PAPER *Special Section on Networks Software*

Proof: A Novel DHT-Based Peer-to-Peer Search Engine

Kai-Hsiang YANG[†a)], ***Member and*** **Jan-Ming HO**[†], ***Nonmember***

SUMMARY 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 in this paper a new P2P keyword search scheme, called "Proof," which aims to reduce the network traffic generated during the intersection process. We applied three main ideas in Proof to reduce network traffic, including (1) using a sorted query flow, (2) storing content summaries in the inverted lists, and (3) setting a stop condition for the checking of content summaries. We also discuss the advantages and limitations of Proof, and conducted extensive experiments to evaluate the search performance and the quality of search results. Our simulation results showed that, compared with previous solutions, Proof can dramatically reduce network traffic while providing 100% precision and high recall of search results, at some additional storage overhead.
key words: *distributed search engine, inverted list, Pagerank, Peer-to-Peer*

1. Introduction

Recent research has proposed several techniques for providing full text keyword search in structured P2P systems [6], [8], [12], [13], [15], and significantly improved techniques for routing queries in unstructured P2P networks [4], [5], [10], [18]. To understand the performance of these systems, Yang et al. [19] had provided a quantitative evaluation and direct comparison of a structured P2P system [12] and an unstructured P2P system with several optimizations [5] for full text search. 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). For a search engine, the response time is the most important performance metric, we then focus on the full text keyword search in structured P2P systems.

Structured P2P systems implement a Distributed Hash Table (DHT) to manage a global identifier (ID) space. A DHT typically implements a greedy lookup protocol that contacts $O(\log n)$ peers, we say in $O(\log n)$ hops, and requires each peer to maintain a routing table of size $O(\log n)$. However, these DHT-based systems do not support keyword searching directly, while keyword searching can easily be implemented by a straightforward method, called the "inverted list search scheme." The system generates an inverted list for each word, and stores it to the peer which is responsible for the word. For example, an inverted list of word 'a' (a → X, Y, Z) indicates that word 'a' appears in documents X, Y, and Z, and it is stored to a peer responsible for the word 'a.' To evaluate a query with several words, the system has to merge corresponding inverted lists to find the pages containing those words. However, transmitting those inverted lists generates a large amount of network traffic. Several recent solutions [6], [12], [13], [15] are proposed to reduce the network traffic generated during the intersection process, however, it has been shown in [11] that the above solutions are not feasible to perform large scale keyword search, because the generated network traffic still exceeds the network capacity. Hence, it is still a promising challenge to provide keyword search service over structured P2P systems.

In this paper, we propose a structured P2P keyword search scheme, called "Proof," which aims to reduce the network traffic generated during the intersection process of inverted lists. We apply three main ideas in Proof to reduce network traffic, including the following. (1) A sorted query flow is used to decide the order of peers for the intersection process. The sorted query flow is the order of peers from the smallest list size to the largest one, which can prevent the larger lists from being transmitted over the network. (2) We trade the storage for reducing network traffic. We store Bloom filters [1] as the content summaries of web pages in the inverted lists, so that the first peer can filter out impossible list items before transmitting. This technique can effectively reduce a lot of network traffic. (3) We trade the recall metric of results for further reducing network traffic. A query in Proof is assumed to contain a required number of results, say k, and based on the probability of false-positive for checking the Bloom filters, we can stop the above checking procedure in the first peer when we have a high probability that the top k results are already found. By this design, computation time and network traffic for a single query can be dramatically reduced. Besides, we also discuss the advantages and limitations of Proof, and conducted extensive experiments to evaluate the search performance and quality of results. Our simulation results showed that, compared with previous solutions, Proof can dramatically reduce network traffic while providing 100% precision and high recall of search results, at some additional storage overhead.

The main contribution of this work is that we proposed a structured P2P keyword search scheme with three new designs to reduce network traffic which is generated during the

Manuscript received July 31, 2006.
Manuscript revised October 30, 2006.
[†]The authors are with Institute of Information Science, Academia Sinica, Taiwan, R.O.C.
a) E-mail: khyang@iis.sinica.edu.tw
DOI: 10.1093/ietcom/e90–b.4.817

intersection process of inverted lists. Compared with recent solutions [6], [12], [13], [15], Proof is more feasible to perform large scale full text search service, because the network traffic is dramatically reduced.

