TSS: Efficient Term Set Search in Large Peer-to-Peer Textual Collections

Hanhua Chen, *Member*, *IEEE*, Jun Yan, *Member*, *IEEE*, Hai Jin, *Senior Member*, *IEEE*, Yunhao Liu, *Senior Member*, *IEEE*, and Lionel M. Ni, *Fellow*, *IEEE*

Abstract—Previous multikeyword search in DHT-based P2P systems often relies on multiple single keyword search operations, suffering from unacceptable traffic cost and poor accuracy. Precomputing term-set-based index can significantly reduce the cost but needs exponentially growing index size. Based on our observations that 1) queries are typically short and 2) users usually have limited interests, we propose a novel index pruning method, called TSS. By solely publishing the most relevant term sets from documents on the peers, TSS provides comparable search performance with a centralized solution, while the index size is reduced from exponential to the scale of O($n$log(n)). We evaluate this design through comprehensive trace-driven simulations using the TREC WT10G data collection and the query log of a major commercial search engine.

Index terms—Peer-to-peer, multikeyword searching, ranking.

1 Introduction

Since the emergence of peer-to-peer (P2P) [14], [17], [22] file sharing applications [9], such as Napster, Gnutella, and BitTorrent, millions of users have started using P2P tools in searching of desired data. P2P networks have also shown a great potential to become a network tool for sharing information on the Internet based on the following observations:

1. Information on the Internet resides on millions of Web sites and desktop disks. P2P-based search has the ability to leave the shared, but distributed, data at their origin, rather than collecting and maintaining them in a centralized repository.
2. There are significant performance, scalability, and availability benefits from distributing the indexing and querying load over large networks of collaborating peers.
3. P2P search is more robust than centralized search as the demise of a single server is unlikely to paralyze the entire search system.
4. There is growing concern about the fact that the world is dependent on a few quasi-monopolistic search engines. It is difficult to guarantee that they always bring objective results to users due to their susceptibility to commercial interests, possible biases in thematic coverage, or even censorship for different reasons [4].

- *H. Chen and H. Jin are with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China. E-mail: {chenhanhua, hjin}@hust.edu.cn.*
- *J. Yan is with Microsoft Research Asia (MSRA), Beijing, China. E-mail: junyan@microsoft.com.*
- *Y. Liu and L.M. Ni are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology (HKUST), Hong Kong. E-mail: {liu,ni}@cse.ust.hk.*

Manuscript received 26 Feb. 2009; revised 20 Aug. 2009; accepted 1 Sept. 2009; published online 6 Apr. 2010.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2009-02-0088.
Digital Object Identifier no. 10.1109/TC.2010.81.

Currently, the distributed retrieval systems are not yet comparable to the centralized search engines, due to the poor efficiency. Efficient searching of very large textual collections, which is required by applications such as Web search engines and large-scale digital library systems, appears to be very challenging to implement in a P2P architecture. Different from traditional search engines, it is often difficult, if not impossible, to maintain a centralized content index in a large-scale P2P network. Existing P2P retrieval systems often rely on a scalable distributed hash table (DHT) [29] to build distributed indexes.

A DHT is a class of distributed system that provides a lookup service similar to a hash table, while the responsibility for maintaining the mapping from keys to values is performed by a large number of cooperative distributed nodes. Any participating node can efficiently inserts (key, value) pairs and retrieve the value associated with a given key in a distributed manner. For example, Chord [29] uses consistent hashing function, such as SHA-1, to hash both keys and nodes (IP addresses) uniformly and randomly into the same ring-like identifier space. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k (successor node of k) in the identifier space. In an N-node network, each node maintains information only about $\mathrm{O}(\log N)$ other nodes, while a lookup requires $\mathrm{O}(\log N)$ messages.

By using DHTs, it is possible to build the index which maps an individual keyword to the global documents containing the keyword across the network. Using this single-keyword-based index, the list of entries for each keyword in a search query can be retrieved. Multikeyword search is conducted by merging all the lists. For example, consider a two-keyword query “peer-to-peer network,” the query is decomposed into “peer-to-peer” and “network,” and then, the two keywords are searched separately with a consequent intersection operation. Although effective techniques such as Bloom filters (BFs) [6], [23] are utilized to minimize the bandwidth consumption, the optimized search cost is still claimed unacceptable [15].

One effective way of reducing the search cost is to precompute the index using term set indexing, which maps a term set to a set of documents that contain the multiple terms. Such an approach, however, is not widely adopted because it often incurs exponentially growing index size. For example, given a document with n terms, a straightforward term-set-based indexing needs the scale of the storage cost of 2^n. Even if the length of the combination is fixed and only very few metadata words are indexed, the total number of entries is far more than that of the standard single-term-based index.

In this paper, we focus on the scalability problem of term set indexing in large-scale P2P search engines. By analyzing large traces collected from a major commercial search engine, we have the following observations: 1) queries are typically short and 2) users always have limited interests. Based on the observations, we propose a novel index pruning method for distributed term set index, called TSS. TSS utilizes the *Term Frequency-Inverse Document Frequency* (TFxIDF) [24] model to rank the relevance between a term set and the document from which the term set is extracted. It then selects to publish the top relevant term sets into the global index atop a novel distributed multidimensional hashing table. TSS utilizes a pushing synopsis-based gossip algorithm to collect the global statistical *Inverse Document Frequency* (IDF) information of terms in the P2P network. By hybridizing the unstructured protocol with the global index, the TSS term set index achieves multikeyword searching and TFxIDF-based ranking in large-scale P2P networks. We simulate the hybrid P2P network and evaluate the performance of our method on the TREC WT10G test collection [32] and the query logs of a major commercial search engine [33]. Results show that TSS design significantly reduces the search cost as well as provides comparable search performance and ranking accuracy with state-of-the-art centralized search engines at the index cost of $O(n\log(n))$.

The main contributions of TSS design are threefold: 1) the term-set-based distributed indexing method significantly reduces the bandwidth cost for multikeyword query search; 2) the index pruning method reduces the scale of the term set index from exponential to $O(n\log(n))$; and 3) we achieve global TFxIDF ranking, making multikeyword search scalable in large-scale P2P textual collections.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 introduces the TSS design. Section 4 describes the testbed. Evaluation methodology is presented in Section 5. Section 6 shows the experimental results. We conclude the paper in Section 7.

2 Related Work

Existing P2P search engines can generally be classified into two types: 1) federated engines using unstructured P2P networks and 2) distributed global inverted index in structured P2P networks.

In the first type, peers which maintain indexes of their local documents are organized in an ad hoc fashion. A basic search method is flooding. To reduce the search cost of flooding-based schemes, many approaches focus on the issue of query routing. The proposed algorithms commonly search a query at two steps: the peer step and document step. First, a group of peers with potential answers to the query are detected. Second, the query is submitted to the identified most relevant peers to return answers from their local indexes. PlanetP [8] proposes to maintain in every peer a global term-to-peer index which contains a mapping "$t \rightarrow p$" if term t is in the local index of peer p. For each query, it ranks the peers using the statistical information of the replicated global index. It is actually difficult for every node to collect and store such global index information in a large-scale P2P network. Lu and Callan [18] propose to use language model to locally rank the neighboring peers and forward the queries to the top-ranked neighbors which are most likely to have the answers.

