Aggregation of a Term Vocabulary for P2P-IR: A DHT Stress Test*

Fabius Klemm and Karl Aberer

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{Fabius.Klemm,Karl.Aberer}@epfl.ch

Abstract. There has been an increasing research interest in developing full-text retrieval based on peer-to-peer (P2P) technology. So far, these research efforts have largely concentrated on efficiently distributing an index. However, ranking of the results retrieved from the index is a crucial part in information retrieval. To determine the relevance of a document to a query, ranking algorithms use collection-wide statistics. Term frequency - inverse document frequency (TF-IDF), for example, is based on frequencies of documents containing a given term in the whole collection. Such global frequencies are not readily available in a distributed system. In this paper, we study the feasibility of aggregating global frequencies for a large term vocabulary in a P2P setting. We use a distributed hash table (DHT) for our analysis. Traditional applications of DHTs, such as file sharing, index keys in the order of tens of thousands. Aggregation of a vocabulary consisting of millions of terms poses extreme requirements to a DHT implementation. We study different aggregation strategies and propose optimizations to DHTs to efficiently process large numbers of keys.

1 Introduction

Performing Information Retrieval (IR) on top of Peer-to-Peer (P2P) systems has become an active research field in recent years. In such systems, the peers organize to jointly build a distributed index. Most of the work in P2P-IR has concentrated on efficiently distributing the index. In [12], for example, the authors use a distributed hash table (DHT) to map keywords to responsible peers for indexing. However, this and similar approaches assume that global statistics of the term vocabulary are available and ready to use for, e.g., calculating top-k results.

Another indexing technique is presented in [15] and is based on CAN [11]. Documents and queries are represented as latent semantic indexing (LSI) vectors in a Cartesian space. This space is mapped into a structured P2P network keeping semantically related indexes co-located. Once again, global statistics of

* The work presented in this paper was carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European FP 6 STREP project ALVIS (**002068**).

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 187–194, 2007.

the term vocabulary, which are necessary to compute the weights for the vectors in the Cartesian space, are assumed to be available.

In this paper we study the aggregation of global statistic of a large term vocabulary using DHTs. Our main contributions are: a) We introduce optimization strategies that improve the performance of a DHT to efficiently handle concurrent insertions of very large numbers of keys. b) We present an analysis of term vocabulary aggregation for Internet-scale full-text retrieval.

Our paper is structured as follows: Section 2 gives a brief overview of structured P2P systems. In section 3 we introduce our optimization strategies for DHTs. Results of some practical experiments are presented in section 4. Sections 5 and 6 finish with discussion and conclusions.

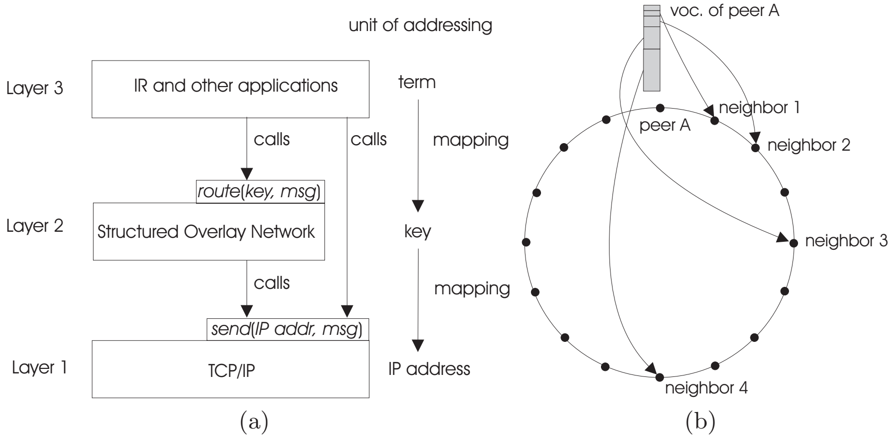

Fig. 1. a) 3-layered architecture b) Insertion of a term vocabulary

2 Overview of Structured P2P Systems

We first provide a short introduction to distributed hash tables (DHTs), also called structured overlay networks. For a clearer presentation, we structure a peer into three layers (figure 1(a)).

The lowest layer provides communication between two peers using TCP/IP. It provides the service *send(IP address, message)*, which sends a message to the peer listening at the given IP address. This service is used by the structured overlay network on layer 2 as well as applications on layer 3.

