Proof: A DHT-based Peer-to-Peer Search Engine

Kai-Hsiang Yang
Institute of Information Science, Academia Sinica
Taipei, Taiwan 407, R.O.C.
Email: khyang@iis.sinica.edu.tw

Jan-Ming Ho
Institute of Information Science, Academia Sinica
Taipei, Taiwan 407, R.O.C.
Email: hoho@iis.sinica.edu.tw

***Abstract*— In this paper we focus on building a large scale keyword search service over structured Peer-to-Peer (P2P) networks. Current state-of-the-art keyword search approaches for structured P2P systems are based on inverted list intersection. However, the biggest challenge in those approaches is that when the indices are distributed over peers, a simple query may cause a large amount of data to be transmitted over the network. We propose a new P2P keyword search scheme, called "Proof", to reduce network traffic for queries. The key idea is storing a content summary for each web page in the inverted list, so that a query can be processed by only transmitting a small size of candidate results. Our simulation results showed that, compared with previous DHT-based P2P systems, Proof can dramatically reduce network traffic and computation time. It provides 100% precision and 90.09% recall of search results, at an acceptable cost of storage overhead, even when the number of peers and documents increases continually.**

I. Introduction

Search engines have become increasingly popular in recent years. The most famous search engine, Google, currently index 8 billion documents with approximately 10,000 cluster machines to distribute the search load. Although Google has been very successful in providing cluster computing search services for the Web, a distributed Peer-to-Peer (P2P) search engine would be more robust and practical than a cluster computing system due to its scalability, fault-tolerance, and self-organizing nature.

Recent research has proposed several techniques for providing full text keyword search in structured P2P systems [12], [14], [11], [6], [7], and significantly improved techniques for routing queries in unstructured P2P networks [8], [5], [18], [4]. To understand the performance of these systems, Yang et al [19] had provided a quantitative evaluation and direct comparison of a structured P2P system [11] and an unstructured P2P system with several optimizations [5] for full text search, including the popular and widely deployed super-peer systems used in Kazaa and the latest versions of Gnutella [17]. Their results show that all these systems use roughly the same bandwidth to process queries, and the structured systems provide the best response time (30 percent better than a super-peer system), but has a high cost of document publishing. For a search engine, the response time is the most important performance metric, we then focus on structured P2P systems in this paper.

Structured P2P systems implement a Distributed Hash Table (DHT), which manages a global identifier (ID) space. However, they do not support keyword searching directly, while keyword searching can easily be implemented by a straightforward method. For a word 'a', its inverted list (a $\rightarrow$ X, Y, Z) indicates that the keyword 'a' appears in documents X, Y, and Z, and is stored in the peer that manages the space DHT(hash(t1)). Based on these indices, a query with t keywords can be answered by at most looking for t peers to merge their corresponding inverted lists. This approach is called as "inverted list search scheme". However, it has been shown in [9] that this approach is not feasible to perform large scale keyword search, because evaluating a query usually consumes a large amount of network bandwidth.

Hence, in this paper we propose a structured P2P keyword search scheme, called "Proof", which aims to reduce network traffic and shorten search latency of a keyword search while providing a high quality of search results. The key idea in Proof is storing a content summary for each web page in the inverted list, and a peer can select possible items in inverted list by checking their content summaries. Hence, only a small amount of possible results is transmitted over the network.

We had conducted extensive experiments on the TREC data sets and our simulation results showed that, compared with previous DHT-based P2P systems, Proof can dramatically reduce network traffic and computation time. It also provides 100% precision and 90.09% recall of search results, at an acceptable cost of storage overhead, even when the number of peers and documents increases continually.

The remainder of this paper is organized as follows. Section II briefly reviews current DHT-based P2P search schemes and their limitations. Section III introduces the system architecture of Proof. In Section IV, the index structures and search algorithm are presented in detail. Section V describes our experimental methodology, and the experimental results are presented in the Section VI. Finally, our conclusions are presented in Section VII.