The enhanced properties of the network topology are extensively used to improve the performance of the federated search engines. Sripanidkulchai et al. [28] identify the natural principle in Gnutella and other P2P networks called interest-based locality, which reveals that if a peer has a particular piece of content that a user is interested in, it is very likely that it will have other items (s)he might be interested in as well (in this paper, we use the terms "item" and "documents" interchangeably). In their design, peers with similar interest are linked together. By forwarding the queries through the interest-based shortcuts, a significant amount of unnecessary flooding is avoided. In Bibster [21], peers advertise their expertise, which contains a set of topics that the peer is an expert in. Other peers may accept these advertisements or not and create semantic links to their neighbors according to the expertise similarity. These semantic links form a semantic overlay for intelligent query routing. Li et al. [16] propose the SSW scheme which dynamically clusters peers with semantically similar data closer to each other and maps these clusters in a high-dimensional semantic space into a one-dimensional small-world network that has an attractive trade-off between search path length and maintenance costs.

By considering the inherent heterogeneity of peers, superpeer-based P2P architectures can further improved the search performance of a federated search engines. In this kind of architecture, peers with more memory, processing power, and network connection capacity provide distributed directory services for efficient and effective resource location. Thus, the peers that are limited in these resources will not become bottlenecks in the network. In [27], Shen et al. proposed a hierarchical summary indexing framework in which the content is summarized in different levels: document level, peer level, and superpeer level.

Another effective approach to improve the search performance of a federated search engines is the replication strategy. By replicating items and queries properly across the network, such strategies can effectively improve the search successful rate while avoiding exhaustively flooding the unstructured P2P networks. Existing replication strategies in unstructured federated P2P search networks can be divided into two categories: the query-popularity-aware replication approach [7] and the query-popularity-independent replication strategy [19], [31]. In the query-popularity-aware replication strategy, the number of replicas is related to the query rates/popularity, while in the query-popularity-independent replication strategy, all items and queries are equally replicated regardless of the popularity of the related queries. Inspired by the principle of birthday paradox, such schemes randomly distribute an optimal number of item replicas and query replicas into the network to achieve the high probability of rendezvous of an item and the related query on some node receiving both kinds of replicas. Although existing replication strategies enable an unstructured P2P system to return nearly all the matched documents in the network, it is difficult for them to satisfy the users of a content search engine, because the users are always impatient to pick out the desired documents from the tremendous amount of unranked results. On the contrary, they often expect a high-quality top-10 or top-20-ranked documents.

DHT-based searching engines are based on distributed indexes that partition a logically global inverted index in a physically distributed manner. Currently, there are two kinds of distributed index mechanisms: single-term-based inverted indexes and term-set-based indexes.

Existing DHTs can naturally support mapping every individual term to a set of documents across the network that contain the term. Using this single-term-based index, a list of entries/documents for a given keyword in a query can be retrieved by using existing DHT lookups. In [30], frequent terms of a document are selected to be published into the global index. When such a keyword is published, the list of other terms in the document is replicated with the identifier of the document in the posting list. Multikeyword search is performed by first locating the position of the DHT node which is responsible for one given keyword, and then, performing a local search in the posting list for the other left keywords in the query. Finally, the list of documents that contain all the keywords is returned as the results. Little is known about the performance of the full text search using selected keyword publishing, because a few selected frequent terms are not representative for a document [24].

An effective multikeyword searching scheme looks up the sets of documents for separate keywords in the query from multiple DHT nodes and returns the intersection. Although only a few nodes need to be contacted, each has to send a potentially large amount of data across the wide-area network, making the distributed intersection operations bandwidth costly. Reynolds and Vahdat [23] propose to use Bloom filters to encode the transferred lists while recursively intersecting the matching document set. The proposed scheme uses an inverse verification strategy to pick out the false positives of BFs. Chen et al. [6] prove that the optimal setting of BF in terms of communication cost is determined by the global statistical information of keywords. They further derive an effective approach for a real-world system to achieve optimal BF settings for multikeyword search through numerical analysis.

Another way to reduce the bandwidth cost is to precompute term set index that maps a set of terms to the list of global documents containing them. By avoiding the distributed intersection operations across the wide-area network, a term set index is potentially effective to reduce the communication cost [11]. The major drawback of term-set-based index is that the index size may grow exponentially. To reduce the unacceptable index size, Bender et al. [4] extend the single-term-based index. In their design, a DHT node responsible for keyword x caches additional posting lists for term sets which contain x and other terms searched together with x in previous queries. However, the incremental design also relies on the operations using single-term index, suffering from the same problems with the methods proposed in [6], [23].

3 TSS Design

In this section, we first introduce our design overview, and then, discuss the term set search scheme with the proposed pruning method. Section 3.3 shows how TSS ranks the results for a query. We present the pushing synopsis gossip algorithm for collecting global statistical information in Section 3.4.

3.1 Solution Outline

A TSS hybrid P2P network is a hybridization of 1) an unstructured P2P network which enables a gossiping protocol to gather global statistical information, and 2) a distributed hash table for global term-set-based indexing. Each peer participates in an unstructured network and acts as a structured DHT node as well. In the application of P2P Web [6], each peer represents a Web server.

With the facility of an unstructured network, TSS utilizes a push-synopsis gossip algorithm for gathering the global statistical information such as keyword popularity information including *term document frequency* (it denotes the number of global documents containing a term) and the total number of documents across the network. Based on the statistical information, every peer can easily compute the global *Inverse Document Frequency* of terms. Together with the local *Term Frequency* (TF) information, peers use a *vector space model* (VSM) to rank the relevance between term sets and the documents, and the top-ranked "*term set* $\rightarrow$ *document*" pairs are published as postings in the global index. Local TF information is piggybacked onto the postings to be inserted. Thus, the statistics collected via the unstructured protocol enable a DHT node to get TFxIDF results ranking for a query that matches a term set for which the node is responsible.

For the term-set-based indexing, we build a distributed multidimensional hash table, based on classical P2P lookup services. While our approach is general to any of these techniques, for simplicity, the following discussion assumes architecture closely related to the Chord protocol [29]. In order to reduce the index size of TSS, every peer uses an index pruning method which only inserts the postings (*term set* $\rightarrow$ *document*) that have top relevance ranking values in the local peer index.

To solve the load balance problem of DHTs caused by heterogeneous node capacity, we can utilize the virtual host technique [26], where each physical node is responsible for more than one logical host (virtual server) depending on its capacity.

3.2 Term-Set-Based Indexing

Given a set of terms and a full list of global documents which contain the words in the term set, the *Distributed Term-set-based Inverted Index* (DTII) can map the term set to the list of documents. In contrast to the traditional *Distributed Single-term-based Indexing* (DSII), DTII takes much lower communication cost when performing a multiterm

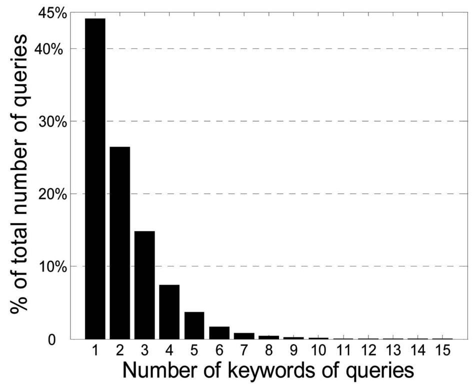

Fig. 1. Search engine query length distribution.

query. However, the scale of the naive DTII is considerably larger than DSII due to the fact that there are much more term sets than individual terms. Specifically, for a given document with n distinct terms, the set of all possible combined terms has $\sum_{i=1}^{n}\binom{n}{i} = 2^n$ elements in the worst case. To address this issue, we propose a pruning method to reduce the index size of DTII as well as preserve the search performance.