Layer 2 is the routing layer. It provides the service $route(key, message)$, which routes a message to the peer responsible for the key. It creates and maintains a routing table, which, given a key, determines the IP of the next hop peer for forwarding the message. Therefore, layer 2 provides a key → IP mapping. Most DHTs, such as CHORD, Pastry, or P-Grid [14,13,1] create routing tables of size

$O(log(n))$, where n is the number of peers in the system. The routing entries are chosen in such a way that the resulting graph has small world properties [10,8]. Routing a message between any two peers is then guaranteed to take $O(log(n))$ overlay hops on average.

On layer 3 we have the application that is using the DHT. In our case, it is an IR application, which inserts a local term vocabulary into the DHT using a $route(key, message)$ function provided by layer 2. To perform the mapping of a term to a key layer 3 uses a hash function, which is usually provided by layer 2.

3 Aggregation of Term Vocabulary

This section describes aggregation local document frequencies frequencies to global frequencies. We will first describe the usage scenario and then discuss insertion strategies.

3.1 Usage Scenario

Each peer stores a local document collection. From its local document collection, each peer creates a local term vocabulary. For each term in the local vocabulary, a peer determines the local document frequencies, i.e. the number of documents the term appears in. Each peer inserts its complete vocabulary together with the local frequencies into the DHT. Each term and its local frequency are packed into a message and routed using a key that has been created by hashing the term. The peer responsible for the key receives the message and stores the containing (term, frequency) pair in its local database. For each term, there is exactly one responsible peer. Therefore, for a given term there is one peer that will receive all local frequencies of this term to calculate its global frequency.

Assume a small document collection of 200,000 documents per peer, which has a term vocabulary of about 190,000 terms [1]. All peers concurrently insert their vocabulary into the DHT. We will now present several strategies to handle such a flood of messages and discuss their advantages and disadvantages.

3.2 Blunt Message Handling

When an application calls $route(key, message)$, the straight-forward procedure is to use the routing table to map the key onto the next-hop IP and pass the message to layer 1 to be sent to the next hop. This strategy works fine when only a couple of hundreds to a few thousands of messages have to be inserted and when those messages are reasonably large. However, when inserting a term vocabulary, sending millions of small messages containing only a (term, frequency) pair individually is extremely inefficient. The overhead of message headers is high and message compression is ineffective for small messages.

[1] We computed this value from a sample collection of Reuters news articles available at http://about.reuters.com/researchandstandards/corpus/. The growth of the vocabulary follows Heap's law [9].

3.3 Splitting the Vocabulary into Blocks

Our second strategy optimizes the insertion process by processing (term, frequency) pairs in blocks. Figure 1(b) shows for peer A how the vocabulary is divided into blocks. Most DHTs use randomized hashing to achieve a uniform distribution of keys. In this case, about 1/2 of the terms in the vocabulary maps to peers on the left side of the circle and are therefore sent to neighbor 4 as next hop. 1/4 will be sent to neighbor 3, 1/8 to neighbor 2, 1/16 to neighbor 1, and 1/16 maps to peer A itself. In general, $O(Log(n))$ blocks have to be sent.

This scheme has the following advantages: a) some of the blocks are large enough to be efficiently compressed. b) Shipping few large packets over TCP/IP is faster than shipping many small packets of only a few bytes.

However, in which layer of the architecture should we split the vocabulary into blocks? If we do it in layer 3, it has to know about the key → IP mapping of layer 2, which should be hidden to upper layers. Handing the whole vocabulary down to layer 2 would require making the interface of layer 2 application-specific, which is also not a desirable solution. Therefore, we propose a third strategy, message queueing at layer 1.

3.4 Message Queuing

In this strategy layer 3 and 2 do not have to deal with message packing at all. Messages with single (term, frequency) pairs are handed over to layer 1 with the *send(IP address, message)* function. Layer 1 takes care of efficiently shipping messages to their next-hop IP. To build blocks of messages layer 1 maintains a queue for each outgoing IP address. As the size of the routing table at layer 2 is $O(log(n))$, we also need $O(log(n))$ outgoing IP queues at layer 1. Each queue stores messages according to the following scheme: Each queue has a timer and a threshold. Messages are delayed in the queue until either the threshold is reached or a timeout occurs. A timer is started when a message is inserted into an empty queue. When the message threshold is reached or a timeout occurs all messages in the queue are packed together. This pack of messages is then compressed and sent to the next-hop IP as one large packet.