The remainder of this paper is organized as follows. Section 2 briefly reviews current DHT-based P2P search schemes and their limitations. Section 3 introduces the system architecture of Proof. In Sect. 4, the index structures and search algorithm are presented in detail. Section 5 describes our experimental methodology, and the experimental results are presented in Sect. 6. Finally, our conclusions are presented in Sect. 7.

2. Related Works and Basic Concepts

Previous works for providing full text search service in DHT-based P2P systems can be classified into two models: the inverted list model and the vector space model.

The inverted list model is using the inverted lists as mentioned above. Tang et al. [15] use the $TF-IDF$ methods to choose some important keywords for each document, and store them in inverted lists to support the keyword search. However the size of inverted list become too huge. Reynolds and Vahdat [12] adopted a technique to perform the intersection operation by only transmitting the Bloom filter of inverted lists. However they do not explain how to choose the proper size of the Bloom filter when the number of query keywords is larger than 2. Karthikeyan et al. [13] sort the inverted lists by Pagerank, and during the intersection operation, each peer only transmits the top $x\%$ results. The recall metric of results drops too much in this solution. Li et al. [11] suggested combining several techniques to reduce the cost of a distributed intersection, including caching, applying the Bloom filter, and document clustering.

For the vector space model, pSearch [16] implements a vector space model approach on a CAN overlay network. Both data and queries are represented by vectors and the query operation is performed in a multi-dimensional Cartesian space. However this solution is not scalable [17] due to the "dimensionality curse" phenomenon when the dimensionality increases.

2.1 Basic Concepts

2.1.1 Pagerank

A ranking algorithm is essential to a search engine to provide more important web pages in the top of search results. In this paper, we use Google's pagerank algorithm [3] as the ranking algorithm in Proof. The algorithm assigns each web page a Pagerank value which represents the importance of the page. In short, the PageRank value is a :vote;, by all the other pages on the Web, about how important a page is. A link to a page counts as a vote of support.

The intuition behind the pagerank algorithm is based on the *random surfer model*. A user visiting a page is likely to click on any of the links with equal probability and at some random point he decides to move to a new page. The formal Pagerank formulation is defined as:

$$P(a) = (1-d) + d \times \sum_{\forall \ inlinks \ i} \frac{P(i)}{C(i)}, \quad (1)$$

where $P(i)$ is the Pagerank of page i, which links to page a; $C(i)$ is the number of outbound links on page i; and d is a factor set between 0 and 1. Note that the algorithm only depends on the link structures between web pages. After the computation converges, each web page is assigned a Pagerank value as its importance.

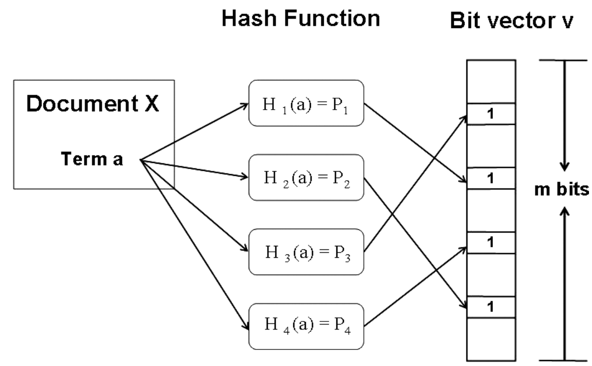

Fig. 1 Hashing word 'a' into the X' Bloom filter.

2.1.2 Bloom Filter

A Bloom filter [1] is a hash-based string that summarizes a document, $X = \{a_1, a_2, \ldots, a_n\}$, of n words, and provides an efficient method to check whether a word exists in the document. The concept, (illustrated in Fig. 1) is to allocate a string, v, of m bits, initially all set to 0, and then choose p independent hash functions, $h_1, h_2, \cdots, h_p$, each maps a word to a value within the range $1, \cdots, m$. For each word $a \in X$, the bits at positions $h_1(a), h_2(a), \cdots, h_p(a)$ in v are set to 1. (One bit might be set to 1 multiple times.) To check whether a word, b, exists in the document X, we check the bits at positions $h_1(b), h_2(b), \cdots, h_p(b)$. If any one of them is 0, b is certainly not in the document X. Otherwise, we conjecture that b exists in the document X with a certain probability that we might be wrong (we call it false positive).