II. Related work

Previous works on the content searching in DHT-based P2P systems can be classified into two kinds of models: inverted list models and vector space models. The inverted list model is based on the inverted lists. Tang et al. [14] proposed a search scheme that stores few important keywords of documents into inverted lists to support the keyword searching, however it can not support full-text search because the size of inverted list will become very huge. Reynolds and Vahdat [11] adopted a technique in the database system to perform the marge

Proceedings of the 2006 IEEE/WIC/ACM International Conference
on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)
0-7695-2747-7/06 $20.00 © 2006 IEEE

operation more efficiently. It transmits the Bloom filter (a kind of content summary) of inverted lists, instead of the large-sized inverted lists themselves. However, the authors only show the way to decide the Bloom filter size for a query with two keywords, and we also show search performance of their solution is not good enough in our experiments. Besides, Karthikeyan et al. [12] store the Pagerank values of pages in the inverted lists by Pagerank order. To evaluate a query, each peer only transmits top x% of possible results to the next peer. This scheme reduces the bandwidth cost, while decreasing the recall rate. Li et al. [9] suggested combining several techniques to reduce the cost of the merge operation, including caching, applying the Bloom filter, and document clustering.

On the other hand, pSearch [15] implements a vector space model on a CAN overlay network. Both data and queries are represented as keyword (semantic) vectors, so that a query can be evaluated by searching in a multi-dimensional Cartesian space. However this approach is not efficient for more than 30 dimensions and is not scalable [16] due to the "dimensionality curse" phenomenon.

III. System Architecture and Model

The Proof system, shown in Fig. 1, comprises a crawler, a database, an index generator, and a distributed DHT-based P2P system. The crawler follows a crawling schedule to collect web pages and exacts the hyperlink information for computing their Pagerank values. At a system-defined time, an index generator is invoked to produce new index structures and publish them to the DHT-based P2P system.

In the following section, we focus on the DHT-based P2P system, which stores index structures and evaluates queries. A peer in the P2P system can be any cooperating server or personal computer that has enough capability to handle the search load. Actually, in a P2P system, any peer can be a local entry point of the Proof, which means it can accept queries from other computers and return the search results to the user. Note that the system is independent of the underlying P2P overlay network and routing protocol, and we use a Chord model [13] as an example of the underlying P2P overlay network.

Assume the system contains N peers and uses a consistent hash function, such as SHA-1 [10], to assign an M-bit identifier to each peer. There is a total of N_{doc} documents in the system, and a vocabulary, T, which is the set of all keywords in the documents. Each document d_i is composed of several keywords $t_j \subset T$. Given a query, q, containing several query keywords, $qt_j \subset T$, and a user-specified result threshold, k, the search problem can be defined as: finding the top k relevant documents that contain all the query keywords, qt_j.

A solution to the search problem includes an index structure and a search algorithm to evaluate queries. However, a good search scheme has to minimize user latency, computation time and network traffic at an acceptable storage cost, and also provide high quality of search results, i.e. the precision and recall metrics.

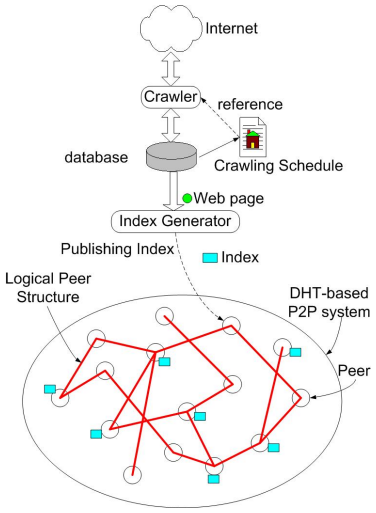

Fig. 1. The Proof system architecture, which comprises four elements: crawler, database, index generator, and DHT-based P2P system.

IV. Index structure and search algorithm

We now introduce document information and index structure in Proof, and then describe the index generation process and search algorithm.

A. Document Information

Each document X in Proof has four properties, including a document ID ($id(X)$), a Pagerank value ($pagerank(X)$), a Bloom filter of m bits ($BF(X)$), and a Bloom filter precision ($pre(X)$). Besides the document ID, which is assigned by the system DHT, the other three properties are described as follows.

(1) Pagerank value. According to the Google's Pagerank algorithm [3], each web page is assigned a Pagerank value, which represents the importance of the page, and the search results should be ranked by the Pagerank value.