3.2.1 Observations on User Behaviors

We recently analyzed the query logs of a major commercial Web search engine. The query length distribution is plotted in Fig. 1, where we can see that 85.31 percent of the queries consist of at most three terms; 96.47 percent queries have five terms or less. Based on this observation, it is feasible to index the keyword set with only a small set of terms. Since the long-term combinations have little opportunity to be queried, the basic strategy of our pruning method is to only index the short-term combinations. For example, if we only index the term sets with no more than three terms, the index for an n-term long document can be significantly reduced to $\sum_{i=1}^{3}\binom{n}{3}$ with the complexity of $O(n^3)$.

3.2.2 Index Pruning Algorithm

To further reduce the index size, we propose a pruning method based on the TFxIDF ranking model. This concept is motivated by another observation that when users search the Web, they often prefer a small number of results with top relevance rank values. For example, user behavior of search engines has been studied by iProspect [2], which shows that about 88 percent users change their searches after reviewing the first three pages (commonly top 30 results). All the proportions are shown in Fig. 2.

Mathematically, given a document collection of a peer, for each document d, we only index the top $\lambda n \log(n)$ relevant term sets using the vector space model, where n is the number of keywords in d and λ is the parameter to control the pruning scale. Later in Section 6, we will show how the formal bound is reached. In vector space model, each document or term set is represented as a vector, where each dimension is associated with a distinct term. The value of each entry in a vector is the weight representing the importance of that term in the corresponding document or term sets. Given a term set s, we can compute the relevance of each document d with s by the cosine similarity:

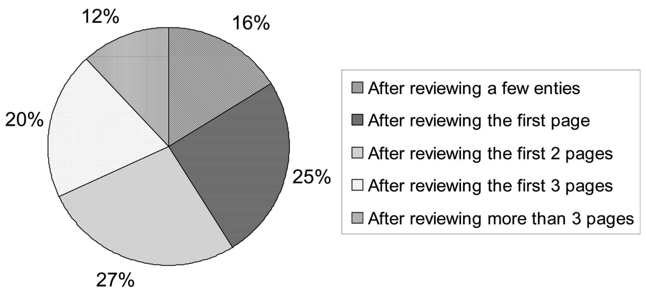

Fig. 2. User behavior study result by iProspect.

$$sim(s,d) = \frac{\sum_{t\in s} \omega_{s,t} \times \omega_{d,t}}{\sqrt{|s| \times |d|}}, \quad (1)$$

where $|s|$ and $|d|$ are the number of terms in s and d; $\omega_{s,t}$ and $\omega_{d,t}$ represent the weight of term t in term set s and the weight of term t in document d, respectively.

We use TFxIDF [24], the most popular and effective term weighting scheme, to measure the importance of a word in a document relative to a corpus. In a given document, a word's importance increases proportionally to the number of times it appears in the document and inversely proportional to its frequency in the corpus. A word occurring frequently in a particular document but rarely elsewhere in the corpus is thought to be "important" for that document only.

Specifically, TF denotes the term frequency property that is local and content-oriented to a document:

$$\omega_{d,t} = TF_t = 1 + \log(f_{d,t}), \quad (2)$$

where $f_{d,t}$ is the number of times that term t appears in document d.

The IDF quantifies the fact that terms appearing in many documents in a collection are less important:

$$\omega_{s,t} = IDF_t = \log\left(1 + \frac{N}{f_t}\right), \quad (3)$$

where N is the total number of documents in the collection and f_t is the number of documents that contain term t.

This leads to a similarity measure between term set s and document d:

$$sim(s,d) = \frac{\sum_{t\in s}\left((1+\log(f_{d,t})) \times \log\left(1 + \frac{N}{f_t}\right)\right)}{\sqrt{|s| \times |d|}}. \quad (4)$$

We summarize the index pruning algorithm in Fig. 3. When a peer updates the global index, it browses the documents in its local index. When it meets a document d, it performs the following operations. For each in-document term set s with not more than $l_{\max}$ terms, the peer computes the similarity between s and d according to (4). In the formula, the values of f_t and N are computed using a randomized gossip algorithm. The values of $f_{d,t}$ and $|d|$ are obtained from the statistics of document d. The peer picks out the top $\lambda n \log(n)$ relevant term sets from d, where n is the number of distinct terms in d. The selected $s \rightarrow d$ pairs are inserted into the global index, while the values of $f_{d,t}$ and $|d|$ are piggybacked onto the $s \rightarrow d$ entry for global ranking during search.

```
Algorithm: Index Pruning
for ( each document $d$ in peer document set $D$ )
    for (each term-set $s$ with not more than $l_{max}$ terms )
        compute $f_t$ ($t \in s$) and $N$;
        compute $f_{d,t}$ and $|d|$ for document $d$;
        $s_{s,d} \leftarrow sim(s, d)$;
    end for
    rank $\{s_{s,d}\}$ in a decreasing order;
    insert the top $\lambda n \log(n)$ relavant $s \rightarrow d$ pairs with $f_{d,t}$ and $|d|$ into
    the global index;
end for
```

Fig. 3. Index pruning algorithm.

It is not difficult to see that precomputing the index makes a trade-off between storage/communication costs and the computational cost. The straightforward pruning for a document with n distinct terms need to perform sorting operations on a vector of $O(n^3)$ item sets, whose worst computation complexity is $O(n^3 \log n)$ by using a heap data structure. We analyze the WT10G collection [32], a large Web page collection provided by the US National Institute of Standards and Technology (NIST). The statistical result shows that a Web page has an average size of 5.91 KB (with the multimedia objects filtered), which typically includes several hundreds of distinct terms. The worst case may be for the document with a large number of terms, such as a dictionary. When implementing a real-world system, we can reduce the scale of term set combinations by using some heuristic schemes in the information retrieval area. For example, by the intuition that the desired semantically related term occurrences often occur closer to each other (in the same paragraph or ideally in the same sentence) than unrelated term occurrence [12], the *term distance* features can be utilized to effectively reduce the scale of the ranking vector.

3.2.3 Distributed Multidimensional Hashing Table

To support the distributed lookup, we build a novel multidimensional DHT structure based on Chord [29]. For simplicity, here, we set the value of $l_{\max}$ to 3. Given a term set $s = \{k_1, k_2, k_3\}$, we assign each keyword an identifier using a basic hash function $h(k)$ such as MD5-128. The hashed identifiers of keywords are combined together to identify the term set s:

$$Key(s) = h(k_1)h(k_2)h(k_3). \tag{5}$$

For the term set with less than three keywords, we combine the hash value of each keyword from the term set and fill the tail bits of *Key*(*s*) with "0." For example, for the set with two terms

$$s = \{k_1, k_2\}, Key(s) = h(k_1)h(k_2) \| \underbrace{0 \cdots 0}_{r\ bits},$$

where r is the length of the bit vector of the hashed value of $h(k)$. For a set with a single keyword, we use similar method.

