Distributed Ranked Search

Vijay Gopalakrishnan[1], Ruggero Morselli[2], Bobby Bhattacharjee[3], Pete Keleher[3], and Aravind Srinivasan[3]

[1] AT&T Labs – Research
[2] Google Inc
[3] University of Maryland

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 results, not necessarily 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 decentralized setting. We analyze the overhead of our approach, and quantify 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 scale with little overhead to large systems, and b) the accuracy of the results obtained using distributed ranking is comparable to that of a centralized implementation.

1 Introduction

Search infrastructures often order the results of a query by application-specific notions of *rank*. Users generally prefer to be presented with small sets of ranked results rather than unordered sets of all results. For example, a recent Google search for "HiPC 2007" matched over 475,000 web pages. The complete set of all results would be nearly useless, while a very small set of the top-ranked results would likely contain the desired web site. Moreover, 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 decentralized manner is difficult because decisions about which results to return are made locally, but the basis of the decisions, rank, is a global property. Technically, we could designate one node as being responsible for ranking all the search results. Such an approach, however, would result in this peer receiving an unfair amount of load. Further, there are the issues of scalability and fault-tolerance with using just one node.

The main contribution of this paper is the design and evaluation of a decentralized algorithm that efficiently and consistently ranks search results over arbitrary documents. Our approach is based on approximation techniques using uniform random sampling, and the classic centralized Vector Space Model (VSM) [1]. Our results apply to both structured and unstructured networks.

Our analysis shows that the cost of our sampling-based algorithm is usually small and *remains constant as the size of the system increases*. We present a set of simulation results, based on real document sets from the TREC collection [2], 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, and that the approach is robust to sampling errors, initial document distribution, and query location.

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.

2 The Vector Space Model (VSM)

The Vector Space Model (VSM) is a classic information retrieval model for ranking results. 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 $\boldsymbol{d} = (w_{1,d}, w_{2,d}, \ldots, w_{T,d})$. A query is also represented as a vector $\boldsymbol{q} = (w_{1,q}, w_{2,q}, \ldots, w_{T,q})$, 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; relevant documents will differ from the query vector by a small angle while irrelevant documents will differ by a large angle. Given two vectors X and Y, the angle θ between them can be computed using $\cos\theta = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$. This equation is also known as the cosine similarity, and has been used in traditional information retrieval to identify and rank relevant results.

2.1 Generating Vector Representation

The vector representation of a document is generated by computing the *weight* of each term in the document. The key is to assign weights such that terms that capture the semantics in the document and therefore help in discriminating between the documents are given a higher weight.

Effective term weighting formulae have been an area of much research (e.g., [3,4]), unfortunately with little consensus. While any of the commonly used formulae can be used with our scheme, we use the weighting formula used in the SMART [5] system as it has shown to have good retrieval quality in practice:

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

where $w_{t,d}$ 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

In this section, we present our distributed VSM ranking system for keyword-based queries. 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 first describe our assumed system model and then discuss each of these components in detail.

3.1 System Model

Our ranking algorithm is designed for both structured and unstructured P2P systems. Our algorithm constructs an inverted index for each keyword and these indexes are distributed over participating nodes (which are assumed to be cooperative). An *inverted index* of a keyword stores the list of all the documents having the keyword. We assume that the underlying P2P system provides a *lookup* mechanism necessary to map indexes to nodes storing them. While APIs for lookup are available in all structured systems, we rely on approaches such as LMS [6] and Yappers [7] for lookup over unstructured systems. The underlying P2P system dictates how the indexes are mapped to nodes; structured P2P systems store indexes at a single location, while an index may be partitioned over many locations in unstructured systems. Each node *exports* a set of documents when it joins the system. A set of keywords (by default, all words in the document) is associated with the document. The process of exporting a document consists of adding an entry for the document in the index associated with each keyword. When querying, users submit queries containing keywords and may specify that only the highest ranked K results be returned. The system then computes these K results in a distributed manner and returns the results to the user.

3.2 Generating Document Vectors

Recall that to generate a document vector, we need to assign weights to each term of the document. Also recall Equation (1), 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 can be easily obtained by counting the frequency of the word in the document. 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.

