Performance of Bloom Filter for Multikey Word Search

C.Madula,
Department of Information Technology.
Sathyabama University, Chennai.
Madula.777@gmail.com

A.Gowri,
Department of Information Technology,
Sathyabama University, Chennai.

Abstract - *In peer to peer network retrieval mechanisms provide a scalable distributed hash table (DHT) that store keywords and the location of document. It allows every individual keyword to be mapped to a set of documents. Earlier Schemes single keyword based index, a list of entries for each keyword in a query can be retrieved by using existing DHT lookups. For multi keyword search, it increases the traffic and latency it performance is not effective So, Searching of multiple keyword is the became a challenging one. Commonly utilize Bloom Filter to overcome the traffic cost. It's not effective for the large storage system. So, In this paper we optimize the setting of bloom filter encoding mechanism, by increasing the bit array size and number of hash function. From this we can minimize the false positive rate.*

1. Introduction

A peer-to-peer (abbreviated to P2P) computer network is one in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to various resources such as files, peripherals, and sensors without the need for a central server. P2P networks can be set up within the home, a business, or over the Internet. Each network type requires all computers in the network to use the same or a compatible program to connect to each other and access files and other resources found on the other computer. P2P networks can be used for sharing content such as audio, video, data, or anything in digital format.

P2P is a distributed application architecture that partitions tasks or workloads among peers. Peers are equally privileged participants in the application. Each computer in the network is referred to as a node. The owner of each computer on a P2P network would set aside a portion of its resources—such as processing power, disk storage, or network bandwidth—to be made directly available to other network participant, without the need for central coordination by servers or stable hosts. With this model, peers are both suppliers and consumers of resources and also it can supplement the user detection and recovery and improve quality assurance activity of the product. in contrast to the traditional client–server model where only the server supply (send), and clients consume (receive). Emerging collaborative P2P systems are going beyond the era of peers doing similar things while sharing resources, and are looking for diverse peers that can bring in unique resources and capabilities to a virtual community thereby empowering it to engage in greater tasks beyond those that can be accomplished by individual peers, yet that are beneficial to all the peers.

Structured P2P networks employ a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired file/resource, even if the resource is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. The most common type of structured P2P networks implement a distributed hash table (DHT), in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot.

A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table; (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of

participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

Keyword search is a common operation in file sharing networks, because only after locating a file, a user can download it. Controversy surrounds these networks because they allow users to share files which often are illegal to distribute. One of the first P2P networks which allowed users to share files Using this central server, a peer requested a list of peers which are sharing items matching the query. After this list is returned by the server, a peer can decide if and which items to download. Items are downloaded in a direct fashion, thus no intermediate peers. A true P2P network being a network in which all peers are equal (the central server is not equal to all other peers). This mechanism can adapt to system's churn and reduce traffic.

2. Related works

Google has had great success providing centralized search services for the Web. However, we believe that for peer-to-peer file systems and archival storage networks, a distributed search service is better than a centralized one. First, centralized systems provide a single point of failure. Failures may be network outages; denial-of-service attacks, as plagued several or censorship by domestic or foreign authorities. In all such cases, a replicated distributed system may be more robust. Second, many uses of peer-to-peer distributed systems depend on users voluntarily contributing computing resources. A centralized search engine would concentrate both load and trust on a small number of hosts, which is impractical if those hosts are voluntarily contributed by end users. Both centralized and distributed search systems benefit from replication. Replication improves availability and throughput in exchange for additional hardware and update costs. A distributed search engine benefits more from replication, however, because replicas are less susceptible to correlated failures such as attacks or network outages. Distributed replicas may also allow nodes closer to each other or to the client to respond to queries, reducing latency and network traffic. Although a sufficiently small index need not be partitioned at all, our target application is a data set large enough to overwhelm the storage and processing capacities of any single node. Thus, some partitioning scheme is required. There are two straightforward partitioning schemes: horizontal and vertical.