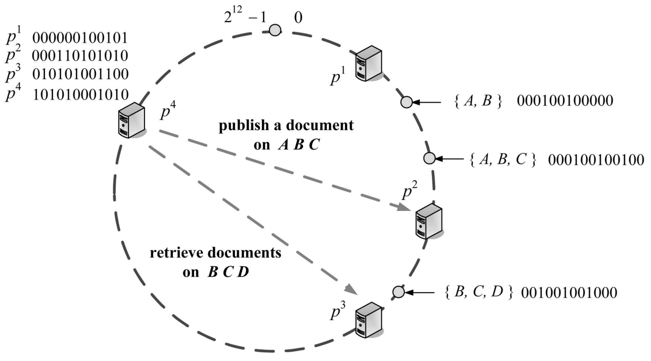

Fig. 4. Distributed index based on multidimensional hashing.

The identifier of node p is generated by hashing the node's IP address IP using another hash function $H(k)$ with the hashed value of $3 \times r$ bits, such as SHA-384 [1]:

$$Key(p) = H(IP). \tag{6}$$

TSS maps the keys of term sets to nodes using the Chord protocol. For keyword set s, suppose its combined identifier is $Key(s) = h(k_1)h(k_2)h(k_3)$, we assign it to the first node whose identifier is equal to or follows the identifier of s in the identifier space.

Fig. 4 depicts an example of the inverted index of TSS. We assume that each of terms A, B, C, and D separately has the 4-bit hashed identifiers 0001, 0010, 0100, and 1000. We also have four nodes p^1, p^2, p^3, and p^4, all with a 12-bit hashed identifier. The node p^4 publishes a document which contains keyword set $\{A, B, C\}$, and the combined hash key of the set is "000100100100." The node p^2 is responsible for the term set. Fig. 4 also shows the example that p^4 posts a query $\{B, C, D\}$ with the combined hash key "001001001000," and the query is routed to p^3.

Using the above distributed multidimensional hash table, we can distribute the term-set-based inverted index in the structure overlay. In TSS, the inverted index has posting list structure as follows:

$$Key(s) = \{(d, (f_1, f_2, f_3), |d|)\}, \tag{7}$$

where d is the global identifier for the relevant document. The integers f_1, f_2, *and* f_3 separately count the frequency of the keywords k_1, k_2, and k_3 of s appearing in document d, and $|d|$ denotes the length of document d. Built on top of the Chord protocol, TSS can easily look up the queries with no more than $l_{\max}$ keywords using the proposed term-set-based index. We extend our strategy for queries with more than $l_{\max}$ keywords in Section 3.3.

3.3 TSS Search

In TSS, a query is first inserted into the structured overlay. The Chord protocol routes the query to the responsible DHT node, which then ranks the list of documents for the query q using TFxIDF ranking scheme:

$$sim(q, d) = \frac{\sum_{t \in q} (1 + \log(f_{d,t})) \times \log\left(1 + \frac{N}{f_t}\right)}{\sqrt{|q| \times |d|}}, \tag{8}$$

where $|q|$ is the length of the query and returns the top-k results. During search, the lookup requires average

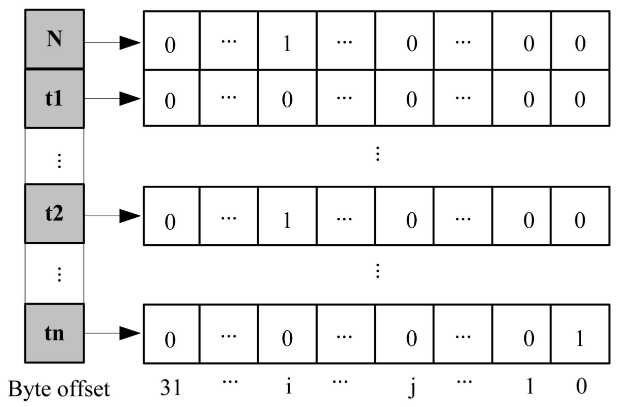

Fig. 5. Synopsis for computing term frequency.

$O(log(n))$ messages [29], where n is the number of peers in the network.

As we have seen that TSS can efficiently handle queries which have no more than $l_{\max}$ keywords. For the query which has more keywords, TSS first ranks the keywords according to their global frequency in an ascending order, and then, retrieves the set of URLs of the top-ranked documents which contain the first $l_{\max}$ terms. During retrieval, TSS piggybacks the ranking value with the other terms onto the retrieval message for a local search for the other terms. Only the documents with stable ranking values are returned after the local search.

TSS multidimensional index provides a cost-effective solution for multikeyword search, because it avoids the costly distributed intersection operations in wide-area networks performed by existing single-term index schemes [19], [23]. Note that a ranking scheme is critical for a P2P text retrieval system. Without ranking schemes, a complete list of results can possibly raise an unacceptably amount of communication cost roughly proportional to the size of the network, making the searching scheme unscalable.

3.4 Gathering Global Statistics

Within the structure of hybrid P2P network, we use a variant of the push synopsis gossip algorithm [20], [34] to gather the global statistics.

Specifically, the algorithm for gathering statistics has three main operations.

1. Initialization: At the initialization phase, each peer browses its local index and generates a local synopsis using the duplicated insensitive counting techniques pioneered by Flajolet [10]. The synopsis structure is designed as follows:

   A synopsis includes: a) a bit vector for counting N (the total number of documents in the P2P networks), denoted by *bitvect_N*; and b) an index for keywords with the format: $\{(t, bitvec\_f_t)\}$, where $bitvec\_f_t$ is a bit vector for the statistic of f_t (the number of documents containing the keyword t). Consider initializing *bitvec_N* as an example. A peer browses its local index. When it picks up a document first time, it does the coin flip experiment by flipping a coin up to k times and counts the times it sees heads before the first tail. It saves this count as r. Then, it sets the rth bit of *bitvect_N* to "1." We illustrate the data structure of the synopsis in Fig. 5 in detail, where the bit vector for N is denoted as *bitvect_N*, while the bit vector for t_i is denoted as $bitvec\_f_{t_i}$.
2. *Gossiping*: The synopses are disseminated among peers through the unstructured overlay using a randomized gossip algorithm [13]. During each round of gossip, each node chooses a random neighbor and sends the neighbor its synopsis. Fig. 6 shows an example of one round of gossip, where nodes A-P are all peers in TSS.
3. *Merging*: When a peer receives the synopsis from its neighbor, it merges the synopsis as follows:

   a. perform the bitwise-or operation on the *bitvec_N* from both synopses;
   b. browse the two synopses, for the keywords in both synopses, perform the bitwise-or operation on the pair of bit vectors for f_t; and
   c. for those keywords in the synopsis of the neighbor but not in the local synopsis, perform a union operation.

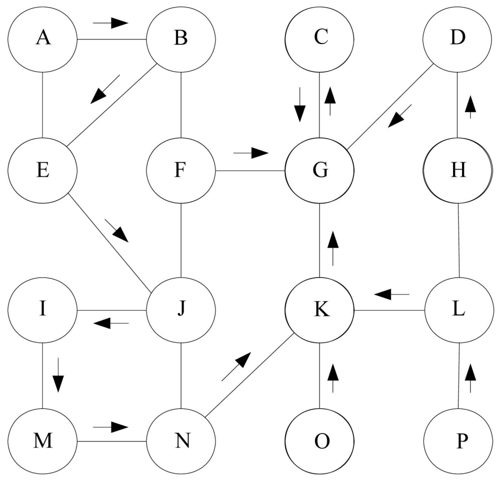

Fig. 6. An example of one round of gossip.

Fig. 7 shows the process at Peer B after it merges the synopsis that Peer A gossips to it. In this example, the keyword "protocol" is in the original synopsis of Peer A but not in that of Peer B. The algorithm merges the information by inserting the bit vector for keyword "protocol" from Peer A into the new version of the synopsis of Peer B. In the same example, the keyword "network" is in both original synopses. The algorithm performs a bitwise-or operation on both bit vectors for this same keyword.

