Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

KRBKSS: a keyword relationship based keyword-set search system for peer-to-peer networks*

ZHANG Liang (张 亮)†, ZOU Fu-tai (邹福泰), MA Fan-yuan (马范援)

(*Department of Computer Science & Engineering, Shanghai Jiaotong University, Shanghai 200030, China*)

†E-mail: zhangliang@cs.sjtu.edu.cn

Received July 4, 2004; revision accepted Oct. 9, 2004

Abstract: Distributed inverted index technology is used in many peer-to-peer (P2P) systems to help find rapidly document in which a given word appears. Distributed inverted index by keywords may incur significant bandwidth for executing more complicated search queries such as multiple-attribute queries. In order to reduce query overhead, KSS (keyword-set search) by Gnawali partitions the index by a set of keywords. However, a KSS index is considerably larger than a standard inverted index, since there are more word sets than there are individual words. And the insert overhead and storage overhead are obviously unacceptable for full-text search on a collection of documents even if KSS uses the distance window technology. In this paper, we extract the relationship information between query keywords from websites' queries logs to improve performance of KSS system. Experiments results clearly demonstrated that the improved keyword-set search system based on keywords relationship (KRBKSS) is more efficient than KSS index in insert overhead and storage overhead, and a standard inverted index in terms of communication costs for query.

Key words: Peer-to-peer (P2P), Keyword-set search (KSS), Keyword relationship
doi:10.1631/jzus.2005.A0577 **Document code:** A **CLC number:** TP393

INTRODUCTION

Peer-to-peer (P2P) systems are now one of the most prevalent Internet distributed applications due to their greater scalability, fault-tolerance, and self-organizing nature. This trend was triggered in 1999 by Napster (2001), a centralized architecture with a central directory server that offers an index to locate data items. P2P system can be divided into two classes. One class is an unstructured P2P system, where the overlay topology is formed in accordance with some loose rules. Popular P2P file-sharing systems like Gnutella (2001) and Kazaa (2001) use unstructured network designs. These networks typically adopt flooding-based search techniques to locate files, contacting all accessible nodes in TTL hops. Their basic characteristics are simplicity and the huge overhead they produce by contacting many nodes. Flooding-based techniques are effective for locating highly replicated items, but they are poorly suited for locating rare items.

The other class of the decentralized architectures is a structured P2P system commonly referred to as Distributed Hash Tables (DHTs) (Zhao *et al.*, 2000), where the overlay topology is tightly controlled. Distributed inverted index technology is used in many P2P systems to help find rapidly document in which a given word appears. The peers that are required to store or index data items are precisely determined based on some hashing algorithms. This tightly controlled structure enables forwarding of queries deterministically, and achieves very effective content location. However, distributed inverted index by keywords may incur significant bandwidth for executing more complicated search queries such as multiple-attribute queries. This is unacceptably large bandwidth for query in a P2P system because bandwidth available to most nodes in the Internet is rather

*Project supported by the National Natural Science Foundation of China (No. 60221120145) and Science & Technology Committee of Shanghai Municipality Key Project (No. 02DJ14045), China

small. In order to reduce query overhead, KSS (Gnawali, 2002) partitions the index by a set of keywords. A KSS index is considerably larger than a standard inverted index, since there are more word sets than there are individual words. And insert overhead for KSS grows exponentially with the size of the keyword-set while query overhead is reduced to the result of a query as no intermediate lists are transferred across the network for the join operation. The insert overhead and storage overhead are obviously unacceptable for full-text search on a collection of documents even if KSS makes use of the distance window technology (Gnawali, 2002).

Our work is motivated by user query keywords model to improve the insert and storage efficiency of KSS. In this work, we exploit the observation that in typical KSS many keywords pairs mapped to the nodes in the network are not or seldom used in users' actual queries because in KSS the relationship between query keywords is not taken into consideration. We aim to extract the relationship information between query keywords from users' queries logs to improve the performance of KSS system.