Partitioning for each keyword an index stores, it must store a match-list of identifiers for all of the documents containing the keyword. A horizontally partitioned index divides this list among several nodes, either sequentially or by partitioning the document identifier space. Google operates in this manner. A vertically partitioned index assigns each keyword, undivided, to a single node, a small sample index partitioned horizontally and vertically, with K1 through K5 representing keywords and doc1 through doc9 representing documents that contain those keywords. A vertically partitioned index minimizes the cost of searches by ensuring that no more than k servers must participate in answering a query containing k keywords. A horizontally partitioned index requires that all nodes be contacted, regardless of the number of keywords in the query. However, horizontal indices partitioned by document identifier can insert or update a document at a single node, while vertically partitioned indices require that up to k servers participate to insert or update a document with k keywords. As long as more servers participate in the overlay than there are keywords associated with an average document, these costs favor vertical partitioning. Furthermore, in file systems, most files change rarely, and those that change often change in bursts and may be removed shortly after creation, allowing us to optimize updates by propagating changes lazily. In archival storage systems, files change rarely if at all. Thus, we believe that queries will outnumber updates for our proposed uses, further increasing the cost advantage for vertically partitioned systems. Vertically partitioned indices send queries to a constant number of hosts, while horizontally partitioned indices must broadcast queries to all nodes. Thus, the throughput of a vertically partitioned index theoretically grows linearly as more nodes are added. Query throughput in a horizontally partitioned index does not benefit at all from additional nodes. Thus, we chose vertical partitioning for our search engine.

Bloom filter is a hash-based data structure that summarizes membership in a set. By sending a Bloom filter based on A instead of sending A itself, we reduce the amount of communication required for *s*B to determine A\B. The membership test returns false positives with a tunable, predictable probability and never returns false negatives. Thus, the intersection calculated by *s*B will contain all of the true intersection, as well as a few hits that contain only *k*B and not *k*A. The number of false positives falls exponentially as the size of the Bloom filter increases.

Caching can eliminate the need for sA to send A or F (A) if server sB already has A or F (A) stored locally. We derive more benefit from caching Bloom

filters than from caching entire document match lists because the smaller size of the Bloom representation means that a cache of fixed size can store data for more keywords. The benefit of caching depends on the presence of locality in the list of words searched for by a user population at any given time. To quantify this intuition, we use the same ten-day IRCache trace described to determine word search popularity. Keyword popularity roughly followed a Zipf distribution, with the most common keyword searched The dominance of popular keywords suggests that even a small cache of either the Bloom filter or the actual document list on A is likely to produce high hit rates. Zhang and Suel propose to use multiple rounds BFs to reduce the cost for large-scale textual collections. By transmitting the BFs of sets instead of raw sets among peers, block by block, the BF-based schemes effectively reduced the communication cost. Reynolds and Vahdat claim that optimal BF settings can be achieved through minimizing the false positive of a BF. In this paper, we show that minimizing the false positive of a BF is far from optimal settings.

3. Minimizing Communication Cost for Multikeyword Search

Before we discuss the mechanism for reducing the communication cost for multikeyword search, we introduce the following concepts: Observations on user behaviors. We recently analyzed four months query logs of a major commercial web search engine. The query length distribution. From which we can observe that 56 percent of the queries consist of at least two terms. This indicates that multikeyword search is quite common in web content searching.

3.1 Optimal setting of bloom filter

In this section, we show how to achieve the minimized communication cost by using optimal settings of BFs. We analyze the communication cost quantified. We consider three typical situations $|X|<|Y|$, $|X|=|Y|$, and $|X|>|Y|$. We set r to 250 bits based on the research results conducted on Google search engine, which show that the average URL length. We adjust the parameters m and k and examine how the value of $f(m, k)$ changes. We find that the intersection order is critical for minimizing the communication cost. When $|X|$ is not greater than $|Y|$, the

Communication cost can be minimized. The value of

$f(m, k)$ is significantly influenced by the variable m. The minimal value of $f(k,m)$ can be achieved when m is set as an optimal value. The minimal communication cost changes very slightly when we adjust the value of parameter k while fixing the value of parameter m. The results demonstrate that the optimal BF is determined by the popularities of keywords and the intersection order. Much benefit can be achieved if we transmit the BF for the set of a less popular keyword to the DHT node responsible for a popular keyword during the process of distributed intersection. Based on these observations, given $|X|$, $|Y|$ and k, the objective of our optimal BF based intersection algorithm is to enable each node intelligently choose the optimal m *and* the intersection order to achieve the minimal communication cost. In this design we first sort the keywords for an intersection operation in increasing order according to their popularities, $|X|<|Y|$. By varying the values of $|X|$ and $|Y|$ we obtain a set of sample values for optimal m. The optimal values of m for any given $|X|$ and $|Y|$. The results show that with same values of $|X| / |Y|$, the value $m / |X|$ is a constant, where m is the optimal setting. For simplicity, we use u to denote $|X| / |Y|$ and v to denote $m / |X|$. Thus, we can derive a function $v = f(u)$. Thus each node can determine the optimal settings of BF according to the popularities of query keywords with no extra configuration cost.