The salient feature of the Bloom filter is that there is a clear tradeoff between m and the probability of a false positive. By the analysis in [1], after hashing n words into a Bloom filter of size m, the probability of a false positive can be estimated as $(1-(1-\frac{1}{m})^{pn})^p$, where p is the number of hash functions, m is the size of Bloom filter, and n is the number of hashed words.

In this paper, for a document or a query with n words, the process to hash all n words into a Bloom filter is called to "generate" a Bloom filter for the document or the query.

3. System Architecture

The Proof system, shown in Fig. 2, comprises a crawler, a

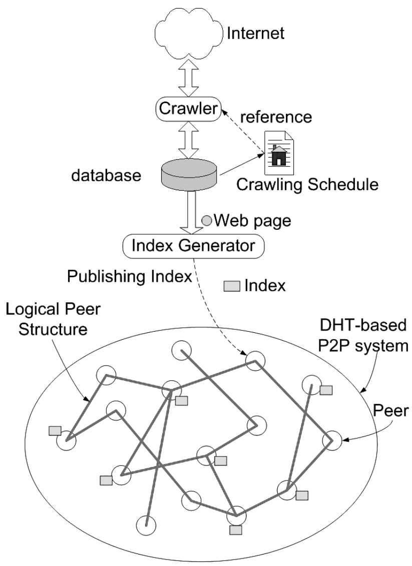

Fig. 2 The Proof system architecture, which comprises four elements: crawler, database, index generator, and DHT-based P2P system.

database, an index generator, and a distributed DHT-based P2P system. The crawler follows a crawling schedule to collect web pages from the Internet, and processes them to extract the hyperlink information and makes their indices. 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 [14] as an example of the underlying P2P overlay network.

The formal model of the DHT-based P2P system is defined as follows. Assume that the system contains N peers and uses a consistent hash function, such as SHA-1 [7], to assign an M-bit identifier to each peer. There is a total of N_{doc} documents in the system, and one 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, in the P2P system.

The 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.

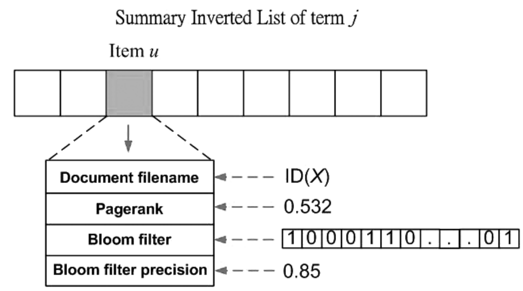

Fig. 3 Summary Inverted List of keyword j.

4. Index Structure and Search Algorithm

4.1 Document Information

In Proof, each document X contains four kinds of data: a document ID ($id(X)$), a Pagerank value ($pagerank(X)$), a Bloom filter of size m bits ($BF(X)$), and a Bloom filter precision ($pre(X)$). Besides the document ID, which is assigned by the system DHT, the meanings of the other three kinds of data are as follows.

(1) Pagerank value. After the system applies the pagerank algorithm in Equation 1 on all web pages, each web page is assigned a Pagerank value to represent its importance.

(2) Bloom filter. When a web page is crawled, Proof will automatically generate a Bloom filter as its summary.

(3) Bloom filter precision. Base on the description in Sect. 2.1.2, the probability of a false positive can be estimated as $(1-(1-\frac{1}{m})^{pn})^{p}$, where p is the number of hash functions, m is the size of Bloom filter, and n is the number of inserted elements. In Proof, we define the Bloom filter precision as (1 - the probability of a false positive), to represent the probability of finding a true result.