RELATED STUDIES

P2P networks have been studied intensively in the last few years. A number of systems and algorithms have recently been developed that support P2P search. The search index in Napster (2001) is centralized; the storage and serving of files is distributed. From the file sharing and downloading perspective, Napster is a P2P system. However, from the indexing perspective, Napster is a centralized system. The original Gnutella (2001) algorithm uses flooding for object discovery and contacts all accessible nodes within the TTL value. Although it is simple and manages to discover the maximum number of objects in that region, the approach does not scale, producing huge overhead to large numbers of peers. In Gnutella2 (Stokes, 2002), when a super-peer (or hub) receives a query from a leaf, it forwards it to its relevant leaves and also to its neighboring hubs. These hubs process the query locally and forward it to their relevant leaves. No other nodes are visited with this algorithm. Neighboring hubs regularly exchange local repository tables to filter out unnecessary traffic between them. Modified-BFS and Intelligent-BFS (Kalogeraki *et al*., 2002) and APS (Tsoumakos and Roussopoulos, 2003) are variations of the flooding scheme, with peers randomly or intelligently choosing only a certain proportion of their neighbors to forward the query to. These algorithms certainly reduce the average message production compared to previous methods, but the accuracy and efficiency are still a question. In Local Indices (Yang and Garcia-Molina, 2002), each node indexes the files stored at all nodes within a certain radius r and can answer queries on behalf of all of them (Kalogeraki *et al*., 2002). A search is performed in a BFS-like manner, but only nodes accessible from the requester at certain depths process the query. The method's accuracy and hits are very high, since each contacted node indexes many peers. On the other hand, message production is comparable to the flooding scheme, although the processing time is much smaller because not every node processes the query.

These strategies above are for searching in unstructured P2P networks. Next, we will discuss searching schemes in structured P2P networks. Recent research efforts in structured P2P seek to provide the illusion of having a global hash table shared by all members of the community. Frameworks like Tapestry (Zhao *et al.*, 2000), Pastry (Rowstron and Druschel, 2001), Chord (Stoica *et al.*, 2001) and CAN use different techniques to spread (key, value) pairs across the community and to route queries from any member to where the data is stored. Although this distributed hash structure could be used to create an inverted index, it would not be so efficient. In all these systems, there is a time cost needed to contact the right node in order to publish a single key. If we want to share a document's content then we need to publish every unique word contained in it. Besides this, the communication cost of performing a multi-keyword query is very high.

The keyword-set search system (KSS) proposed by Gnawali *et al.*(2002) is a P2P keyword search system that uses a distributed inverted index partitioned across the nodes in the network by a set of keywords. In KSS, the index is partitioned by sets of keywords. KSS builds a distributed inverted index that maps each set of keywords to a list of all the documents that contain the words in the keyword-set. When a user issues a query, the keywords in the query

are divided into sets of keywords. The document list for each set of keywords is then fetched from the network. Thus search using KSS results in a smaller query time overhead. The main benefit of KSS is the low communication cost of performing a multi-keyword query. In a typical KSS setting, two-word queries involve no significant communication, since they are processed wholly within the node responsible for that keyword set. Queries with more than two words require that document lists be communicated and joined; but these lists are smaller than in a standard inverted index, because the number of documents that contain a set of words is smaller than the number of documents that contain each word alone. Single-word queries are processed using a standard single-word inverted index, and require no significant communication. Thus KSS outperforms the standard distributed inverted scheme at the expense of storage overhead. Although the query overhead for the target application is reduced, insert overhead for KSS grows exponentially as the size of the keyword-set used to generate the keys for index entries. That is, KSS causes much more insert overhead than the traditional single word publication.

KRBKSS OVERVIEW

An overview of the constructing KRBKSS process is given in Fig.1, and described in detail in the following sections. According to Fig.1, constructing the KRBKSS system is composed of two steps: (1) Use of KWRDA to discover relationship between query keywords from query logs, which can be obtained from WWW or FTP search websites; (2) In KRBKSS, just map these edges output by KWRDA in Step (1) instead of all keywords pairs in keywords-set search system (KSS).

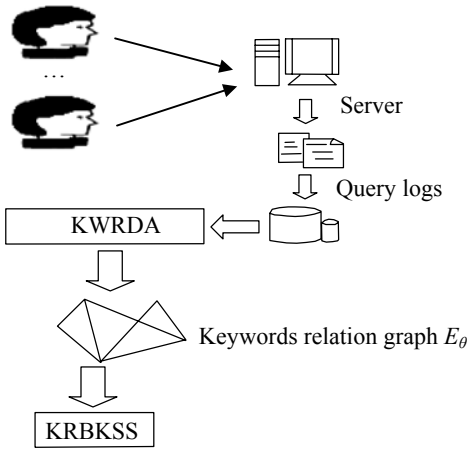

Fig.1 The process of constructing KRBKSS

KWRDA ALGORITHM

In order to discover the relationship between keywords queried by users, which can be useful in distributed inverted indexing of document, we proposed the KWRDA algorithm. The Keywords Relationship Discovery Algorithm (KWRDA) takes a server access log as input and maps it into a graph which expresses the relationship between keywords. The algorithm has three basic steps:

(1) Construct a directed graph $G(A,E)$ according to the query logs

The set of vertices A in graph G(A,E) corresponds to the search terms used in the user queries. The set of edges E corresponds to search terms co-occurrence as observed in the user queries. $E=\{e|weight(e)>0\}$. Since the graph $G(A,E)$ is a directed graph, $E_{A1\to A2}$ and $E_{A2\to A1}$ should be distinguished form each other. The weight of a directed edge is defined as follows:

$$weight(E_{A1\to A2})=\frac{freq(A1\cap A2)}{freq(A1)}$$

$$weight(E_{A2\to A1})=\frac{freq(A1\cap A2)}{freq(A2)}$$

where $A1$ and $A2$ are vertices in set A. The $freq(X)$ represents the frequency that search term X occurs in users' query. For instance, if a query procedure contains the search terms "P2P" and "search" the frequency of the relevant vertices is added one respectively. The weights on the directed edge (P2P $\to$search) are computed as the normalized frequencies by dividing them with the occurrence frequencies of the "P2P" vertices. The effect of the normalization is to remove the bias for characteristics that appear very often in all users.

(2) Pruning the graph to $G(A,E)|_\theta$ according to a given connectivity threshold θ

As the connectivity of the resulting graph G is

usually high, we use a connectivity threshold, aiming at reduction of the number of edges in the graph. The connectivity threshold represents the minimum weight allowed for the edge's existence. When this threshold is high the graph will be sparse and when the threshold is lower the graph will be dense.

In graph $G(A,E|_\theta)$, the set of vertices A in graph $G(A,E|_\theta)$ is same to the set of vertices A in graph $G(A,E)$, which corresponds to the search terms used in the user queries. However, the set of edges $E|_\theta$ corresponds to search terms co-occurrence as observed in the user queries.

$$E|_\theta=\{e|e\in E \text{ and } weight(e)\geq\theta\}$$

It is obvious that different connectivity threshold θ may output different $E|_\theta$. The larger connectivity threshold θ is, the sparser the graph is.

(3) Output $E|_\theta$

This step outputs the vertex pairs corresponding to $E|_\theta$, which expresses the co-occurrence relationship of query keywords.

SYSTEM ARCHITECTURE

KRBKSS can be implemented in any P2P platform that supports Distributed Hash (DHash) Table interface such as Chord, CAN and Tapestry. In this section, like KSS, we will describe an example system using the Chord system (Fig.2).

The Chord layer in the system architecture provides for support for one operation: given a key, it maps the key onto a node. Each node in the system that uses consistent hashing maintains a data structure called finger table, by which the query is routed closer and closer to the target nodes or its successor.

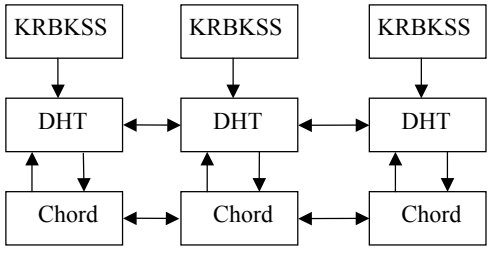

Fig.2 KRBKSS system architecture. Each peer has KRBKSS, DHash and Chord layers

The DHash layer implements a distributed hash table for the Chord system. Dhash provides a simple get-put API that allows a P2P application to put a data item in the nodes in the P2P network and get data given their ID from the network.

KRBKSS provides two operations to client application. One is to insert document operation, which extracts the keywords from the document by means of the relationship between keywords, and generates index entries, and stores them in the network. The other is to find the document list for disjoint set of keywords in the query, and return the intersected list of documents.

KRBKSS PUBLICATION AND QUERY

KRBKSS works as follows: When a user shares a file, KRBKSS uses the keywords relationship to generate the index entries for the file, hashes the keyword-set to form the key for the index entry, maps the keys to the nodes in the network using consistent hashing and Chord.