After $O(\log\ n)$ rounds of gossip, every node gets the global statistics. If the first "1" bit counting from the left end of a bit vector is at the ith position, the estimated global statistical count associated with this bit vector is, with high probability, $2^i/0.77351$ [10].

4 Testbed

In this section, we first introduce the data set and query logs we use for the evaluation of the TSS performance. Then, we describe how we have collected the Gnutella traces for simulating the P2P network.

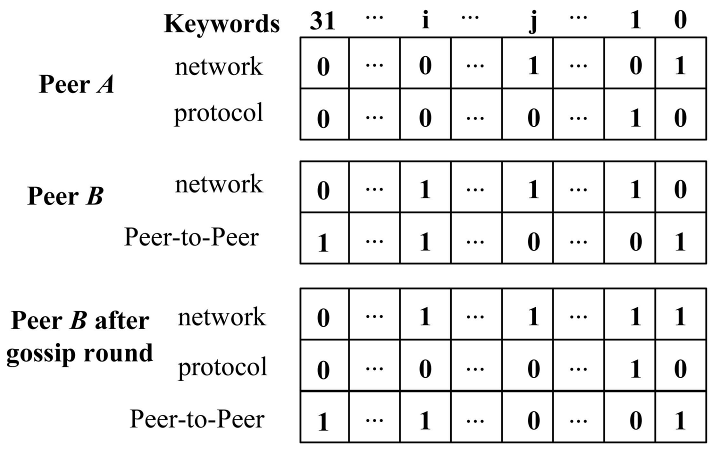
Keywords 31 ··· i ··· j ··· 1 0
Peer A
network 0 ··· 0 ··· 1 ··· 0 1
protocol 0 ··· 0 ··· 0 ··· 1 0
Peer B
network 0 ··· 1 ··· 1 ··· 1 0
Peer-to-Peer 1 ··· 1 ··· 0 ··· 0 1
Peer B after gossip round
network 0 ··· 1 ··· 1 ··· 1 1
protocol 0 ··· 0 ··· 0 ··· 1 0
Peer-to-Peer 1 ··· 1 ··· 0 ··· 0 1

Fig. 7. Synopsis merging during one round of gossip.

4.1 Data Set

We built one test data set for evaluating the performance of content-based search in P2P networks based on NIST WT10G Web corpus [32], a large test set widely used in text retrieval research. The data set has 10 gigabyte, 1.69 million Web page documents. The WT10g data were divided into 11,680 collections based on document URLs. Each collection, on average, has 144 documents with the smallest one having only five documents. The average size of documents is 5.91 KB. All data set was preprocessed with the Porter stemming algorithm to reduce words to their root (e.g., "running" becomes "run"), and common stop words such as "the," "and," etc., were removed from the data set [32].

4.2 Queries

For evaluation, we use two sets of queries. The first set is the queries provided with the NIST WT10G benchmark (TREC topic 501-550). We use the standard TREC benchmark to evaluate the NIST queries separately using TFxIDF and TSS. The benchmark includes the standard evaluation program, the NIST TREC queries, and the binary mapping of whether a document in WT10G collection is relevant to a particular query, where the relevance is judged by the NIST experts.

The second set of queries is the query logs we analyzed in Section 3.2.1. The system trace from a major commercial search engine and the WT10G Web collection are both quite representatives for real-world systems.

4.3 Trace Collection

Due to the large user community and the open architecture, we choose Gnutella [3] as the overlay network in the testbed. We have developed a crawler in Java based on the open-source Gnutella client, Limewire [3], to collect the topology information. We use the Gnutella topology trace we previously [5] collected to simulate a real P2P network. In our simulations, we randomly distribute the WT10G collections into the Gnutella peers.

5 Performance Evaluation

5.1 Communication Cost

To evaluate the search bandwidth cost, we compare the average number of postings transferred per TSS query during the query search process with that by using the scheme of single-term-based index. During searching, the scheme using single-term-based index transfers the posting lists for the query keywords among DHT nodes to achieve the intersection, while the TSS looks up the multidimensional index and returns all the matched results. Fig. 8 shows an enormous reduction of bandwidth consumption per query of the TSS compared to the single-term-based indexing. In the figure, we compare the average number of postings (the ID of document related to a key in DHT) transferred during the query search process.

We further compare the communication cost of TSS with those of the Bloom filter techniques presented in [23] and [19]. We use the communication cost of the single-term-based index scheme without Bloom filter techniques as the baseline and examine the improvements using different schemes. The results show that the BF-based scheme with the minimized false positives reduces the communication cost to 1/13 of the baseline, while the BF-based scheme with the optimal settings provided by Luo et al. [19] can achieve 1/18 of the baseline. Results show that TSS significantly reduces the communication cost to 1/70 of the baseline. Table 1 shows the performance improvement of different schemes.

In the above evaluation, to make the comparison fair, the simulator returns the list of all the matched documents for both TSS and the existing schemes. Thus, the potential number of results for a given query is roughly proportional to the number of documents in the network. The cost of

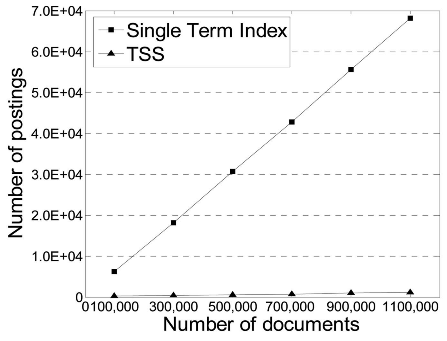

Fig. 8. Number of transferred postings per query.

TABLE 1
Cost Reduction TSS versus Existing Schemes

Technique	Cost compared to baseline
Single-term indexing	1.0
BF-based scheme with minimized false positives	1/13
BF-based scheme with optimal settings	1/18
TSS term-set index	1/70

returning all the matched results to the client will grow linearly with the size of the network. Single-term-index-based schemes, such as the scheme based on Bloom filters, can achieve a substantial constant factor improvement, but it does not eliminate the linear growth in cost. As described in Section 3.3, TSS indeed solves the problem of scalability by ranking the relevance of documents and returning the results with top ranking scores. In the following, we examine the quality of the returned results. We compare the retrieval accuracy of TSS with that of the centralized ranking scheme.

5.2 Search

To evaluate the search performance, three widely accepted metrics, *Recall*, *Precision*, and *F-Score* [32], are used:

$$Recall(q) = \frac{\#\text{of relevant docs. returned}}{\text{total }\#\text{ of relevant docs. in the network}}, \tag{9}$$

$$Precision(q) = \frac{\#\text{ of relevant docs. returned}}{\text{total }\#\text{ of docs. returned}}, \tag{10}$$

where q is the query posted by the user. *Recall* (q) captures the fraction of the returned relevant documents out of the total relevant documents available in the peer-to-peer system. *Precision* (q) captures the fraction of relevant documents in the returned results. We use the *F-Score* to measure the overall performance:

$$F - Score = \frac{2 \times Precision \times Recall}{Precision + Recall}. \tag{11}$$

The ranking accuracy is evaluated using the metric $p@k$ [32], proportion of the relevant results in top-k-ranked documents.

We experiment with the WT10G data set. All the metrics are measured according to the mean value of all the queries tested.