4.2 Summary Inverted List

Basically, Proof applies a new index structure, called the "Summary Inverted List" (SIL), which is extended from the basic inverted list. Each item in SIL stores the four data of a document. Figure 3 shows an example where a document, X, contains the word j, then the SIL of word j will have a list item, say item u, to represent the document X, and the item u contains data about the document X, including 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 ($pre(X) = 0.85$). In addition, Proof sorts all items in a SIL by their Pagerank values, and a SIL of word j is stored in the peer which is responsible for word j over the structured P2P network.

4.3 **Important Ideas in Proof**

In this section, we present the three main ideas applied in Proof to reduce network traffic during the intersection process.

4.3.1 Sorted Query Flow

The first idea in Proof is using a sorted query flow to decide the order of peers for the intersection process of inverted lists. The sorted query flow chooses the order of peers from the shortest SIL to the longest one.

Let us consider an example: assume that a query contains three query words qt_1, qt_2, and qt_3, and their summary inverted lists (SILs) are stored in peers P_1, P_2 and P_3 respectively, and assume that $|SIL_1| \gg |SIL_2| \gg |SIL_3|$, where $|SIL_1|$ means the size of SIL of word qt_1. If the system follows the original order of words ($P_1 \rightarrow P_2 \rightarrow P_3$) to process the intersection, the generated network traffic is $|SIL_1| + |SIL_1 \cap SIL_2| + |SIL_1 \cap SIL_2 \cap SIL_3|$, where $\cap$ means the intersection of SILs. Generally, $|SIL_1|$ is much larger than $|SIL_1 \cap SIL_2|$ and $|SIL_1 \cap SIL_2 \cap SIL_3|$. Hence, if we apply the sorted query flow ($P_3 \rightarrow P_2 \rightarrow P_1$) to process the intersection, it is clear to see why the sorted query flow can reduce network traffic, because the network traffic becomes $|SIL_3| + |SIL_3 \cap SIL_2| + |SIL_3 \cap SIL_2 \cap SIL_1|$, where $|SIL_3|$ is much smaller than $|SIL_1|$.

4.3.2 Content Summary Filtering

The second idea in Proof is storing Bloom filters as the content summaries of web pages in SILs. Remember that the Bloom filters can provide us to check whether a given word exists in a web page (in Sect. 2.1.2), so based on the content summaries in SILs, the first processing peer can filter out impossible list items before transmitting.

Let us consider the same example in Sect. 4.3.1 when we already applied the sorted query flow ($P_3 \rightarrow P_2 \rightarrow P_1$) to process the intersection. If Peer P_3 can check each item in SIL_3, and filters out all impossible items before transmitting, assume the size of filtered list, say $|SIL_3'|$, is much smaller than $|SIL_3|$, so the generated network traffic will be reduced to $|SIL_3'| + |SIL_3' \cap SIL_2| + |SIL_3' \cap SIL_2 \cap SIL_1|$. That is the second idea used in Proof, and please notice that the size of filtered list, $|SIL_3'|$, depends on the precision of checking Bloom filters.

4.3.3 Stop Condition of Filtering

The third idea in Proof is to set a stop condition for checking Bloom filters in the first processing peer. Because a query in Proof is assume to have a result threshold k, and all items in SILs are sorted by their Pagerank values, we can stop the checking process when we have a high probability that all the top k results are already found by the checking. Remember that the Bloom filter precision in SIL is defined as (1 - the probability of a false positive) in Sect. 4.1, so that we can calculate the expected number of results, denoted by EN_{result}, by summing all the Bloom filter precision of possible results during the checking process. Hence, we then set the stop condition as a checking threshold, denoted by $T_{checking}$, and the checking procedure stops when EN_{result} reaches the checking threshold $T_{checking}$.

However, it is not easy to decide the value of $T_{checking}$, because checking Bloom filters has a probability of a false positive, which means some of possible results are not true results, and they will be filtered out in the following intersection process. Hence, if we set a small $T_{checking}$, it is possible that we can not get k results for a query, which means the recall metric maybe drops. To avoid the effect, we set the checking threshold $T_{checking}$ to the value $k + \Theta$, where Θ is the assurance number to adapt the recall metric. Basically, we trade the recall metric for further reducing network traffic.