This approach has the following advantages: a) it is completely hidden to upper layers: many small messages can be inserted into the DHT in bursts and efficiently processed at layer 1. b) It is more flexible compared to approach 2: messages from other peers that have to be forwarded can be packed together with messages originating from the same peer.

The threshold should be set high enough for compression to be effective, but not too high to avoid unnecessary delays. The queue threshold can be defined in either number of message or number of bytes. To avoid that time critical messages, such as queries, are delayed, we added a message flag that specifies whether a message can be delayed. Time-critical messages are instantly sent, irrespective of timeout and threshold.

3.5 Avoiding Flooding

When all peers in the network concurrently start to insert their vocabularies, the network can be flooded and break down if no additional measures are taken. In this subsection, we will present mechanism to avoid such an overload of the system.

Priority queues. As first mechanism to avoid overload we propose using priority queues: We have two types of messages: a) the messages that are already in the system and are travelling (over several overlay hops) to their final destination. b) The messages that are about to be inserted into the system (by an application on layer 3). The first type of messages should have priority over the newly inserted messages. We give higher priorities to messages that have been in the system for a long time. Such messages are close to their final destinations and therefore will soon get out of the system to make space for new messages.

Receiver feedback. The second mechanism is receiver feedback. We are in an environment of heterogeneous peers, i.e. some peers have more processing power than others. It is therefore important to avoid that slower peers are flooded with messages. TCP flow control already does some work, however, cannot fully prevent slower peers from being overloaded. Therefore, we introduced a feedback mechanism on top of TCP. A peer (at layer 1) can forward the next message to the same IP address only after having received an acknowledgement, which the receiver returns after the message has been processed. The delay of this acknowledgement thus depends on the current load of the receiving peer.

4 Experimental Results

We performed experiments in the local EPFL gigabit LAN on 64 SUN Ultra 10 with 360 MHz CPU and 256 MB RAM. The results are for 4, 8, 16, 32, and 64 peers, one peer running per machine. We implemented our DHT in JAVA. Messages are objects, which are serialized and compressed at layer 1. For compression we use the java GZIP classes. The queue threshold is 100 messages and the timeout 1s. Figure 2(a) shows the results of our runs for vocabularies of 96,000 and 192,000 terms per peer.

The first observation we can make is that the execution time to insert all vocabularies is about twice as long for 192K terms as for 96K terms. We therefore conclude that our implementation is stable.

Second, we observe that the insertion time grows considerably slower than the total number of terms: with 192K terms per peer, increasing the total number of terms from $4 * 192K \approx$ 770K to 64 * 192K $\approx$ 12M, a factor of 16, increases the total insertion time from 460s to 1600s, which is a factor of 3.5.

Figure 2(b) shows experiments with varying queue thresholds of 1, 10, and 100 messages. A threshold of 1 means that messages are not queued, i.e. each messages is sent individually. The number of keys inserted by each peer is 19,2K, i.e. 10% of the amount in figure 2(a). With queue threshold t set to 10, the

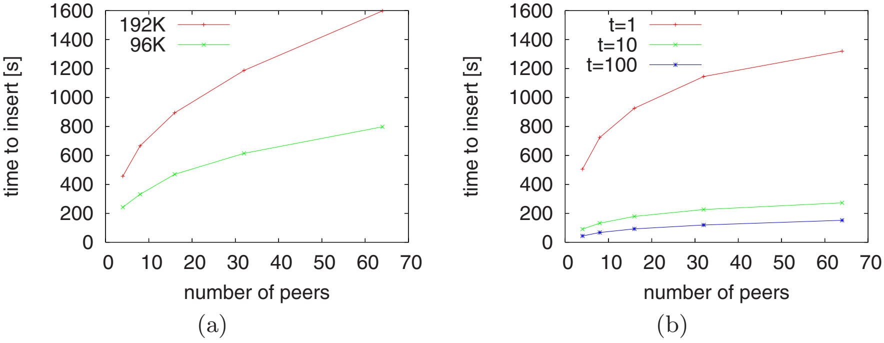

Fig. 2. Insertion times for a) varying voc. sizes and b) varying queue thresholds

insertion time decreases already significantly by 80%. A queue threshold of 100 messages leads to a decrease of 89%. The decrease in bandwidth consumption is 87% for $t = 10$ and 97% for $t = 100$. The reason for this dramatic decrease is that Java produces very large object serializations, which can be very well compressed. Compression of multiple small messages can therefore increase the performance significantly.