AND Query-A common solution for a multikeyword search needs conducting a distributed intersection operation in a wide area network. Fig. 2a gives an example of a two-keyword (x, y) search. The query is first routed to the DHT node which is responsible for keyword x. Then, X, the set of identifiers of documents that contain keyword x, is transmitted to the node which is responsible for keyword y for a consequent intersection operation to achieve X \ Y , where Y is the set of document identifiers whose corresponding documents contain keyword y. The final results are returned to the client. Based on the analysis of the four months query logs in the WT10G data collection, we find that in most cases, the minimal cardinality of the set of documents that contain any single keyword in a query is much larger than the cardinality of the intersection of all the sets. Thus, in a DHT-based P2P web search system, the straightforward distributed intersection operation only achieves a relatively small result set at the cost of sending complete sets in the wide area network. Clearly, the communication cost can be saved by transmitting the BF of sets instead of raw sets among peers for distributed intersection. Existing work claimed that minimizing the false positive rate of a BF is most efficient in reducing the communication cost. In this paper, we show that this is not the case. For the same example discussed above, our design reduces the communication cost by sending an optimal BF based on X, BF(X), instead of sending X When BF(X) is transmitted to the DHT

node which is responsible for keyword y, the node determines the intersection of X and Y based on BF(X). Because the BF has no false negatives, the result set will contain

Algorithm1: Distributed algorithm for "AND" query

Require: Query q (x and y)

List of postings separately containing x and y and Accommodated by DHT nodes S_x and S_y in a Distributed manner

Ensure: List of documents containing both x and y

Step (1): Issue Query

1: Client choose S_x to send the query a(x,y).

Step (2): compute BF(x)

1: S_x identifies the set X which contain all the Document containing x.

2: S_x generates a BF with optimal setting for X using hash function {hj(.),1<j<k}.

3: compute Y∩BF(X)←Ø.

Step (3): compute Y∩BF(X)

1: S_y identifies the set Y which contain all the documents cont.ining y.

2: Y∩BF(X)←Ø.

3: for i=1 to|Y| do

4: S_y checks bi, an item of Y against BF(X) with hash

function {hj(.),1<j<k}.

5:if ∀ (j) (1<j<k),s.t.hj(bi)=1then

6: Y∩BF(X)← (Y∩BF(X))∪{bi}

7:S_y transmits Y∩BF(X) to S_x.

Step(4): reverse verification

1:S_x picks out the false positive.

2:X∩Y← X∩(Y∩BF(X))

3:$S_x$sends the result X∩Y to the client.

All elements of the true intersection. Due to the possible false positives, the result set may contain elements that contain only keyword y but not x. typically, a client would like to retrieve only the exact intersection of X and Y. Thus, the result set, denoted by Y \ BF(X), is sent back to the DHT peer responsible for keyword x to remove the false positives.

OR Query-In some applications; we need "OR" queries, which desire the results containing any keyword in the query. Such query is critical for queries whose keywords are rare in the system. A search engine may combine both the "AND" and "OR" results for a multikeyword query to the users. Presents an example of the straightforward strategy for a two-keyword "OR" query. At the beginning, the query is separately sent to the DHT nodes responsible for keywords x and y, respectively. Then, the DHT nodes separately send back the complete list for each keyword. At last, the results of both keywords are merged at the client. Thus, the total communication cost, in our design the query is first routed to the DHT node which is responsible for keyword x. Then, BF(X) will be forwarded to the DHT node that is responsible for keyword y to pick out the documents which are not in X by checking elements in Y using BF(X). Only the set picked out, denoted by Y - BF(X), is returned to the client for a consequent union operation. Algorithm 2 shows the process of distributed union for "OR" queries in detail. Ouralgorithm for distributed union is promising for reducing traffic cost of some queries (especially, the queries consist of rare but strongly correlated terms). Using the global statistics information of keywords, we can use a threshold to select the strategy. We use BF for distributed union operation; otherwise, we use the straightforward strategy. Note that Y -BF(X) is slightly different from Y _ X due to the false positives. Thus, some results of X \ Y will be missed in the final results. The trade-off between the loss rate and the communication cost. It shows that when m is increased, the communication cost increases while the loss rate decreases. In this example, the communication cost can be reduced .The loss rate can become quite acceptable by using a BF with larger size, while the communication cost will rise.