We conduct the following two experiments:

In the first experiment, we separately search the TREC queries provided by NIST on TSS P2P network and the centralized search engine. We compare the performance of TSS against that of the centralized search engine using the standard TREC benchmark in which the relevant documents for each query are labeled by NIST. We found that the traditional naive TFxIDF baseline is greatly dependent on the number of relevance mappings of the data set. We also found that WT10G test data set has a rather small fraction of relevance judgments, since it is expensive to make relevance judgments for so many documents per query. To make the evaluation clearer, in the first experiment, we use the WT10G data set without the unjudged documents. In the first step, TSS does not target at competing with more elaborated centralized retrieval algorithms.

Table 2 shows the retrieval accuracy in the standard TREC evaluation, where *Recall* and *Precision* are averaged over all queries (TREC topic $501 \sim 550$) provided by NIST with the WT10G data set. The results show that TSS has slightly lower *Recall* but higher *Precision* than centralized TFxIDF-based search scheme. The *R-Precision* [32] measures precision after R documents have been retrieved, where R is the total number of the relevant documents for a query. The result shows that TSS has better *R-Precision* than centralized TFxIDF scheme.

The TSS algorithm reduces the scale of term set for indexing. This indeed decreases the recall but lead to possible better precision. TSS makes a good trade-off between *Recall* and *Precision*. A higher Precision is commonly more desirable for a search engine user as iProspect's results, which have been shown in Fig. 2. The *F-Score* shows that the overall performance of TSS is comparable to that of the centralized search engine.

Fig. 9 further compares the $p@k$ ranking accuracy of TSS against the centralized TFxIDF baseline. The results show that when the scale of the index size is $n$log (n), the rank accuracy of TSS is comparable to the centralized TFxIDF baseline in different scale of top-k documents set. We change the scale of index size to $n(\log n)^2$ and find that the rank accuracy of TSS is slightly higher than that of the centralized TFxIDF baseline.

Fig. 10 plots average *Precision* over all test queries at each given recall level and reflects the search behavior of a particular run over the entire spectrum of recall. The result shows that TSS has better *Precision* at almost all given *Recall* levels.

TABLE 2
Accuracy Comparison TSS versus Centralized TFxIDF

Algorithms	Precision	Recall	F-Score	R-Precision
TFxIDF	47.27%	48.84%	0.4804	0.4647
TSS	51.64%	43.02%	0.4694	0.5066

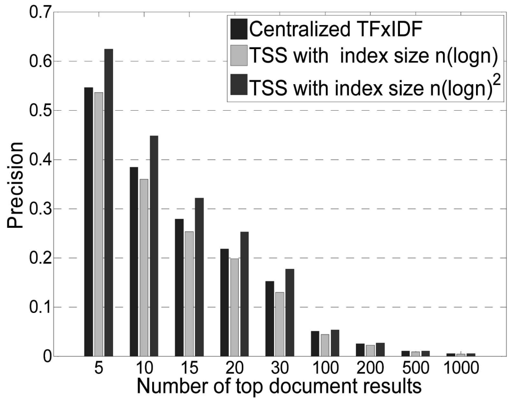

Fig. 9. Precision of top-ranked documents.

TABLE 3
Performance of TSS Retrieval

	Precision	Recall	F-Score
$k = 5$	67.06%	95.03%	0.7863
$k = 10$	70.41%	94.96%	0.8086
$k = 20$	73.25%	94.90%	0.8268
$k = 30$	74.54%	94.86%	0.8348
$k = 40$	75.35%	94.84%	0.8397
$k = 50$	75.90%	94.82%	0.8431

In the second experiment, we examined TSS using the query logs from the commercial search engine. We use the search results by TFxIDF from the centralized single collection of WT10G as the baseline, and measured how well the P2P network could reproduce the baseline. Accuracy is measured with modified forms of set-based *Recall* and *Precision*, defined as follows [18]:

$$Recall(q) = \frac{|r|}{|A|}, \quad (12)$$

$$Precision(q) = \frac{|r|}{|R|}, \quad (13)$$

where R is the set of the documents returned by TSS P2P network, A is the set of top-ranked documents returned by searching using the centralized TFxIDF scheme on the WT10G collection, and r is the intersection of R and A.

Since the WT10G collection and the query logs are from different sources, there is no way to guarantee that there are relevant documents in the WT10G collection for all these queries. To cope with the problem, we filtered the queries which have no results returned by TFxIDF. By doing so, we could guarantee that all the considered queries are from the source of the WT10G collection. Note that in the experiment, we use the results directly retrieved in TFxIDF scheme as the baseline. The returned results by using centralized TFxIDF are viewed as relevant documents. Since the goal of this experiment is to evaluate how well the TSS could reproduce the TFxIDF scheme, filtering the queries for which TFxIDF returns no results indeed provides a fair comparison.

The results in Table 3 show that TSS has good overall search performance. When the value of k increases, the *Precision* of TSS increases while the *Recall* decreases slightly.

Fig. 11 plots the precision of top-ranked document of TSS. In the experiment, the top 100 results by TFxIDF scheme are used as relevant documents for the queries from the log. The results show that the ranking accuracy of TSS is quite acceptable.

Fig. 12 presents the overlap of the top-k documents returned by the TSS and the centralized scheme. We focus on the high-end ranking because users are often interested only in the top results. The comparison shows significant overlap between the returned result sets. The results also show that when the index size changed from $n\log(n)$ to $n(\log n)^2$ the overlap changes slightly.

In Figs. 13 and 14, we examine the ranking accuracy in different scale of index size. We change the scale of index size $\lambda(n\log n)$ in the TSS pruning algorithm by adjusting the value of λ from 0.05 to 1.5. The results show that when the index cost increases linearly, the ranking accuracy of TSS increases. Fig. 14 also shows a visible inflexion near the point $\lambda = 0.3$. When λ increases after this point, increasing the index size nearly does not increase the ranking accuracy. In practice, different systems may achieve different λ settings following this method. In this work,

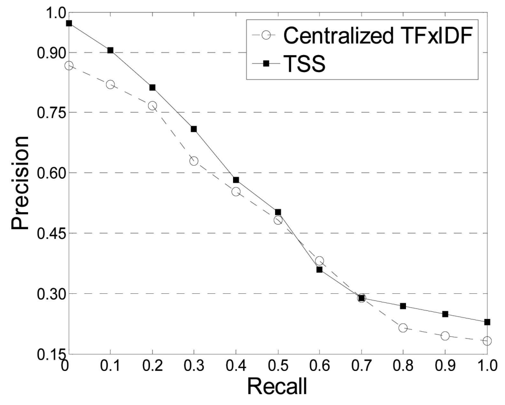

Fig. 10. Precision recall curve.

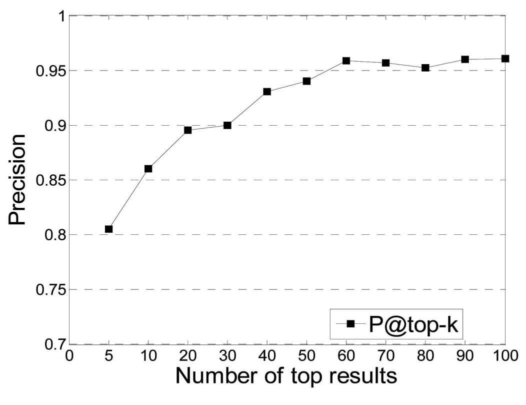

Fig. 11. Precision of top-ranked results.