4.4 Search Algorithm

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

[Query flow arrangement]. For a query Q, the requester peer first sends out the "length request packets" to all peers which are responsible for the SILs of query words. After all the lengths of SILs are received, the requester peer determines the query flow from the shortest SIL to the longest one.

[Query evaluation]. The query process can be divided into the operation in the first processing peer, called the MQPP, and other peers.

(1) MQPP: the goal of the MQPP is to find out sufficient possible results by checking Bloom filters in SILs. For a query Q with several query terms qt_j and a result threshold, k, the MQPP first generates a Bloom filter for the query by hashing all query terms to the filter, as mentioned in Sect. 2.1.2. We use $BF(Q)$ to denote the query's Bloom filter. The checking procedure checks each item in SIL by a bit-and (BitAnd) operation. For an item i, if the condition BitAnd($BF(item\ i)$, $BF(Q)$) == $BF(Q)$ holds, item i is regarded as a possible result and then selected from the SIL. After the checking procedure stops, those selected possible results are sent to the next peer as a "new" ResultList.

In addition, as mentioned in Sect. 4.3.3, the MQPP sets a stop condition for the checking procedure. The expected number of results, EN_{result}, is defined as the sum of all Bloom filter precision of selected results, and the checking procedure stops when the EN_{result} reaches the checking threshold, $T_{checking}$, which is set to the value $k + \Theta$, where Θ is the assurance number. Algorithm 1 shows the pseudo code of search algorithm in the MQPP.

(2) Other peers: excluding the MQPP, each peer in the query flow (which is supposed to own a SIL of a query term) will receive a result list from its previous peer, called "result list," and its goal is to perform the intersection of the received result list and its SIL. After the intersection, the new list is sent to the next peer as a "new" result list.

4.4.1 Advantages and Limitations

We design the Proof system for the following three intuitive

Algorithm 1 Search Algorithm in MQPP

```
procedure MQPPSearch(Q, k, Θ)
  Q: a query
  k: a result threshold
  Θ: an assurance number
  T_checking ← k + Θ
  EN_result ← 0
  generate query Bloom filter, BF(Q)
  for all item i in SIL do
    if BitAnd(BF(i), BF(Q)) == BF(Q) then
      add item i to ResultList
      assign EN_result ← EN_result + pre(i)
    end if
    if EN_result ≥ T_checking then
      stop checking procedure
      send the ResultList to the next peer
    end if
  end for
end procedure
```

advantages. (1) The query flow arrangement chooses the shortest inverted list as the beginning of an intersection operation, which causes fewer results to be transmitted over the P2P network. This yields a great benefit in terms of network load and computation time. (2) The MQPP performs a Bloom filter checking procedure to filter out impossible items from the SIL, so that only a few possible results are transmitted over the P2P network. (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, because all items in SIL are sorted by Pagerank values, the more important items are checked early then the less important ones. However, if the false positive of checking Bloom filters occurred too much, the checking procedure may not get enough k results, so that the recall metric maybe drops. On the other hand, after all the intersection process, all the found results are true results, which means the precision of search results can be guaranteed.

In fact, Proof cannot achieve 100% recall structurally, because it also concerns other merits, such as latency, network traffic, and computation time simultaneously. Therefore, Proof is obviously unfitted to some kinds of searches which need very high recall value.

5. Experimental Methodology

5.1 Data Sets

In our experiments, we used four kinds of 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 1 lists the data set's statistics, including the min, max, average, and standard deviations of the number of different words in a document. The FT data set contained the largest number of documents (210,158), and the average number of words in a document in FT was 126.43. 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, and each query contained 3.8 query words on average.

Table 1 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

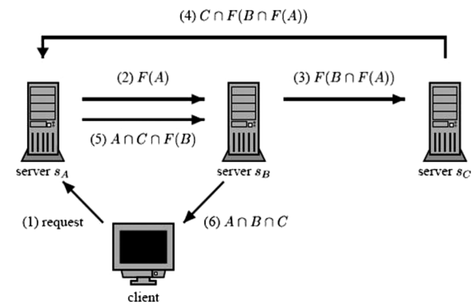

Fig. 4 Query evaluation in the RV search scheme.