(2) Bloom filter. The Bloom filter [1] is a hash-based content summary that represents a set, $A = \{a_1, a_2, \ldots, a_n\}$, of n elements (also called keys). As Figure 2) shows, to generate a Bloom filter for A is to allocate a vector of m bits, initially all set to 0, and then choose p independent hash functions, h_1,h_2,...,h_p, each within the range 1,...,m. For each element $a \in A$, the bits at positions $h_1(a), h_2(a), ..., h_p(a)$ in v are set to 1. (One bit might be set to 1 multiple times.) To membership query a keyword b, we check the bits at positions $h_1(b), h_2(b), ..., h_p(b)$. If any one of them is 0, b is certainly not in set A. Otherwise, we conjecture that b is in the set A with a certain probability that we might be wrong. In Proof, each web page can be regraded as a set of words, and a Bloom filter of those words is the content summary of that web page.

(3) Bloom filter precision. The salient feature of the Bloom filter is that there is a clear tradeoff between m and the probability of a false positive. After inserting n elements into a Bloom filter of size m, the probability of a false positive is $(1 - (1 - \frac{1}{m})^{pn})^p$. With an optimal choice of hash functions,

Proceedings of the 2006 IEEE/WIC/ACM International Conference
on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)
0-7695-2747-7/06 $20.00 © 2006 IEEE

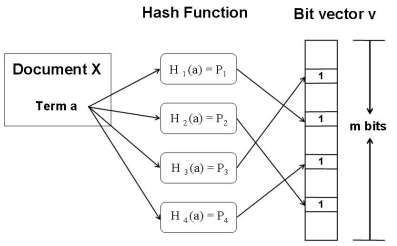

Fig. 2. Inserting a keyword, a, into the Bloom filter of document X.

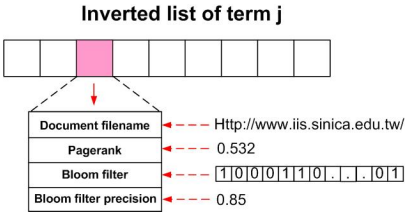

Fig. 3. Summary Inverted List of word j.

we can obtain the minimum probability of a false positive as $(\frac{1}{2})^p$, or $(0.6185)^{\frac{m}{n}}$, where p is the number of hash functions. In Proof, we define the Bloom filter precision as 1 - the probability of a false positive: $1-((1-(1-\frac{1}{m})^{pn})^p)$. The Bloom filter precision is used to measure the probability that a page hits a query.

B. Summary Inverted List

The index structure used in Proof is called the "Summary Inverted List" (SIL), which is extended from the basic inverted list. Each item in SIL not only represents a page, but also stores four properties of it. If a page, X, contains a word j, then the SIL of word j will have a list item representing the document X. As shown in Figure 3, the item includes a document ID ($id(X)$), a Pagerank value ($pagerank(X)$ = 0.532), a Bloom filter ($BF(X)$) and a precision value of the Bloom filter ($pre(X)$ = 0.85).

C. Index Generation Process

The index generation process is periodically invoked to generate SILs and distributes them in the P2P system. The process comprises the following steps.

(1) Pagerank computation. The process first applies an iterative Pagerank formula [3] to assign each document a Pagerank value.

(2) Bloom filter generation. For each page, a Bloom filter of all words in that page is generated, and it precision is also computed at the same time.

(3) Summary Inverted List generation. Suppose there are n pages and m words in the system. An $n \times m$ document-keyword matrix (M) is then created by checking every document. If a word j appears in document i, $M[i][j]$, is set to 1, otherwise is set to 0. For any word j, its SIL can be produced by checking the values in the j'th column. For any page k, if the $M[k][j]$ is 1, an item which represents the page k will be added in the SIL.

(4) Index distribution. After all SILs have been generated, they are distributed to the peers. For example, the SIL of word j is stored in the peer that owns the $DHT(hash(j))$.

(5) Sorting SIL by Pagerank. After distributing all indices, each peer sorts the items in SIL by the Pagerank value in descending order of importance.

D. Search Algorithm

Here we formally present the search process, which comprises two steps: a query flow arrangement and a query evaluation.