The algorithm that KRBKSS creates the index is as follows:

```
word[0…n]=meta-data field
for (i=0; i<n; i++)
      for (j=i+1; j<n; j++)
            if ((word[i], word[j])∈E|θ)
                  set_add(keywords,
                        concat(sort(word[i], word[j])));
for (i=0; i<keywords.size; i++)
      push(index_entries, <hash(keywords[i]),
       documentID>);
```

For example, let A, B, C, and D be the words in a document identified by docID. KSS creates index entries for each of the six combinations (AB, AC, AD, BC, BD, CD). For a set of size two, $C(n, 2)$ gives the number of unique entries in the cross product of sets of n unique keywords. However, we found that the keywords pairs AC, AD, BC, BD, CD have never been or are seldom used as queries pairs by users. Thus in our method KRBKSS only the keywords pairs AB is mapped onto the nodes in the P2P network. And the multi-keywords query algorithm in the KRBKSS system works as follows:

Input: $Q=\{k_1, k_2, \ldots, k_n\}$
Output: a list of documents that contain
all the keywords in the query
$Q'=Q$
While $Q'\neq\varnothing$ or $|Q'|\neq 1$
find the most related two query
keywords km and kp from Q' by E_θ
finding the nodes storing the index entries for the
two keywords and fetches the list
$Q'=Q'-\{km, kp\}$
intersect the results and output

EXPERIMENTS

In this section, we evaluate KRBKSS algorithm by simulation. In order to analyze KRBKSS costs and efficiency for full-text search, we develop a web crawler that takes the web pages www.edu.cn and www.sohu.com as seeds and downloaded the text and HTML files recursively. Our crawler downloaded 42238 HTML and text pages that occupied 492 MB of disk space. In order to compare KDBKSS with existing KSS algorithms, we implemented KSS and standard inverted index. For the simulation we deployed 1800 nodes running on 12 personal computers in a 100 M LAN, each of which has a 1.7 GHz processor with 512 MB RAM running Linux AS 3.0.

In order to find the relationship between query keywords, we used the query logs of the FTP search website bingle.pku.edu.cn from Dec 1, 2002 to Dec 31, 2002.

We developed a scalable system iExtra, which is implemented purely in Java. In the preprocess phase, iExtra parses the HTML pages and removes the invalid characters. After parsing, Chinese paragraphs are extracted and segmented through Maximized-Matched Chinese word segmentation algorithm, and the resulted Chinese words were encoded with a unique ASCII string. We also selected a list of stopwords for filtering the English as well as Chinese stopwords.

We simulated inserting and querying of a document using KRBKSS. Next we ran the KRBKSS algorithm on each text file to create index entries and published them to corresponding virtual peers. We evaluated these algorithms by insert overhead and query overhead at different connectivity threshold θ.

Insert overhead

Insert overhead is the number of bytes transmitted when a document is inserted in the system. When a user asks the KRBKSS system to share a file, the system generates index entries which are inserted in the distributed index. Unlike KSS, in which if we generate index entries for a document with n keywords for typical keyword-pair scheme the overhead required is bounded by $C(n; 2)$, KRBKSS only generates small index entries which results in a small insert overhead.

Fig.3 gives the curves of number of index entities generated vs number of words in a document using the standard inverted indexing scheme, KRBKSS with θ=0, KRBKSS with θ=0.05 and KSS with window size of ten. Fig.4 presents a distribution of the number of index entries generated when each document (page) is inserted in the system using KSS with window size of five, KRBKSS with θ=0 and using the standard inverted indexing scheme. Fig.3 and Fig.4 show that the insert overhead for KRBKSS is much lower than that for KSS and is a little higher than that of the standard inverted index scheme.

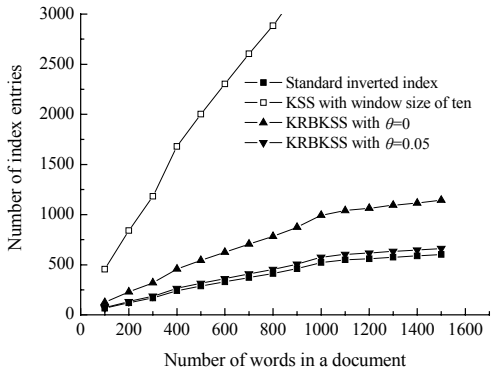

Fig.3 Number of index entities generated vs number of words in a document using the standard inverted indexing scheme, KRBKSS with θ=0, KRBKSS with θ=0.05 and KSS with window size of ten

Query overhead

Query overhead is a measure of bytes transmitted when a user searches for a file in the system. As we know, the overhead to send the intermediate result list in the system from one host to another is the main part of the query overhead.