Let N be the number of nodes in the system, and D and D_t be as above. We choose k nodes uniformly at random. This can be done either with random walks, in unstructured systems [6], or routing to a random point in the namespace in structured systems [8]. We then compute the total number $\tilde{D}$ of documents and $\tilde{D}_t$ of documents with term t at the sampled nodes. 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, then $\tilde{D}$ and $\tilde{D}_t$ are reasonably close to their expected value. If that is the case, then we can estimate D/D_t as $\frac{D}{D_t} \approx \frac{\tilde{D}}{\tilde{D}_t}$

To derive a sufficient condition for this approximation, we introduce two new quantities. Let M and M_t be the maximum number of documents and maximum number of documents with the term t, respectively, on a node. 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. The estimate can be "bad" with a small probability (ϵ). As usual, we let e denote the base of the natural logarithm.

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

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

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

$$\Pr[(1-\delta)\frac{kD}{N} \leq \tilde{D} \leq (1+\delta)\frac{kD}{N}] \geq 1-\epsilon. \tag{3}$$

Proof. The proof is an application of the Chernoff bound [9]. 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 the X_i are in $[0, 1]$ and are independent, we can use the Chernoff bound, which tells us that for any $0 < \delta \leq 1$.

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

which can be rewritten as:

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

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

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

from which we derive the bound on k:

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

□

If we replace D, M, $\tilde{D}$ with D_t, M_t, $\tilde{D}_t$, the theorem also implies that if

$$k \geq \frac{3}{\delta^2} \frac{M_t}{D_t/N} \ln(2/\epsilon) \tag{4}$$

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

The following observations follow from Theorem 1:

- Theorem 1 tells us that for a good estimate, the number of samples needed does not depend on N directly, but on 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.
- If the number of documents D is much larger than the system size N and queries consist 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. In general, sampling is a poor approach for estimating rare properties and alternative approaches are required.
- The number of samples is proportional to the ratios between the maximum and the average number of documents stored at a node (i.e., $\frac{M}{D/N}$ and $\frac{M_t}{D_t/N}$). This means that, as the distribution of documents in the system becomes more imbalanced, more samples are needed to obtain accurate results.

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 at random and independently. Such a distribution approximately models the behavior of a DHT. In this special case, a stronger version of Theorem 1 holds, provided we do sampling without replacement (alternatively, we do *rejection sampling* where, if we sample a node that has already been sampled, we reject this sample and sample again):

Theorem 2. *Let D, N, k be as above and assume each document is stored independently and uniformly at random at one node. For any $\epsilon > 0$, if we choose k samples* ***without replacement*** *with*

$$k \geq \frac{3}{\delta^2} \frac{1}{D/N} \ln(2/\epsilon), \tag{5}$$

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

$$\Pr[(1-\delta)kD/N \leq \tilde{D} \leq (1+\delta)kD/N] \geq 1 - \epsilon. \tag{6}$$

This probability is taken over the random distribution of the documents, as well as the randomness in our sampling.

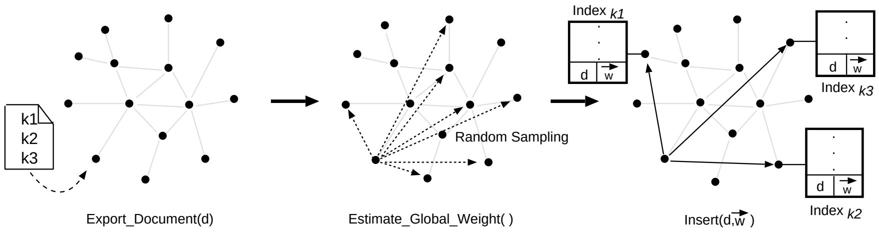

Fig. 1: Various steps in exporting documents and their vector representation

Proof. As opposed to the proof of Theorem 1, we now crucially use the fact that the documents have been mapped *independently*. For every document d, define the random variable X_d as the indicator of the event that the document d is stored at one of the k sampled nodes (i.e., X_d is 1 if this event happens, and is 0 otherwise). So, $\mathrm{E}[X_d] = k/N$. Defining $X = \sum_d X_d$, we see that $\mathrm{E}[X] = kD/N$.