Algorithm1: Distributed algorithm for "OR" queries

Require: Query q (x and y)

List of postings separately containing x and y and accommodated by DHT nodes S_x and S_y in a distributed manner

Ensure: The estimated set of documents containing

xor y

Step (1): Issue Query

1:Client choose S_x to send the query a(x,y).

Step (2): return result X

1: S_x identifies the list of posting X which contain all

the document containing x.

2: S_y sends X to client.

Step (3): computing BF(X).

1: $S_x$creates an empty m-bits bit vector for BF(X),the

bloom filter with minimized false positive, for X.

2: S_x generates a BF with optimal settings for X using

hash function{hj(.),1<j<k}.

3:S_x transmits BF(X) to S_y.

Step(4): estimate union

1:S_y identifies the list of posting Y, which contains the documents containg y.

2:Y∩BF(X)←Ø.

3: for i=1 to|Y| do

4: S_y checks bi, an item of Y against BF(X) with Hash function {hj(.),1<j<k}

5:if ∃ (j) (1<j<k),s.t.$h_j(b_i)$=0 then

6: Y-BF(X) ← (Y-BF(X))∪{b_i}

7: S_y transmits Y-BF(X) to the client.

When we choose algorithms in a real-world system design, we may consider this trade-off between the search quality for the user and system resource consumption. In this design, we minimize the false positive to achieve the best recall rate. Given a specific ratio of m=n, i.e., the number of bits per element, it is easy to prove that the false positive rate. For distributed union is query may have inevitable differences of recall in the distributed union operations due to the false positive of BFs. The first keyword won't have any missing results while the later ones may have missing ones. In the distributed union algorithm, we do not consider a complete search mechanism using reverse verification like the distributed intersection algorithm presented. Such technique indeed achieves 100 percent recall for all the keywords but consumes even more communication cost than the straightforward strategy shown in Fig. 4a that transmits all the sets separately and directly to the client. In practice, when we design a real-world system, we can use some weighting techniques to differentiate the importance of keywords. For example, let the rarer keywords in the query have higher weights and process such keywords before popular keywords.

4. Global Keyword Popularity

Within the structure of a hybrid P2P network, we use a variant of the push-synopsis gossip algorithm first proposed in to gather global keyword popularity in the web. The robust algorithm enables every peer to quickly collect the global statistical term frequency in the P2P web. It is not difficult to see that the global frequency of a term x can be obtained from the node holding the inverted list of x. In this design, we do not obtain the statistics from DHT nodes during query processing due to the extra latency by obtaining such information via DHTs. Specifically; the inquiry message needs to be transferred hops across the wide area network (application layer) to obtain the term frequency. In contrast by using the gossip algorithm, each node can obtain the global term frequency for all the keywords directly from the local synopsis with a constant latency, greatly reducing the query processing time. The main idea of the method is as follows: consider the example of jXj, the global statistical frequency of keyword x, the method first lets all peers in the network check their local index. When the keyword x is found the first time in a document on a peer, this peer does the following experiment: it flips a coin up to t times and counts the number of times the head appears before the first tail. It saves this count in a value Then, the is gossiped among the peers in the network. During each round of gossip, each node chooses a random neighbor and sends the neighbor the FC(X) value it locally holds. After receiving the FC(X) values from a neighbor, a peer computes the maximum value of FC(X), i.e., max FC(X). The results in show that the robust gossip scheme leads the computation of aggregated information to converge exponentially: The pushing synopsis-based gossiping algorithm for estimating global statistical keyword frequency has three main operations:

4.1 Synopsis generation.

At the initialization phase when a peer joins the network the first time, it browses its local index and generates a local synopsis using the duplicated-insensitive counting technique proposed by Flajolet and Martin. The synopsis structure is designed as (x, bitvec_x), where (x, bitvec_x) is a bit vector for counting the statistical frequency of keyword x. Considering initializing bitvec_x as an example, when a peer finds keyword x in a document the first time, it does the coin flip experiment and saves the FC(X) value in by setting the FC(X) bit to "1".

4.2 Synopsis disseminating.

The synopses are disseminated among peers using the randomized gossip algorithm. During gossip round, each node randomly chooses a neighbor and sends the selected neighbor its synopsis. We illustrate an example of one round of gossip.

4.3 Synopsis merging.

When a peer receives the synopsis from its neighbor, it checks the synopsis it receives against its own synopsis and performs the following merging operation. For the keyword t in both synopses, it performs the bitwise-or operation on the pair of bit vectors for bitvect; and for those keywords in the synopsis of the neighbor but not in the local synopsis, it merges the relevant bitvector into its own synopsis. The process at Peer B after it merges the synopsis that Peer A gossips to it. The bitwise-or operation has the effect of computing the maximal of the FC values of the independent experiments in an order insensitive manner.