[Query flow arrangement]. For a query Q with several query words qt_j and a result threshold k, the first processing peer, called the "major query processing peer" (MQPP), requests all other peers which are responsible for the inverted lists of query words to report the length of its SIL. After all the lengths are received, the MQPP then determines a query flow from the shortest SIL to the longest one.

[Query evaluation]. The query process can be divided into the operation in MQPP and the operation in other peers.

(1) MQPP: the goal of query process in MQPP is to select possible results by checking their Bloom filters in inverted list. Here, only the first $T_{checking}$ pages which pass the Bloom filter checking are transmitted to the next peer, where $T_{checking}$ is the checking threshold and default set to $k + \Theta$. Here Θ is a variable called the "assurance number", and the reason to add Θ is because the Bloom membership query has a small probability of false-positives, so finding a little more possible results can guarantees a higher recall of search results. Algorithm 1 shows the pseudo code of search algorithm in MQPP.

The second step is to generate a Bloom filter $BF(Q)$ for the query, and start a checking procedure to match the Bloom filter of each item and that of query. For example, when matching item i and query Q, a bit-and (BitAnd) operation is employed to check whether the condition BitAnd($BF(item\ i)$, $BF(Q)$) == $BF(Q)$ holds; when it is true, item i is regarded as a possible result. During the checking, the MQPP computes the expected number of results, EN_{result}, by summing the Bloom filter precision of all possible results, and when EN_{result} reaches the checking threshold, $T_{checking}$, the MQPP stops the checking procedure immediately and sends the ResultList to the next peer.

(2) The rest peers: each rest peer in the query flow receives a result list from its previous peer, called "previous result list" (PRL), and it then performs a simple checking process to verify the results. As shown in Algorithm 2, the process applies a "BinarySearch" function to match each item in the PRL with its SIL. If an item is matched, it is regarded as a possible result. Otherwise it is ignored. After all list items in PRL have been checked, the new ResultList will be sent to the next peer.

*1) **Advantages**:* We design the Proof system for the following three intuitive advantages. (1) The query flow arrangement

Algorithm 1 Search Algorithm in MQPP

```
1: procedure MQPPSEARCH(Q, k, Θ)
   Q: a query
   k: a result threshold
   Θ: an assurance number
2:    T_checking ← k + Θ
3:    generate query Bloom filter, BF(Q)
4:    for all item i in SIL do
5:       if BitAnd(BF(i), BF(Q)) == BF(Q) then
6:          add item i to ResultList
7:          assign EN_result ← EN_result + pre(i)
8:       end if
9:       if EN_result ≥ T_checking then
10:         stop checking procedure
11:         send the ResultList to the next peer
12:      end if
13:   end for
14: end procedure
```

Algorithm 2 Search Algorithm in other peers

```
1: procedure OPSEARCH(Q, k, PRL)
   Q: a query
   k: a result threshold
   PRL: a previous result list
2:    for all item i in PRL do
3:       if BinarySearch(pagerank(itemi), id(itemi)) == true then
4:          add item i to the ResultList
5:       end if
6:    end for
7:    send the ResultList to the next peer
8: end procedure
```

chooses the shortest inverted list as the beginning of a join operation, which causes fewer results to be transmitted over the P2P network and processed during the join operation. This yields a great benefit in terms of network load and computation time. (2) The MQPP performs a Bloom filter checking procedure to check the items of SIL, so that only a few possible results are transmitted over the P2P network. This design reduces network traffic. Moreover, the Bloom filter's precision can be enhanced by increasing the filter's size, which makes the Bloom membership query more accurate and reduces network traffic further. (3) The MQPP sets a stop condition for the Bloom filter checking procedure. When the stop condition is reached, the MQPP immediately stops the procedure, which reduces the computation time in MQPP. In addition, all the SILs are sorted by the Pagerank value, so the retrieved documents are more important than those that had not been processed. Hence, the precision of search results can be guaranteed.