We crucially note that the different random variables X_d are *independent*, which is a consequence of the fact that we are conducting sampling without replacement. To see this, suppose we have distinct documents $d_1, d_2, \ldots, d_i$, and are given the values of $X_{d_1}, X_{d_2}, \ldots, X_{d_{i-1}}$; even given this information, we now argue that the conditional expectation of X_{d_i} is k/N. Let us expose the set S of k distinct samples in a manner consistent with the given information about $X_{d_1}, X_{d_2}, \ldots, X_{d_{i-1}}$. Since the documents are stored uniformly at random and independently, the probability that d_i has been stored in S, is exactly k/N. Thus, the random variables X_d are indeed independent of each other. We now apply the Chernoff bound to X to yield our result; the calculation-details are the same as in the proof of Theorem 1. $\square$

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.

3.3 Storing Document Vectors

Document vectors need to be stored such that a query relevant to the document can quickly locate them. We store document vectors in distributed inverted indexes. As mentioned previously, an *inverted index* for a keyword t is a list of all the documents containing t. For each keyword t, our system stores the corresponding inverted index like any other object in the underlying P2P lookup system. This choice allows us to efficiently retrieve vectors of all documents that share at least one term with the query.

Figure 1 shows the process of exporting a document. We first generate the corresponding document vector by computing the term weights, which uses the procedure described in section 3.2. Next, using the underlying storage system API, 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 for t is stored at the node responsible for the key corresponding to t. The underlying protocol can be used to efficiently locate this node. A similar approach using inverted indexes has previously been used by [10–13] for searching in structured systems. In unstructured networks, indexes would need to be partitioned or replicated [6, 7].

Reducing storage cost So far, we have assumed that each word in the document is a keyword. Hence an entry is added for the document in the indexes of all the words in the document. A document, however, will not appear among the top few results when its weights for the query terms are low. Hence, not having these low-weight entries in the index does not reduce the retrieval quality of the top few results. We use this intuition to reduce the cost of propagating and storing vectors in indexes. 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 part of the vector. This heuristic has also been successfully used in eSearch [10].

3.4 Evaluating Query Results

A query is evaluated by converting it into a vector representation, and then computing the cosine similarity with respect to each "relevant" document vector. We compute query vectors using the same techniques used to generate the document 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 of that keyword. We then compute the cosine similarity between the query and each of the document vectors stored in the index. This gives us a ranking of the documents available in this index. Finally, we fetch the top-K results computed at each of the indexes and compute the *union* of these result sets. The top-K documents in this union, sorted in the decreasing order of cosine similarities, give us our final result set.

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.

Experimental setup We use the TREC [2] Web-10G data-set for our documents. We used the first 100,000 documents in 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 distribution in unstructured systems.

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

Network Setup	Number of Samples	Top-K results 10	20	30	40	50
1000 uniform	10	8.49 (1.08)	16.99 (1.20)	25.30 (1.55)	33.68 (2.07)	42.28 (2.01)
	20	8.90 (0.99)	17.81 (1.04)	26.44 (1.26)	35.23 (1.87)	44.30 (1.82)
	50	9.28 (0.82)	18.63 (0.82)	27.66 (1.04)	36.08 (1.45)	46.30 (1.46)
5000 uniform	10	6.78 (1.39)	13.58 (1.74)	20.43 (2.39)	27.35 (2.99)	34.59 (3.40)
	20	7.74 (1.29)	15.41 (1.46)	22.92 (1.96)	30.50 (2.47)	38.49 (2.58)
	50	8.52 (1.09)	16.96 (1.18)	25.20 (1.56)	33.59 (2.11)	42.34 (1.98)
1000 Zipf	10	8.27 (1.15)	16.52 (1.26)	24.66 (1.71)	32.82 (2.21)	41.20 (2.27)
	20	8.82 (0.99)	17.63 (1.06)	26.22 (1.35)	34.83 (1.93)	43.70 (1.88)
	50	9.26 (0.80)	18.54 (0.88)	27.52 (1.12)	36.71 (1.49)	46.12 (1.56)
5000 Zipf	10	6.09 (1.54)	12.29 (1.97)	18.58 (2.68)	25.01 (3.39)	31.67 (3.97)
	20	7.34 (1.31)	14.71 (1.62)	21.89 (2.10)	29.34 (2.64)	36.93 (2.90)
	50	8.41 (1.13)	16.73 (1.22)	24.92 (1.61)	33.22 (2.08)	41.71 (2.03)

Table 1: Mean and Std. Deviation of coverage with the distributed ranking scheme.