5.2 Comparison of Search Schemes

We developed a simulator to compare the performance of Proof and the following three P2P search schemes.

(1) Basic inverted list search scheme (Basic). As mentioned in the introduction, the inverted list search scheme is the most basic way to implement keyword searching on structured P2P networks. For example, given a query $Q = \{a, b, c\}$, the search scheme follows the original order of words to intersect their inverted lists. The entire inverted list of term 'a' is transmitted to the peer having the inverted list of term 'b,' and the intersected results are transmitted to the peer having the inverted list of term 'c,' and then all final results are returned to the user. Note that although it consumes a lot of computation time and network bandwidth, the result of this scheme is always correct, so we take the results in this scheme as the answers in our experiments.

(2) SSB search scheme (SSB). This search scheme was proposed by Sankaralingam, Sethumadhavan, and Browne [13]. A Pagerank value of a web page is computed and stored into an inverted list, and all inverted lists are sorted by the Pagerank of list items before query evaluation. The intersection process also follows the original order of words that a user inputs, and each peer only transmits the top x% of results to the next peer. The parameter x affects the transmitted data size and the precision of the final results. 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 [12]. Figure 4 shows the query process for a query with three query keywords. The intersection process also follows the original order of words that a user inputs, and 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.

5.3 Performance Metrics

We measured the network traffic load, computation time, and search quality of each search scheme by the following metrics:

(1) **Load**. We define Load as the number of transmitted results, which is the most important factor affecting network transmission time.

(2) **Computation Time (CT)**. Computation time is the total time for peers to perform the intersection operation. In our experiments, each search scheme was run in the same environment so that we could measure the computation time fairly.

(3) **Precision and Recall of Results**. For a query with a result threshold, k, we define the answer set, R, as the top k final results in the Basic search scheme. Assume that any search scheme returns a search set, A, and let $|Ra|$ be the number of results at the intersection of sets R and A. The recall and precision are defined as follows. $Recall = \frac{|Ra|}{|R|}$, and $Precision = \frac{|Ra|}{|A|}$.

5.4 Simulation Process

We set up the simulation parameters shown in Table 2 before running any simulation for each experiment. These 9 parameters come from what we need to run a simulation. First, for the system, we have to know (1) the number of documents and (2) the number of peers. Second, for the four kinds of data for each document, we have to decide the size: (3) size of a document ID, (4) Bloom filter size, (5) size of a Pagerank value, and (6) size of a bloom filter precision. The rest three parameters are needed for the stop condition of the checking procedure, including (7) the result threshold, k, (8) assurance number, Θ, and (9) the checking threshold, $k + \Theta$.

The simulator first generates all the SILs and distributes them to peers via the consistent hashing algorithm [9], 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.

Table 2 Simulation parameters.

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

6. 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, computation 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.

6.1 Effect of Query Flow

The goal of first experiment is to verify the effect of the sorted query flow on the Load metric. When a user inputs a query, the "original query flow" is defined as the input order of query terms, and the "sorted query flow" is the order of terms from the shortest inverted list to the longest one. We applied these two query flows to each scheme, and the results are shown in Table 3. According to the traffic ratio, the sorted query flow only generates near 40% of the original network traffic Load in both the Basic and SSB schemes, 68% in the RV scheme, and 78.39% in Proof. The reason why the traffic ratio can not be as small as that in other schemes is that, Proof has a stop condition to limit the number of transmitted results. According to this result, we can understand why Proof applies the query flow arrangement before the query evaluation, and the results prove that the sorted query flow does achieve a better search performance.

Because all these three search schemes only use the "original query flow," and we already understand the effect of the query flow. For better understand the effect of other designs in Proof, we then apply the "sorted query flow" to all search schemes in the following experiments for a fair performance comparison (without the effect of the query flow).

6.2 Search Performance Comparison

The goal of this experiment is to compare the search performance of each search scheme. We ran the simulation 100 times, and the average performance results are presented in Table 4. For the Load metric, it is clear that the Basic search scheme has the largest Load (1118.18), the Load

Table 3 Load under different query flow.