In addition to the three operations, we must deal with the change of the global statistics over time because of leaving of nodes. Note that a synopsis with a counter much larger than its exact number almost surely contains data from every local index node. However, a particular local node who leaves has no way of deleting what it has ever contributed the synopsis, making the global statistics stale. To

cope with this problem, in our design when a node receives a synopsis with a counter number larger than some maximum value V , it drops both the incoming and its local bit vector in the synopsis. It then recomputes a local synopsis based on its current document set and performs the gossip protocol based on the new synopsis. This periodic purging of old statistics guarantees that stale information will eventually leave the system. We will study the optimal choice of V in future work.

5. Conclusions

In this paper, we show mathematically that the optimal setting of BF in terms of traffic cost is determined by the numbers of items involved on both sides. We derive an effective approach to achieve BF optimal settings through numerical analysis. We also proposed the optimal order strategies for both "AND" and "OR" queries. We conduct comprehensive simulations based on TREC WT10G test collection and the query logs of a commercial web search engine. Simulation results show that our design outperforms existing work. In the future work, we will try to examine the performance of more comprehensive solutions by using larger scale data collections.

6. References

[9] H.V. Jagadish, B.C. Ooi, and Q.H. Vu, "Baton: A Balanced Tree Structure for Peer-to-Peer Networks," Proc. Int'l Conf. Very Large Data Bases (VLDB), pp.661-672, 2005. 704 IEEE TRANSACTIONS ONKNOWLEDGE AND DATA ENGINEERING, VOL. 24, NO. 4, APRIL 2012 Fig. 19. Traffic for distributed union. Fig. 20. Latency for distributed union. Fig. 21. Traffic cost when varying the threshold. Fig. 22. Latency when varying the threshold.

[10] T. Suel, C. Mathur, J. wen Wu, J. Zhang, A. Delis,M.Kharrazi,X.Long,and K.Shanmugasundaram, "Odissea: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval," Proc. Int'l Workshop Web and Databases (WebDB), 2003.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. ACM SIGCOMM, 2001.

[12] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," Comm. ACM, vol. 13, no. 7, pp. 422-426, 1971.

[13] B.H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," Comm. ACM, vol. 13, no. 7, pp. 422-426, 1971.

[14] P. Reynolds and A. Vahdat, "Efficient Peer-to- Peer Keyword Searching," Proc. Int'l Conf. Distributed Systems Platforms and Open Distributed Processing (Middleware), 2003.

[15] J. Zhang and T. Suel, "Efficient Query Evaluation on Large Textual Collections in a Peer-to-Peer Environment," Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P), 2005.

[16] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen, "Planetp: Using Gossiping to Build Content Addressable Peer-to- Peer Information Sharing Communities," Proc. IEEE Int'l Symp. High Performance Distributed Computing (HPDC), 2003.

[17] B. Yang and H. Garcia-Molina, "Designing a Super-Peer Network," Proc. Int'l Conf. Data Eng. (ICDE), 2003.

[18] H.T. Shen, Y.F. Shu, and B. Yu, "Efficient Semantic-Based Content Search in P2P Network," IEEE Trans. Knowledge and Data Eng., vol. 16, no.7, pp. 813-826, July 2004.

[19] K. Sripanidkulchai, B. Maggs, and H. Zhang, "Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems," Proc. IEEE INFOCOM, 2003.

[20] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Olko, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C.Tempich, "Bibster - A Semantics-Based Bibliographic Peer-to- Peer System," Proc. Int'l Semantic Web Conf. (ISWC), 2004.

[21] J. Lu and J.P. Callan, "Content-Based Retrieval in Hybrid Peer-to- Peer Networks," Proc. Conf. Information and Knowledge Management (CIKM), 2003.

[22] M. Li, W.-C. Lee, and A. Sivasubramaniam, "Semantic Small World: An Overlay Network for Peer-to-Peer Search," Proc. IEEE Int'l Conf. Network Protocols (ICNP), 2004.

[23] C. Tang and S. Dwarkadas, "Hybrid Global- Local Indexing for Efficient Peer-to-Peer Information Retrieval," Proc. Networked Systems Design and Implementation (NSDI), 2004.

[24] S. Robertson, "Understanding Inverse Document Frequency: On Theoretical Arguments for Idf," J. Documentation, vol. 60, pp. 503- 520, 2004.