V. Experimental methodology

A. TREC Data Sets

In our experiments, we used four data sets from the TREC corpus: (1) FT: the Financial Times Limited (1991, 1992, 1993, 1994), (2) CR: Congressional Record of the 103rd Congress (1993), (3) FBIS: Foreign Broadcast Information Service (1996), and (4) LATIMES: Los Angeles Times (1989, 1990). Table I lists the data set's statistics. The FT data set contained the largest number of documents (210,158), and CR was the smallest data set with only 27,922 documents. The average number of keywords in a document in FT was 126.43, and CR had the largest average number 193.47. For all four data sets, the average number of words in a document was less than 200. Besides, we use the "title" field of TREC topics 1-600 as queries. On average, each query consisted of 3.8 keywords, and 87.8% of queries contained less than 6 query keywords.

TABLE I
TREC DATA SETS

Data set	Doc Number	Min	Max	Average	Standard Deviation
FT	210158	2	3092	126.43	99.73
CR	27922	1	5501	193.47	335.04
FBIS	130471	7	6109	140.47	137.28
LATIMES	131896	3	5375	170.03	130.25

B. Comparison of Search Schemes

We developed a simulator to evaluate the performance of the following three search schemes.

(1) Basic inverted list scheme (Basic). As mentioned in the introduction, all inverted lists of query words are transmitted among the processing peers to find the final results. We use the result of this scheme as the final results.

(2) SSB search scheme (SSB). This search scheme was proposed by Sankaralingam, Sethumadhavan, and Browne [12]. A Pagerank value of a page is stored into an inverted list, and all inverted lists are sorted by the Pagerank before query evaluation. During the merge operation, each peer only transmits top x% of results to the next peer. In our experiment, we set up x=30 (SSB-30), 60 (SSB-60), and 90 (SSB-90) as different search schemes.

(3) RV search scheme (RV). The RV search scheme was proposed by Reynolds and Vahdat [11]. The search comprises two phases: in the first phase, each peer sends a Bloom filter of inverted list to next peer, and in the second phase, each peer checks the results by a received Bloom filter. However, the size of Bloom filter is an important factor and usually depends on the data sets. We performed an experiment to choose a best size of filter for our data sets, where the size of filter is 20 times of the number of list items.

C. Performance Metrics

We measured network traffic load, computation time, and search quality of each search scheme in our experiments. First, we define Load as the number of transmitted results, which is the most important factor affecting network transmission time. Second, computation time is the total time for peers to complete the merge operation. Third, search quality is measured by the precision and recall of results.

D. Simulation Process

The simulation parameters used in our experiments are shown in Table II. The simulator first generates all indices and distributes inverted lists to the corresponding peers, and then simulates each search scheme to evaluate queries. In our simulator, all the indices are stored in the main memory; hence, all the operations in peers are memory operations.

Proceedings of the 2006 IEEE/WIC/ACM International Conference
on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)
0-7695-2747-7/06 $20.00 © 2006 IEEE

TABLE II

SIMULATION PARAMETERS

System Parameters	Notation	Default
Number of Peers	N_{peer}	500
Result Threshold	k	50
Number of Documents	N_{doc}	FT data set
Size of a document ID	S_{ID}	128 bits
Bloom Filter Size	m	600 bits
Assurance Number	Θ	25
Checking Threshold	$T_{checking}$	$k + \Theta$
Size of a Pagerank value	$S_{pagerank}$	8 bytes
Size of a Bloom filter Precision	S_{pre}	8 bytes

TABLE III

LOAD UNDER DIFFERENT QUERY FLOW

	Number of Transmitted Result		Reduce Ratio
Scheme	Original query flow (o)	Sorted query flow (s)	(s/o) %
Basic	3989.44	1573.90	39.45%
SSB-30	1149.14	442.98	38.55%
SSB-60	2336.26	909.02	38.91%
SSB-90	3568.96	1403.17	39.32%
RV	1046.97	714.52	68.25%
Proof	122.35	95.91	78.39%

VI. EXPERIMENTAL RESULTS

We now report the experimental results, and as the results for each data sets are quite similar, we only present the results of the FT data set, which was the largest. We first study the effect of different query flows, and then compare the search performance of different search schemes in terms of network load, computaion time, precision and recall. After that, we then study the effect when the number of peers and documents increases. Finally, for the parameters used in Proof, we evaluate the effect of the Bloom filter's size and the checking threshold.