Scheme	Number of Transmitted Result		Traffic Ratio
	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%

Table 4 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%

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.

As for computation time (CT), the CT in the RV search scheme is longest (21.9 sec.), and 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. This result confirms that Proof can reduce the computation time. 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. The recall drops because too many false-positive results occur, so that Proof may not be able to retrieve k results before the checking procedure stops.

Overall, Proof can greatly reduce Load and computation time and provide 100% precision and a high recall.

6.3 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 terms 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 terms 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 100,000). Figure 5 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.

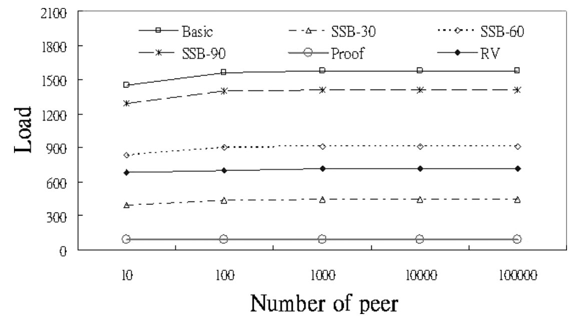

Fig. 5 Load with different numbers of peers.

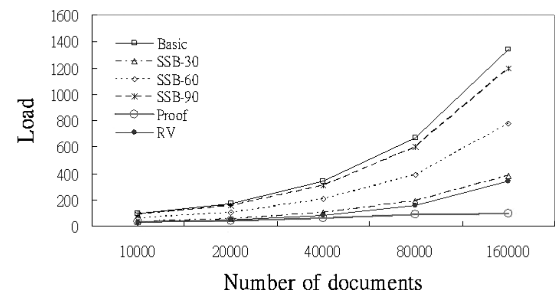

Fig. 6 Load with different numbers of documents.

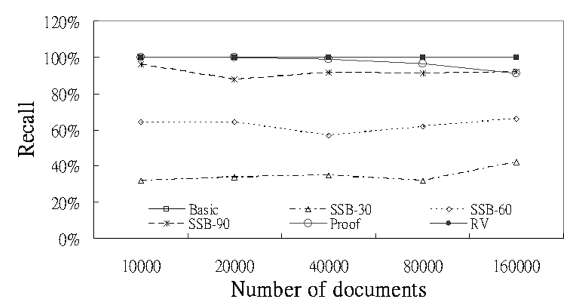

Fig. 7 Recall with different numbers of documents.

6.4 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.

Figures 6 and 7 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

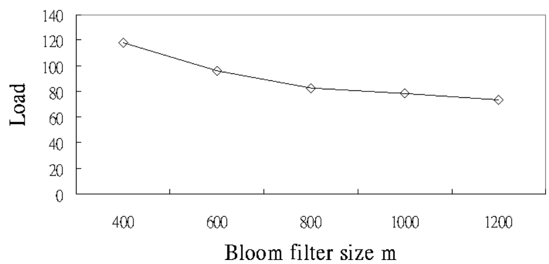

Fig. 8 Load at different Bloom filter size.

Table 5 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%	5.12
600	4.75	88.16%	6.69
800	6.33	92.65%	8.25
1000	7.91	95.01%	9.81
1200	9.49	96.39%	11.38

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.

6.5 Effect of the Bloom Filter Size

The goal of this experiment was to study the effect of the Bloom filter size, m, in Proof. We set up different Bloom filter sizes, m, (= 400, 600, 800, 1,000, and 1,200 bits). According to the results shown in Fig. 8, 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 5 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. Assume that we can accept the storage cost within 7 times of that in the Basic search scheme, we can choose (m = 600) to achieve 88% Bloom filter precision under this storage constrain.

7. 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 three designs in Proof. 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. Note that the distribution of words follows a Zipf-like distribution [2]; hence, the distribution of inverted lists is unbalanced under consistent hashing. Besides, 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.