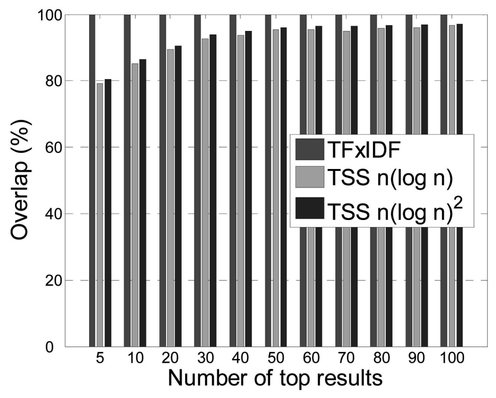

Fig. 12. Overlap with top-k results.

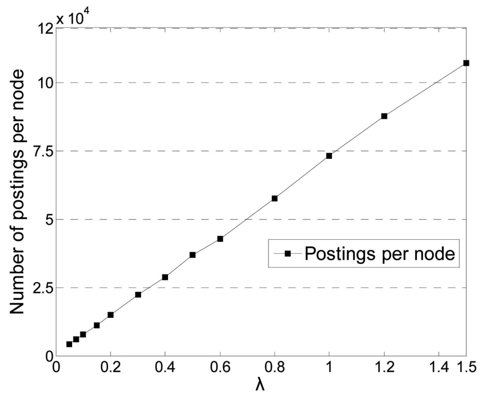

Fig. 14. Change of index size.

we draw the conclusion that TSS can achieve comparable search performance against a centralized search scheme with the index cost of $O(n\log(n))$.

5.3 Maintenance

Compared with single-term index scheme, the downside to term-set-index-based scheme is the index size [11]. Let it be supposed that a node in the system contains m documents each with n terms. Each node join/leave event in TSS will incur $O(mn\log n)$ DHT updating operations, while we need $O(mn)$ by using the single-term-based index. To quantify the index maintaining cost for TSS indexing, we investigate the average number of postings inserted into a structured superpeer in our simulation, and compare it to the size of the single-term index scheme. In Fig. 15, we see that a node stores more postings associated with TSS compared to single-term indexing.

As the above results show, each TSS query operation can gain a lower bound of $70\times$ improvement (achieved when it returns the full list of matching documents) in traffic saving. The overall benefit of TSS design depends on the *querying frequency/updating frequency* ratio in the search network. The cost can be amortized over all the queries in the system. As the search load increases, the proportion of the index updating cost to the overhead indeed decreases. Although it is difficult, if not impossible, to achieve such statistics from real-world systems, we believe in search-centric systems, such as Web search engine [4], [6] and large-scale digital library systems, where queries are much more frequent than updating, the global indexing scheme is a good trade-off. On the other hand, compared with centralized Web search engine, the index maintaining cost associated with updating can be offset by eliminating the need to repeatedly crawl the content in the network into a centralized repository.

5.4 Gossiping Cost Analysis

Another cost of TSS is the storage cost for the synopsis used for the gossip algorithm. In TSS, the bit vector for each unique term takes 4 bytes. The average word length is about five. Each term will take 9 bytes in the synopsis. Considering the worst case, the Oxford English Dictionary Second Edition contains about 615,100 words. Thus, the synopsis for all the terms takes 5.28M at most. By using the text compression techniques, such as the Burrows-Wheeler Transform, which has a measured compression ratios of 2.95, TSS can reduce the storage size for the synopsis to about 1.78 MB. Considering that the median object size in P2P systems, such as KaZaA and Gnutella, is about 4 MB [25], we believe that a synopsis of 1.78 MB is acceptable.

The gossip algorithm also adds a bandwidth overhead to the system. However, the price to pay for collecting the global statistics is quite acceptable for a real-world system due to the fact that the global statistics in the networks, such as Web and large-scale digital library systems, are slowly changing, while the robust gossip scheme causes the computation of aggregated information to converge

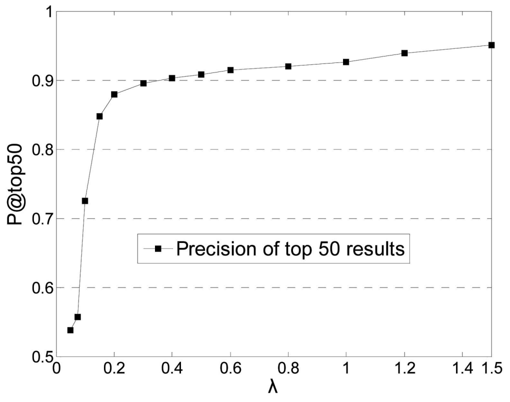

Fig. 13. Change of ranking accuracy.

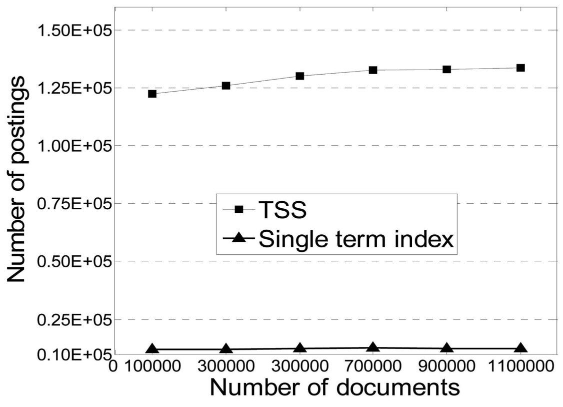

Fig. 15. Inserted posting per peer.

exponentially. Therefore, infrequent and approximate computation of these statistics is sufficient in TSS.

6 Conclusions and Future Work

We propose TSS scheme for multiterm search in P2Ps in which an index pruning algorithm is introduced to reduce bandwidth consumption. We conduct comprehensive trace-driven simulations based on the TREC WT10G data and the query log of a commercial search engine. Results show that our method is able to provide comparable performance and ranking accuracy with a centralized search engine with the index cost of $O(n \log(n))$. Motivated by the fact that queries can reflect real information requirements of users, we will use query-driven adaptive method to further improve the recall of TSS search in our future work.

Acknowledgments

This work was funded by National Natural Science Foundation of China (NSFC)/Research Grants Council (RGC) under grant 60731160630, Hong Kong RGC grants N_HKUST614/07, and HKUST Nansha Research Fund NRC06/07.EG01.