A. Effect of Query Flow

The first experiment is designed to evaluate whether the Load is affected by different query flows. When a user inputs a query, the "original query flow" is defined as the input order of query keywords, and the "sorted query flow" is the order of keywords from the shortest inverted list to the longest one. Table III shows the Load of each search scheme under these two query flows. On average, the sorted query flow can reduce the Load to 39% of the original Load in both the Basic and SSB schemes, 68.25% of the original Load in the RV scheme, and 78.39% of the original Load in Proof. The results prove that the sorted query flow achieves a better search performance. Thus, we apply the sorted query flow to all search schemes in the following experiments.

B. Search Performance Comparison

The goal of the second experiment was to compare the search performance of each search scheme. We ran the simulation 100 times, and the average performance results are presented in Table IV. For the Load metric, it is clear that the Basic search scheme has the largest Load, i.e., 1118.18, the Load in SSB-x search scheme is x% of the Load in the Basic search scheme, and most importantly, the Load in Proof is the smallest (93.08), which is only 8.32% of the Load in the Basic search scheme.

TABLE IV

SEARCH PERFORMANCE

schemes	Load	CT (sec)	Precision	Recall
Basic	1118.18	10.6	100%	100%
SSB-30	331.94	9.9	92.59%	23.64%
SSB-60	664.38	10.3	93.19%	47.36%
SSB-90	1004.46	10.5	97.82%	88.97%
RV	722.87	21.9	100%	100%
Proof	93.08	5.18(1.08+4.1)	100%	90.09%

With regard to the computation time (CT), the CT in the RV search scheme is longest (21.9 sec.). Recall that the query flow in this scheme comprises two phases, so it may take more time to complete the query process. Besides, the CT in Proof is only 5.18 seconds (1.08 sec. in MQPP and 4.1 sec. in other peers), which is 48.86% of the time in the Basic search scheme. Hence, the result confirms that Proof can reduce the computation time. Note that user latency results from the computation time in peers and the time to transmit the results between peers. According to the Load and CT in Proof, it is obvious that the user latency in Proof is shortest.

Recall that the answer set of each query is defined as the top k results in the Basic search scheme, so the precision and recall in the Basic search scheme are 100%. In the SSB-x scheme, the precision slightly drops as x decreases, but the recall sharply drops to 23.64% when x = 30. This shows that many results can not be retrieved if only the top x% of results are transmitted. In Proof, the precision is 100%, which is the same as that in the Basic search scheme, and the recall is 90.09%, which is also higher than that in the SSB-x search scheme. Basically, the precision in Proof is always 100%, because all inverted lists are sorted by Pagerank. However, Proof sets a stop condition for the Bloom filter checking procedure in MQPP, if too many false-positive results occur, Proof may not be able to retrieve k results, thus the recall may decline.

Overall, Load and computation time in Proof can be greatly reduced, which means user latency can be substantially shortened, and Proof can provide 100% precision and a high recall.

C. Effect of Number of Peers

In this experiment, we studied variation of the Load when the number of peers increases. Recall that each inverted list is stored in a peer, for a given query, when two successive query keywords are mapped to the same peer, no any result needs to be transmitted; hence, the Load will be lower. When the number of peers is small, the probability of two successive query keywords being mapped to the same peer is high; therefore, Load is smaller than when there is a large number of peers.

We ran the simulation 50 times for each different setting of the number of peers, N_{peer}, (=10, 100, 1,000, 10,000, and

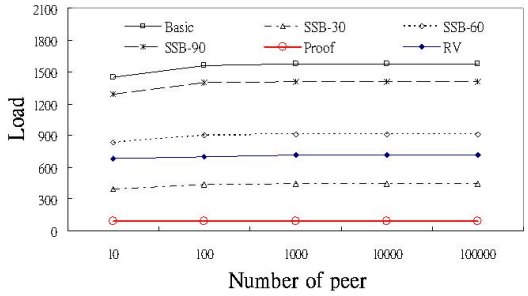

Fig. 4. Load with different number of peers.