Fig.5 gives mean data transferred in KB when searching using the standard inverted index with Bloom Filter, the standard inverted index without

Bloom Filter, KSS with window size of 5, KRBKSS with θ=0, KRBKSS with θ=0.05, for a range of query words. Fig.5 shows that the query overhead for KRBKSS is much lower than that of the standard inverted index scheme, with or without Bloom Filter, and is a little higher than that for KRBKSS.

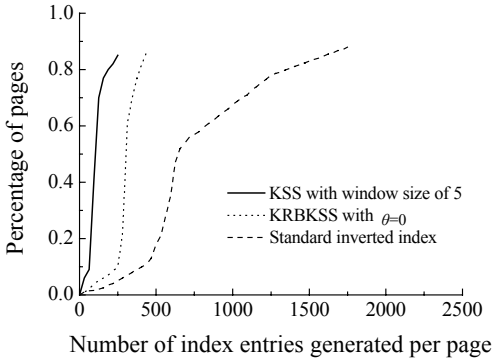

Fig.4 Cumulative distribution of the number of documents for which the given number of index entries in *x*-axis are generated using the standard inverted indexing scheme, KRBKSS and KSS with window size of five

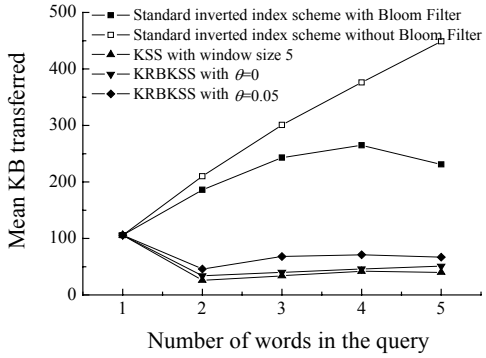

Fig.5 Mean data transferred in KB when searching using the standard inverted index with Bloom Filter, the standard inverted index without Bloom Filter, KSS with window size of 5, KRBKSS with θ=0, KRBKSS with θ=0.05, for a range of query words

CONCLUSION

In this work, we exploit the relationship between query keywords, which can be extracted from users' queries logs, to improve the performance of KSS system. We present the design and implementation of KRBKSS, a keyword relationship based keyword-set search system that was implemented using Java. The main idea of this paper is as follows: At first, we may make use of KWRDA to find the relationship between query keywords from query logs obtained from WWW or FTP search websites such as bingle.pku.edu.cn, sheenk.com, and so on. Next, in Step (1) in KRBKSS, we just map these edges output by KWRDA instead of all keywords pair in keywords set search system, which may decrease insert overhead and storage overhead dramatically compared to KSS. Experiments results clearly demonstrated that KRBKSS index is more efficient than KSS index in insert overhead and storage overhead, and more efficient than a standard inverted index in terms of communication costs for query. In a forthcoming paper, the authors will show how to automatically or adaptively determine a proper parameter θ that can result in good tradeoff between insert overhead and query overhead.

References

Gnawali, O.D., 2002. A Keyword-set Search System for Peer-to-Peer Networks. Master's Thesis, MIT's Thesis Lib.

Gnutella, 2001. http://gnutella.wego.com.

Kalogeraki, V., Gunopulos, D., Zeinalipour-Yazti, D., 2002. A Local Search Mechanism for Peer-to-Peer Networks. Proceedings of the Eleventh International Conference on Information and Knowledge Management, p.300-307.

Kazaa, 2001. http://www.kazza.com.

Napster, 2001. http://www.napster.com.

Rowstron, A., Druschel, P., 2001. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), p.329-350.

Stoica, I., Morris, R., Karger, D., Kaashoek, M., 2001. Chord: a scalable peer-to-peer lookup service for Internet applications. *Computer Communication Review*, **31**(4):149-160.

Stokes, M., 2002. Gnutella2 Specifications, Part One. http://www.gnutella2.com /gnutella2 search.htm.

Tsoumakos, D., Roussopoulos, N., 2003. Adaptive Probabilistic Search (APS) for Peer-to-Peer Networks. Technical Report CS-TR-4451, University of Maryland.

Yang, B., Garcia-Molina, H., 2002. Improving Search in Peer-to-Peer Networks. Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), p.5-14.

Zhao, B., Kubiatowicz, J., Joseph, A., 2000. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report, UCB/CSD-01-1141, University of California, Berkeley.