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. Each result presented (except for details from individual runs) is an average of 50 runs.

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

1. **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.
2. **Fetch** We define fetch as the minimum number R' such that, when the user obtains the set of $\mathcal{R}'$ 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 returned by the distributed case had been B, then the fetch for $K = 3$ would be 4.
3. **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.

4.1 Coverage

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.

The results are presented in Table 1. It is clear from Table 1 that the distributed ranking scheme performs very similar to the centralized implementation. On a 1000 node network with documents distributed uniformly, the mean accuracy for the top-K results is close to 93% with 50 random samples. Even with 10 random samples, the results are only slightly worse at 85% accuracy.

With 5000 nodes, the retrieval quality is not as high as a network with 1000 nodes. With 20 random samples, the mean accuracy is 77% for top-K results. There is a 8% increase in mean accuracy when we increase the sampling level and visit 1% (50) of the nodes. This result is a direct consequence of Theorem 1. 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 1 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 similar to that of the uniformly distributed case. With 10 samples, however, the mean 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 7%. Hence, we believe our scheme can be applied over lookup protocols on unstructured networks without appreciable loss in quality.

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 experiments 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 2 and 3. 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 fetch is quite small even if only ten nodes are sampled. For instance, sampling 10 nodes, we need 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 need 11 entries 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 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 lesser sampling, however, we need to fetch a lot more results to cover the top-K. This behavior, again, is predicted by Theorem 1: when the number of nodes increases without a corresponding increase in the number of documents, the samples needed to guarantee a bound on 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% compared to the network where documents are

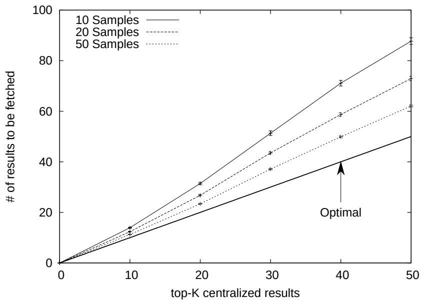

Fig. 2: Average fetch of the distributed ranking scheme with 1000 nodes. The error bars correspond to 95% confidence interval.

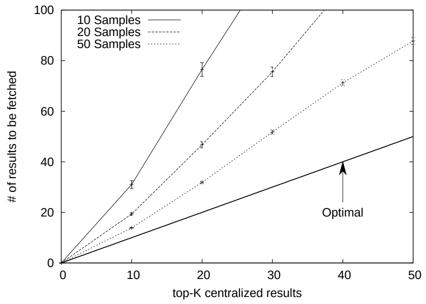

Fig. 3: Average fetch of the distributed ranking scheme with 5000 nodes. The error bars correspond to 95% confidence interval.

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 new query vector is generated each time a query 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 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 record the top-50 results for different runs and compare the results against each other and against the centralized implementation.

Figure 4 shows 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 4, query 1, run 2 retrieved documents ranked $1 \dots 25$, 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. In fact, a detailed analysis of our data shows that this trend holds in our other experiments as well.

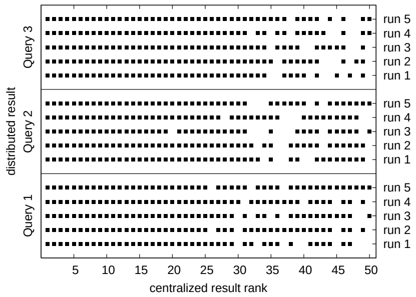

Fig. 4: Consistency of top-50 results in distributed ranking for three different queries from Q_{5K} set

Network Setup	Top-K results 10	20	30	40	50
500 nodes	7.75	16.11	25.08	33.62	42.24
1000 nodes	7.99	16.33	24.58	32.98	41.59
2000 nodes	7.67	15.85	23.96	32.00	40.11
5000 nodes	6.95	15.21	22.99	30.66	38.85

Table 2: Mean coverage when the number of nodes and documents scale proportionally.