5 Discussion

5.1 Redistribution of Aggregates

Once the local term frequencies are aggregated, the global frequencies have to be distributed to interested peers. If all peers are interested in the same vocabulary, we could simply broadcast the global frequencies. Efficient broadcast strategies in DHTs have been presented in recent research papers, such as in [6,7]. However, such an assumption is not realistic in large networks. Another possibility is that aggregates are streamed to interested peers. This could be done using a multicast protocol, such as presented in [4,5]. The integration of such a protocol is part of future work.

5.2 Fighting Malicious Peers

As all peers are allowed to insert term-frequency pairs, some peers could try to insert false values to change ranking to their advantages. Trust in P2P is not the focus of this paper. Nevertheless, we sketch possible solutions: a) There exist environments where all peers in the network are trusted, e.g. when they belong to a company network or a closed P2P group of cooperating universities. Malicious behavior would lead to the exclusion from the group. b) False frequencies that are excessively high could be detected by comparing to former values of the global frequencies of this term. However, a peer could still repeatedly insert small values for the same term to increase its frequency. A solution could be to monitor, which peer is inserting which frequencies to detect malicious pees.

Trust in P2P is a large research area on its own. We believe that our application of DHTs is not fundamentally different from other applications and that solutions in trust management would therefore be applicable.

5.3 Updating Term Frequencies

Local document collections and therefore term vocabularies keep changing over time. It is necessary to update term frequencies. One possible approach might be to re-run the complete aggregation process, e.g. every couple of days or weeks, depending on how fast the local document collections change. Another option might be to instantly insert updates. The responsible peer that aggregates frequencies for a certain term would then have to estimate update rates to calculate approximate global term frequency. Further problems during the aggregation process can arise when peers fail. Handing peer failures and replication of data, however, is orthogonal to this work. We leave improvements in these areas to future work.

5.4 Scaling It Up

In our experiments the size of the local vocabularies was about 200K terms, which corresponds to a collection of roughly 200K documents. The insertion time with 64 peers was approx. 30 min. Let's assume a peer stores 5 million documents. The corresponding term vocabulary would contain about 1 million terms (according to Heap's law [9]) and could therefore be inserted in less than 3 hours with 64 peers (five times the time necessary than for the 200K vocabulary). In a network of 2000 peers the insertion time would be approx. 5 hours. Such a network could thus maintain the global term vocabulary for a collection of 10 billion documents, about the size of the Google index a the time of writing.

6 Conclusions

In this workshop paper we showed that it is possible to aggregate an Internet-scale term vocabulary with P2P technology. This result is important as many P2P-IR systems require global document frequencies of terms for efficient indexing and ranking. We proposed mechanisms to improve standard DHTs to handle very large numbers of messages. These strategies can serve as suggestions for improving existing DHT implementations. Aggregation of a term vocabulary is only one (though very important) possible application that can benefit from our improvements. In principle our techniques using message packing, priority queues, and receiver feedback are necessary for efficiently implementing any distributed application that sends very large numbers of small messages.

As future work, we are planning experiments in a more "hostile" environment than the university Intranet, e.g. in PlanetLab [2]. In such an environment we

[2] www.planet-lab.org

expect more unevenly loaded peers and large variations in network delays as well as peer failures, which will require refinements of our queueing strategies.

References

1. K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Sixth International Conference on Cooperative Information Systems*, 2001.
2. K. Aberer, F. Klemm, M. Rajman, and J. Wu. An Architecture for Peer-to-Peer Information Retrieval. *27th Annual International ACM SIGIR Conference (SIGIR 2004), Workshop on Peer-to-Peer Information Retrieval*, 2004.
3. A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, New York, NY, USA, 2004. ACM Press.
4. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, 2003.
5. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, October 2002.
6. S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *IPTPS*, pages 304–314, 2003.
7. A. Ghodsi, L. O. Alima, S. el Ansary, P. Brand, and S. Haridi. Self-correcting broadcast in distributed hash tables. In *Series on Parallel and Distributed Computing and Systems (PDCS'2003), ACTA Press, Calgary*, 2003.
8. S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. 2005.
9. H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA, 1978.
10. J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. 2001.
12. P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. *Middleware03*, 2003.
13. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
14. 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 ACM SIGCOMM*, 2001.
15. C. Tang, C. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM*, 2003.