References

[1] B.H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Commun. ACM, vol.13, no.7, pp.422–426, 1970.
[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," INFOCOM, vol.1, pp.126–134, 1999.
[3] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," Computer Networks and ISDN Systems, vol.30, no.1–7, pp.107–117, 1998.
[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," ACM SIGCOMM 2003, pp.407–418, Aug. 2003.
[5] B.F. Cooper, "An optimal overlay topology for routing peer-to-peer searches," ACM/IFIP/USENIX 6th International Middleware Conference, pp.82–101, 2005.
[6] O. Gnawali, A keyword set search system for peer-to-peer networks, Master's Thesis, Massachusetts Institute of Technology, June 2002.
[7] National Institute of Standards and Technology, Secure hash standard, FIPS 180-1 Standard in U.S. Department of Commerce/NIST, April 1995. http://www.itl.nist.gov/fipspubs/fip180-1.htm
[8] Y.-J. Joung, C.-T. Fang, and L.-W. Yang, "Keyword search in dht-based peer-to-peer networks," Proc. 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05), pp.339–348, Washington, DC, USA, 2005.
[9] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," ACM Symposium on Theory of Computing, pp.654–663, May 1997.
[10] A. Kumar, J. Xu, and E. Zegura, "Efficient and scalable query routing for unstructured peer-to-peer networks," INFOCOM 2005, pp.1162–1173, March 2005.
[11] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris, "On the feasibility of peer-to-peer web indexing and search," 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), pp.207–215, Berkeley, CA, USA, Feb. 2003.
[12] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," Proc. International Middleware Conference, pp.21–40, June 2003.
[13] K. Sankaralingam, S. Sethumadhavan, and J.C. Browne, "Distributed Pagerank for P2P Systems," Proc. 12th International Symposium on High Performance Distributed Computing, pp.58–68, June 2003.
[14] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," Proc. 2001 conference on applications, technologies, architectures, and protocols for computer communications, pp.149–160, 2001.
[15] C. Tang and S. Dwarkadas, "Hybrid global-local indexing for efficient peer-to-peer information retrieval," Proc. Symposium on Networked Systems Design and Implementation (NSDI), pp.211–224, June 2004.
[16] C. Tang, Z. Xu, and M. Mahalingam, "pSearch: Information re-

trieval in structured overlays," ACM HotNets-I, pp.89–94, Oct. 2002.
[17] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," VLDB'98: Proc. 24rd International Conference on Very Large Data Bases, pp.194–205, San Francisco, CA, USA, 1998.
[18] K.-H. Yang, C.-J. Wu, and J.-M. Ho, "AntSearch: An ant search algorithm in unstructured peer-to-peer networks," IEICE Trans. Commun., vol.E89-B, no.9, pp.2300–2308, Sept. 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," INFOCOM 2006, April 2006.

Kai-Hsiang Yang received a B.A. degree in Department of Mathematics from National Taiwan University and his Ph.D. degree in Department of Computer Science and Information Engineering from National Taiwan University. He is currently a Postdoctoral Fellow in the Computer Systems and Communication Lab, Institute of Information Science, Academia Sinica. His research interests include Web mining, Peer-to-Peer computing, search engine technique, network protocols and architecture, network security, and information retrieval.

Jan-Ming Ho received his Ph.D. in electrical engineering and computer science from Northwestern University in 1989. He received his B.S. in electrical engineering from National Cheng Kung University in 1978 and his M.S. from the Institute of Electronics, National Chiao Tung University in 1980. He joined the Institute of Information Science, Academia Sinica, Taiwan, R.O.C as a associate research fellow in 1989, and was promoted to research fellow in 1994. He was deputy director of the Institute from 2000 to 2003. He visited the IBM T. J. Watson Research Center in the summers of 1987 and 1988, the Leonardo Fibonacci Institute for the Foundations of Computer Science, Italy in summer 1992. He is Associate Editor of IEEE Transaction on Multimedia. He is a member of IEEE and ACM. He was Program Chair of Symposium on Real-time Media Systems, Taipei, 1994–1998, General Co-Chair of International Symposium on Multi-Technology Information Processing, 1997 and will be General Co-Chair of IEEE RTAS 2001. His research interests target the integration of theory and application research, and include digital archive technology, web services, information extraction and knowledge management, content network and continuous video streaming, and combinatorial optimization.