4.4 Scalability

In this experiment, we evaluate the scalability of our scheme with increasing system size. Theorem 1 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 2 shows the mean 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 storage cost

Recall our optimization 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 20 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 3. 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 entries, 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.

Weight Threshold	Q_{5K} 10	30	50	Q_{pop} 10	30	50	Q_{rare} 10	30	50
0.0 (0.0)	8.90	26.44	44.30	8.32	26.31	44.49	8.59	26.01	44.47
0.05 (55.5)	8.90	26.44	44.34	8.33	26.32	44.40	8.50	26.01	44.47
0.10 (85.0)	8.90	26.40	44.22	8.32	26.17	43.87	8.59	26.01	43.54
0.20 (97.2)	7.64	20.43	30.97	6.39	17.90	26.70	8.46	21.41	28.43
0.30 (99.3)	4.53	7.98	8.88	2.79	6.84	9.90	6.66	9.78	9.99

Table 3: The mean 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.

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 trade-off between search quality and decreased index size.

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 [14] and Pastry [15]) and in an unstructured setting (e.g., Yappers [7] and LMS [6]). 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 [1] in Section 2. Latent Semantic Indexing (LSI) [16] 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 [17]) in a distributed setting are known, it is still an open question as to how the data should be organized for effective retrieval. There has also been work on implementing PageRank [18] in a distributed setting (e.g., [19]), but it cannot be applied on an arbitrary document set because of the lack of hyper-links.

Fagin et al.'s Threshold Algorithm (TA) [20] can also be used to compute the top-K results. Given sorted lists of <`item,value`> pairs, the value of an item x is defined by applying a monotonic function to the values that x has in each list. [21] and [22] provide optimizations to TA to reduce the amount of data transferred in a distributed setting. In this paper, we store the entire vector in each index entry and use the union operation to group the results. Instead, we could just store the term weights in the index and use TA to compute the final result set. We plan to explore this option in the future.

Distributed Search over P2P systems The idea of using Vector Space Methods has been applied previously in the context of P2P search. PlanetP [23] is a content-based search scheme that uses VSM for ranking search results. Nodes store vectors locally, but gossip digests of their local content. Queries are evaluated by ranking the *nodes* first and then evaluating the query using VSM at the top-ranked nodes. pSearch [24] uses VSM and LSI to generate document and query vectors, and maps these vectors to a high-dimension P2P system. Bhattacharya et al. [25] 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 While pSearch and SPH have the information to rank search results, the way data is organized makes it inefficient to return a globally ranked result set. Also, none of them (PlanetP included) explain how to compute the document and query vectors. The work presented in this paper can be directly applied in both these settings to generate the document and query vectors.

There have been attempts to support ranked searches using other ranking strategies. Odissea [26], a P2P distributed search infrastructure, proposes to make use of TA to rank search results. The authors however do not mention how the individual term weights are computed. Balke et al. [27] propose a method to perform ranking over a P2P network consisting of super-peers, arranged in a HyperCuP topology.

There have also been proposals that use distributed inverted indexes to support the boolean query model. Reynolds et al. [12] were the first to propose the use of inverted indexes for searching in structured P2P systems. Loo et al. [13] design a hybrid solution that uses a DHT with inverted indexes to locate rare documents and Gnutella-style flooding for popular documents. eSearch [10] is a boolean query system for textual data over Chord [14]. 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. None of these directly support ranking search results.

Finally, Li et al. [28] 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.

7 Acknowledgments

We thank Divesh Srivastava and the referees for their comments and suggestions. This work was partially supported by NSF awards CCR-0208005, CNS-0626636, and NSF ITR Award CNS-0426683. Bobby Bhattacharjee was also supported in part by a fellowship from the Sloan Foundation. Part of Aravind Srinivasan's work was done while on sabbatical at the Network Dynamics and Simulation Science Laboratory of the Virginia Bioinformatics Institute, Virginia Tech.