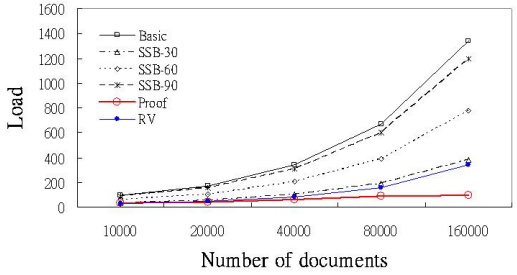

Fig. 5. Load with different numbers of documents.

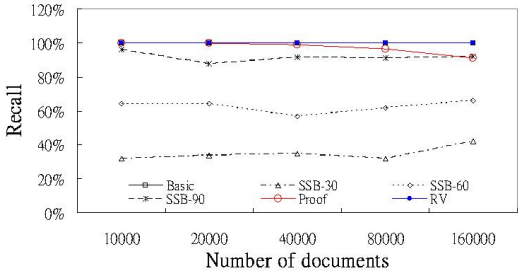

Fig. 6. Recall with different numbers of documents.

100,000). Fig. 4 depicts the average Load for different numbers of peers. Clearly, when the number of peers increases exponentially, the Load slightly increases and rapidly converges to a value. In this result, the Load in each search scheme converges when the number of peers is larger than 100. Most importantly, the Load in Proof is the smallest and remains fairly stable when the number of peers continually increases.

D. Effect of Number of Documents

After analyzing the effect of the number of peers, we now study variation of the Load when the number of documents increases. We can predict that, when the number of documents increases, all inverted lists become longer and the Load increases. We ran the simulation 50 times for each number of documents, N_{doc}, (=10,000, 20,000, 40,000, 80,000, 160,000), and the documents were randomly chosen from the FT data set.

Figs. 5 and 6 show the variation of Load and recall with each number of documents. The Load in the Basic and SSB-x schemes linearly increases with the number of documents, and the Load in the RV search scheme is similar to the Load in the SSB-30 search scheme. However, the Load in Proof is always the same. The main reason is because of the stop condition in MQPP, the Load are not affected by the size of SILs. For the recall metric, the recall in the Basic and RV schemes is always 100%, and the recall in SSB-x search scheme is x%. In Proof, the recall decreases very slowly as the number of documents increases; it is still 90% when the number of documents reaches 160,000. The reason for the decrease in recall is that more false-positive results occur as the number of documents increases, so that the query process can not retrieve k results before the stop condition is activated.

E. Effect of the Bloom Filter Size

The goal of this experiment was to study the effect of the Bloom filter size, m, in Proof. According to the analysis of Load, if m is larger, the Load will be smaller. We set up different Bloom filter sizes, m, (= 400, 600, 800, 1,000, and 1,200 bits). According to the results shown in Fig. 7, Load slowly decreases as m increases. When a larger filter is used, the filter's precision improves, so that few false-positive results occur in MQPP; hence, the Load decreases. Table V lists the Bloom filter precision and storage cost for each m. The Bloom filter precision = 78% when m = 400, and rapidly increases to 88.16% when m = 600. The last column in the table is the ratio of the total storage size in Proof to that in the Basic search scheme. Here we used 16 bytes for a document ID, 8 bytes for a Pagerank value and a Bloom filter precision. When m = 600 bits, the Bloom filter precision is high enough and the storage cost (6.69X) is acceptable.

F. Effect of the Checking Threshold

Recall that the Bloom filter checking procedure stops when the number of retrieved results reaches the checking threshold $T_{checking}$. This experiment analyzes the effect of the $T_{checking}$ on the performance metrics. We run the simulation at the setting of k=50, and Θ/k = 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2.

Table VI shows the experimental results. We can easily observe that the Load linearly rises when the Θ/k increases, because in Proof, the MQPP at most sends out the $T_{checking}$(=

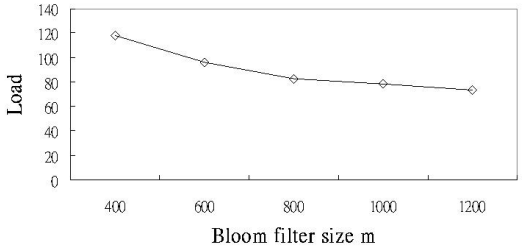

Fig. 7. Load at different Bloom filter size.

