching in P2P systems. They conclude that techniques to reduce the amount of data transferred are required for the such systems to be feasible. We believe that ranking can be successfully utilized as one measure to reduce data transmitted and is hence an important component in making P2P indexing and searching practical.

6 Conclusions

In this paper, we have presented a distributed algorithm for ranking search results. Our solution demonstrates that distributed ranking is feasible with little network overhead. Unlike previous work, we do not assume that the document vectors are provided to the system. Instead, our algorithm computes such vectors by using random sampling to estimate term weights. Through simulations and formal analysis, we show that the retrieval quality of our approach is comparable to that of a centralized implementation of VSM. We also show that our approach scales well under the reasonable condition that the size of the document set grows with the number of nodes.

There are several areas worthy of further investigation. Performance could potentially be improved by mechanisms such as relevance feedback and caching. Our analysis could be extended to account for popularity-based replication. While our system provides a first solution to distributed ranking, other approaches, e.g. using gossip (as described in Section 3.4), are also promising. We plan to investigate these approaches in our future work.

References

[1] Trec: Text REtrieval Conference. http://trec.nist.gov/.

[2] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. Estimating aggregates on a peer-to-peer network. Technical report, Computer Science Dept., Stanford University, 2003.

[3] I. Bhattacharya, S. R. Kashyap, and S. Parthasarathy. Similarity searching in peer-to-peer databases. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 329–338, 2005.

[4] C. Buckley. Implementation of the smart information retrieval system. Technical report, Dept. of Computer Science, Cornell University, Ithaca, NY, USA, 1985.

[5] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003.

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

[7] S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, and Computers*, 23(2):229–236, 1991.

[8] W. B. Frakes and R. Baeza-Yates. *Information retrieval: data structures and algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[9] P. Ganesan, Q. Sun, and H. Garcia-Molina. Yappers: A peer-to-peer lookup service over arbitrary topology. In *22nd Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM)*, San Francisco, USA, March 2003.

[10] O. D. Gnawali. A keyword set search system for peer-to-peer networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[11] V. Gopalakrishnan, B. Bhattacharjee, S. Chawathe, and P. Keleher. Efficient peer-to-peer namespace searches. Technical Report CS-TR-4568, University of Maryland, College Park, MD, February 2004.

[12] D. Harman, editor. *The Second Text Retrieval Conference*, Gaithersburg, MD, 1994. National Institute of Standards and Technology.

[13] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Fourth USENIX Symposium on Internet Technologies and Systems (USITS '03)*, Seattle, WA, March 2003.

[14] D. Kempe and F. McSherry. A decentralized algorithm for spectral analysis. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 561–568, New York, NY, USA, 2004. ACM Press.

[15] V. King and J. Saia. Choosing a random peer. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 125–130, New York, NY, USA, 2004.

[16] J. Li, B. T. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, Feb 2003.

[17] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *21st ACM Symposium on Principles of Distributed Computing (PODC '02)*, Monterey, CA, August 2002.

[18] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *First International Workshop on Peer-to-Peer Systems (IPTPS '01)*, pages 53–65, London, UK, 2002.

[19] R. Morselli, B. Bhattacharjee, A. Srinivasan, and M. A. Marsh. Efficient lookup on unstructured topologies. In *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 77–86, New York, NY, USA, 2005.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation algorithm: bringing order to the web. Technical report, Dept. of Computer Science, Stanford University, 1999.

[21] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of IFIP/ACM Middleware 2003*, 2003.

[22] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[23] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[24] G. Salton, E. A. Fox, and H. Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):1022–1036, 1983.

[25] G. Salton, A. Wong, and C. Yang. A vector space model for information retrieval. *Journal for the American Society for Information Retrieval*, 18(11):613–620, 1975.

[26] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[27] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In *6th International Workshop on the Web and Databases (WebDB)*, June 2003.

[28] C. Tang and S. Dwarakadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of USENIX NSDI '04 Conference*, San Fransisco, CA, March 2004.

[29] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of ACM SIGCOMM '03 Conference*, pages 175–186, Karlsruhe, Germany, 2003. ACM Press.

A Appendix

A.1 Proof of Theorem 1

The proof is an application of the Chernoff bound. For $i = 1, \ldots, k$, let Y_i be the random variable representing the number of documents found during the i-th sample. Note that $\tilde{D} = \sum_{i=1}^{k} Y_i$. In order to apply the Chernoff bound, we need random variables in the interval $[0, 1]$. Let $X_i = Y_i/M$ and let $X = \sum_{i=1}^{k} X_i = \tilde{D}/M$. Define:

$$\mu = \mathrm{E}[X] = \frac{kD}{MN}.$$

Since X_i are in $[0, 1]$ and are independent, we can use the Chernoff bound, which tells us that for any $0 < \delta \le 1$.

$$\Pr[|X - \mathrm{E}[X]| > \delta\,\mathrm{E}[X]] \le 2e^{-\frac{\mu\delta^2}{3}},$$

which can be rewritten as:

$$\Pr[(1-\delta)\frac{kD}{N} \le \tilde{D} \le (1+\delta)\frac{kD}{N}] > 1 - 2e^{-\frac{\mu\delta^2}{3}}.$$

We now impose the constraint that the probability above is at least $1 - \epsilon$:

$$2e^{-\frac{\mu\delta^2}{3}} \le \epsilon,$$

from which we derive the bound on k:

$$k \ge \frac{3}{\delta^2}\frac{M}{D/N}\ln(2/\epsilon).$$

A.2 Proof of Theorem 2

For every document d, define the random variable X_d as indicator of the event that the document d is randomly stored at one of the k sampled nodes. For simplicity assume that the k sampled nodes are distinct, so $\mathrm{E}[X_d] = k/N$. Defining $X = \sum_d X_d$ and applying the Chernoff bound to X yields the result. The details are analogous to the proof of Theorem 1.