References

1. Salton, G., Wong, A., Yang, C.: A vector space model for information retrieval. Journal of the American Society for Information Retrieval **18**(11) (1975) 613–620
2. TREC: Text REtrieval Conference. http://trec.nist.gov/ ()
3. Dumais, S.T.: Improving the retrieval of information from external sources. Behavior Research Methods, Instruments, and Computers **23**(2) (1991) 229–236
4. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Information Processing and Management **24**(5) (1988) 513–523
5. Buckley, C.: Implementation of the SMART information retrieval system. Technical report, Dept. of Computer Science, Cornell University, Ithaca, NY, USA (1985)
6. Morselli, R., Bhattacharjee, B., Srinivasan, A., Marsh, M.A.: Efficient lookup on unstructured topologies. In: Proceedings of the 24th symposium on Principles of distributed computing (PODC'05), New York, NY, USA (2005) 77–86
7. Ganesan, P., Sun, Q., Garcia-Molina, H.: 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 (2003)
8. King, V., Saia, J.: Choosing a random peer. In: Proceedings of the 23rd symposium on Principles of distributed computing (PODC '04), New York, NY, USA (2004) 125–130
9. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Annals of Mathematical Statistics **23** (1952) 493–509
10. Tang, C., Dwarakadas, S.: Hybrid global-local indexing for efficient peer-to-peer information retrieval. In: Proceedings of USENIX NSDI '04 Conference, San Fransisco, CA (2004)
11. Gopalakrishnan, V., Bhattacharjee, B., Chawathe, S., Keleher, P.: Efficient peer-to-peer namespace searches. Technical Report CS-TR-4568, University of Maryland, College Park, MD (2004)
12. Reynolds, P., Vahdat, A.: Efficient peer-to-peer keyword searching. In: Proceedings of IFIP/ACM Middleware. (2003)
13. Loo, B.T., Hellerstein, J.M., Huebsch, R., Shenker, S., Stoica, I.: Enhancing P2P file-sharing with an internet-scale query processor. In: Thirtieth International Conference on Very Large Data Bases (VLDB '04), Toronto, Canada (2004) 432–443
14. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SIGCOMM '01, San Diego, California (2001)
15. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proceedings of IFIP/ACM Middleware, Heidelberg, Germany (2001)
16. Deerwester, S., Dumais, S., Landauer, T., Furnas, G., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science **41**(6) (1990) 391–407

17. Kempe, D., McSherry, F.: A decentralized algorithm for spectral analysis. In: STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, New York, NY, USA, ACM Press (2004) 561–568
18. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation algorithm: bringing order to the web. Technical report, Dept. of Computer Science, Stanford University (1999)
19. Wang, Y., DeWitt, D.J.: Computing PageRank in a distributed internet search engine system. In: Thirtieth International Conference on Very Large Data Bases (VLDB '04), Toronto, Canada (2004) 420–431
20. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences (JCSS) **66**(4) (2003) 614–656
21. Cao, P., Wang, Z.: Efficient top-k query calculation in distributed networks. In: PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing, New York, NY, USA, ACM Press (2004) 206–215
22. Michel, S., Triantafillou, P., Weikum, G.: KLEE: A framework for distributed top-k query algorithms. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05), Trondheim, Norway (2005) 637–648
23. Cuenca-Acuna, F.M., Peery, C., Martin, R.P., Nguyen, T.D.: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In: Proceedings of the 12th Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press (2003)
24. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: Proceedings of ACM SIGCOMM '03, Karlsruhe, Germany, ACM Press (2003) 175–186
25. Bhattacharya, I., Kashyap, S.R., Parthasarathy, S.: Similarity searching in peer-to-peer databases. In: Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS'05). (2005) 329–338
26. Suel, T., Mathur, C., Wu, J., Zhang, J., Delis, A., Kharrazi, M., Long, X., Shanmugasundaram, K.: Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In: 6th International Workshop on the Web and Databases (WebDB). (2003)
27. Balke, W.T., Nejdl, W., Siberski, W., Thaden, U.: Progressive distributed top-k retrieval in peer-to-peer networks. In: 21st International Conference on Data Engineering (ICDE'05). (2005) 174–185
28. Li, J., Loo, B.T., Hellerstein, J., Kaashoek, F., Karger, D., Morris, R.: On the feasibility of peer-to-peer web indexing and search. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA (2003)