TABLE V

STORAGE COST AT DIFFERENT BLOOM FILTER SIZE(N=126.4, K=2)

m(bits)	m/n	Bloom filter precision	Storage Cost Proof/Basic
400	3.16	78.01%	5.12X
600	4.75	88.16%	6.69X
800	6.33	92.65%	8.25X
1000	7.91	95.01%	9.81X
1200	9.49	96.39%	11.38X

TABLE VI

PERFORMANCE METRICS AT DIFFERENT CHECKING THRESHOLD (K=50)

Θ/k	Load	Precision	Recall	Storage(Proof/Basic)
0.5	93.08	100.00%	90.09%	6.69X
0.75	101	100.00%	92.55%	6.69X
1	109	100.00%	93.64%	6.69X
1.25	115	100.00%	94.18%	6.69X
1.5	120	100.00%	94.89%	6.69X
1.75	125	100.00%	95.52%	6.69X
2	130	100.00%	96.27%	6.69X

$k + \Theta$) results. Most importantly, the recall rises to 96.27% when increasing Θ so that Θ/k reaches 2. This tells us the recall will rise by using larger Θ, because more relevant results should be processed before the Bloom filter checking procedure stops. Therefore, Proof can achieve higher recall by increasing the Θ/k ratio with a smaller increase of network Load.

VII. CONCLUSION

This paper was motivated by the need for a robust P2P search engine that can provide better search performance and shorter user latency. The main contribution of our work is that we propose a new P2P search scheme, called "Proof", which substantially reduces network traffic during a query process by storing a content summary of a document in inverted lists. By allowing for an acceptable space cost, Proof also substantially reduces user latency and achieves an outstanding performance, as shown by our experiment results. We also present the cost model to show that if a larger Bloom filter is used, network load and computation time could be bounded to a small amount. Most importantly, Proof is easy to implement and independent of the underlying P2P overlay network and routing protocol.

As well as developing a P2P full-text search scheme, which we are now doing, there are several promising directions for future research. In particular, we will consider the load balance issue, especially for the inverted list index structure. Note that the distribution of keywords follows a Zipf-like distribution [2]; hence, the distribution of inverted lists is unbalanced under consistent hashing. In addition, the data consistency control, replications, and recovery mechanisms are also critical to making P2P systems more reliable. We believe that more research in these areas would definitely be worthwhile.

ACKNOWLEDGMENT

This work was supported by the project of NSC 95-2221-E-001-021-MY3.

REFERENCES

[1] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of INFOCOM*, pages 126–134, 1999.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, August 2003.

[5] B. F. Cooper. An optimal overlay topology for routing peer-to-peer searches. In *ACM/IFIP/USENIX 6th International Middleware Conference*, 2005.

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

[7] Y.-J. Joung, C.-T. Fang, and L.-W. Yang. Keyword search in dht-based peer-to-peer networks. In *ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pages 339–348, Washington, DC, USA, 2005. IEEE Computer Society.

[8] A. Kumar, J. Xu, and E. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *INFOCOM 2005*, pages 1162–1173, March 2005.

[9] J. LI, B. LOO, J. HELLERSTEIN, F. KAASHOEK, D. KARGER, and R. MORRIS. The feasibility of peer-to-peer web indexing and search. In *the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[10] N. I. of Standards and Technology. Secure hash standard. FIPS 180-1 Standard in U.S. Department of Commerce/NIST, April 1995.

[11] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.

[12] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed Pagerank for P2P Systems. In *Proceedings of the 12th International Symposium on High Performance Distributed Computing*, pages 58–68, June 2003.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[14] C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, June 2004.

[15] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information retrieval in structured overlays. In *ACM HotNets-I*, October 2002.

[16] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[17] B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proceedings of the 19th International Conference on Data Engineering*, page 49, March 2003.

[18] K.-H. Yang, C.-J. Wu, and J.-M. Ho. Antsearch: An ant search algorithm in unstructured peer-to-peer networks. In *IEICE Transactions on Communications Special Section on Networking Technologies for Overlay Networks*, September 2006.

[19] Y. Yang, R. Dunlap, M. Rexroad, and B. F. Cooper. Performance of full text search in structured and unstructured peer-to-peer systems. In *INFOCOM 2006*, April 2006.