References

[1] FIPS 180-1, Secure Hash Standard, Department of Commerce/NIST, National Technical Information Service, 1995.
[2] Iprospect, http://www.iprospect.com/, 2009.
[3] Limewire, http://www.limewire.com, 2009.
[4] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer, "P2P Content Search: Give the Web Back to the People," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2006.
[5] H. Chen, H. Jin, Y. Liu, and L.M. Ni, "Difficulty-Aware Hybrid Search in Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 1, pp. 71-82, Jan. 2009.
[6] H. Chen, H. Jin, J. Wang, L. Chen, Y. Liu, and L.M. Ni, "Efficient Multi-Keyword Search over P2P Web," *Proc. Int'l World Wide Web Conf. (WWW),* 2008.
[7] E. Cohen and S. Shenker, "Replication Strategies in Unstructured Peer-to-peer Networks," *Proc. ACM SIGCOMM,* 2002.
[8] 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," *Proc. IEEE Int'l Symp. High Performance Distributed Computing (HPDC),* 2003.
[9] B. Fan, J.C.S. Lui, and D.-M. Chiu, "The Design Trade-Offs of BitTorren-Like File Sharing Protocols," *IEEE/ACM Trans. Networking,* vol. 17, no. 2, pp. 365-376, Apr. 2009.
[10] P. Flajolet and G.N. Martin, "Probabilistic Counting Algorithms for Data Base Applications," *J. Computer and System Sciences,* vol. 31, pp. 182-209, 1985.
[11] O.D. Gnawali, "A Keyword-Set Search System for Peer-to-Peer Networks," Master's thesis, MIT, 2002.
[12] D. Hawking and P. Thistlewaite, "Relevance Weighting Using Distance between Term Occurrences," technical report, Dept. of Computer Science, The Australian Nat'l Univ., 1996.
[13] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-Based Computation of Aggregate Information," *Proc. IEEE Symp. Foundations of Computer Science (FOCS),* 2003.
[14] D. Li, J. Cao, X. Lu, and K. Chen, "Efficient Range Query Processing in Peer-to-Peer Systems," *IEEE Trans. Knowledge and Data Eng.,* vol. 21, no. 1, pp. 78-91, Jan. 2009.
[15] J. Li, B.T. Loo, J.M. Hellerstein, M.F. Kaashoek, D.R. Karger, and R. Morris, "On the Feasibility of Peer-to-Peer Web Indexing and Search," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2003.
[16] M. Li, W.-C. Lee, A. Sivasubramaniam, and J. Zhao, "SSW: A Small World Based Overlay for Peer-to-Peer Search," *IEEE Trans. Distributed and Parallel Systems,* vol. 19, no. 6, pp. 735-749, June 2008.
[17] X. Lou and K. Hwang, "Collusive Piracy Prevention in P2P Content Delivery Networks," *IEEE Trans. Computers,* vol. 58, no. 7, pp. 970-983, July 2009.
[18] J. Lu and J. Callan, "Content-Based Retrieval in Hybrid Peer-to-Peer Networks," *Proc. Int'l Conf. Information and Knowledge Management (CIKM),* 2003.
[19] X. Luo, Z. Qin, J. Han, and H. Chen, "DHT-Assisted Probabilistic and Exhaustive Search in Unstructured P2P Networks," *Proc. IEEE Int'l Symp. Parallel and Distributed Processing (IPDPS),* 2008.
[20] S. Nath, P.B. Gibbons, S. Seshan, and Z.R. Anderson, "Synopsis Diffusion for Robust Aggregation in Sensor Networks," *Proc. ACM Int'l Conf. Embedded Networked Sensor Systems (SenSys),* 2004.
[21] M. Plechawski, P. Pyszlak, B.R. Schnizler, R. Siebes, S. Staab, and C. Tempich, "Bibster—A Semantics-Based Bibliographic Peer-to-Peer System," *Proc. Int'l Semantic Web Conf. (ISWC),* 2004.
[22] K.P.N. Puttaswamy, H. Zheng, and B.Y. Zhao, "Securing Structured Overlays against Identity Attacks," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 10, pp. 1487-1498, Oct. 2009.
[23] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching," *Proc. Int'l Conf. Middleware,* 2003.
[24] G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," *Information Processing and Management,* vol. 24, pp. 513-523, 1988.
[25] S. Saroiu, P.K. Gummadi, R.J. Dunn, S.D. Gribble, and H.M. Levy, "An Analysis of Internet Content Delivery Systems," *Proc. Symp. Operating Systems Design and Implementation (OSDI),* 2002.
[26] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 6, pp. 849-862, June 2007.
[27] H.T. Shen, Y.F. Shu, and B. Yu, "Efficient Semantic-Based Content Search in P2P Network," *IEEE Trans. Knowledge and Data Eng.,* vol. 16, no. 7, pp. 813-826, July 2004.
[28] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," *Proc. IEEE INFOCOM,* 2003.
[29] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," *IEEE/ACM Trans. Networking,* vol. 11, no. 1, pp. 17-32, Feb. 2003.
[30] C. Tang and S. Dwarkadas, "Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval," *Proc. USENIX Symp. Networked Systems Design and Implementation (NSDI),* 2004.
[31] W.W. Terpstra, J. Kangasharju, C. Leng, and A.P. Buchmann, "Bubblestorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search," *Proc. ACM SIGCOMM,* 2007.
[32] E.M. Voorhees, "Overview of TREC-2007," *Proc. 16th Text Retrieval Conf. (TREC-9),* 2007.
[33] J.R. Wen, J.Y. Nie, and H.J. Zhang, "Query Clustering Using User Logs," *ACM Trans. Information Systems,* vol. 20, no. 1, pp. 59-81, 2002.
[34] M. Zaharia and S. Keshav, "Gossip-Based Search Selection in Hybrid Peer-to-Peer Networks," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS),* 2006.

Hanhua Chen is currently working toward the PhD degree at the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China. His research interests include peer-to-peer computing, information retrieval, and wireless sensor networks. He was awarded the Microsoft Fellowship in 2005. He is a member of the IEEE and the IEEE Computer Society.

Jun Yan received the PhD degree in digital signal processing and pattern recognition from the Department of Information Science, School of Mathematical Science, Peking University, P.R. China. He has been a research associate at CBI, HMS, and Harvard, Cambridge, Massachusetts, since 2005. In 2006, he joined as an associate researcher at Microsoft Research Asia (MSRA), where he is currently working in the machine learning group. His research interests are online advertising, behavioral targeting, data preprocessing, and information retrieval. So far, he has been a reviewer of several top journals such as *Transactions on Knowledge and Data Engineering*, *Transactions on Pattern Analysis Machine Intelligence*, etc., and PC members of several conferences. He is a member of the IEEE.

Hai Jin received the PhD degree in computer engineering in 1994 from Huazhong University of Science and Technology (HUST), China, where he is currently the Cheung Kong Professor and the dean of the School of Computer Science and Technology. In 1996, he was awarded a German Academic Exchange Service Fellowship to visit the Technical University of Chemnitz in Germany. He worked at the University of Hong Kong between 1998 and 2000, and as a visiting scholar at the University of Southern California between 1999 and 2000. He was awarded the Distinguished Young Scholar Award from the National Science Foundation of China in 2001. He is the chief scientist of ChinaGrid, the largest grid computing project in China. He is the member of Grid Forum Steering Group (GFSG). He has coauthored 15 books and published more than 400 research papers. His research interests include computer architecture, virtualization technology, cluster computing and grid computing, peer-to-peer computing, network storage, and network security. He is a senior member of the IEEE and a member of the ACM.

Yunhao Liu (SM '06) received the BS degree from the Automation Department, Tsinghua University, China, in 1995, the MA degree from Beijing Foreign Studies University, China, in 1997, and the MS and PhD degrees in computer science and engineering from Michigan State University in 2003 and 2004, respectively. He is now an associate professor and the postgraduate director in the Department of Computer Science and Engineering at Hong Kong University of Science and Technology. His research interests include peer-to-peer computing, pervasive computing, and sensor networks. He is a senior member of the IEEE and a member of the ACM.

Lionel M. Ni received the PhD degree in electrical and computer engineering from Purdue University, West Lafayette, Indiana, in 1980. He is a chair professor in the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology (HKUST). He is also the director of the HKUST China Ministry of Education/Microsoft Research Asia IT Key Lab and the director of the HKUST Digital Life Research Center. He has chaired many professional conferences and has received a number of awards for authoring outstanding papers. He is a fellow of IEEE.