On the Feasibility of Peer-to-Peer Web Indexing and Search

Jinyang Li* Boon Thau Loo† Joseph M. Hellerstein† M. Frans Kaashoek*
David R. Karger* Robert Morris*

*MIT Lab for Computer Science †UC Berkeley
jinyang@lcs.mit.edu, {boonloo, jmh}@cs.berkeley.edu, {kaashoek, karger, rtm}@lcs.mit.edu

Abstract

This paper discusses the feasibility of peer-to-peer full-text keyword search of the Web. Two classes of keyword search techniques are in use or have been proposed: flooding of queries over an overlay network (as in Gnutella), and intersection of index lists stored in a distributed hash table. We present a simple feasibility analysis based on the resource constraints and search workload. Our study suggests that the peer-to-peer network does not have enough capacity to make naive use of either of search techniques attractive for Web search. The paper presents a number of existing and novel optimizations for P2P search based on distributed hash tables, estimates their effects on performance, and concludes that in combination these optimizations would bring the problem to within an order of magnitude of feasibility. The paper suggests a number of compromises that might achieve the last order of magnitude.

1 Introduction

Full-text keyword search of the Web is arguably one of the most important Internet applications. It is also a hard problem; Google currently indexes more than 2 billion documents [2], a tiny fraction of the estimated 550 billion documents on the Web [5]. While centralized search engines such as Google work well, peer-to-peer (P2P) Web search is worth studying for the following reasons. First, Web search offers a good stress test for P2P architectures. Second, P2P search might be more resistant then centralized search engines to censoring or manipulated rankings. Third, P2P search might be more robust than centralized search as the demise of a single server or site is unlikely to paralyze the entire search system.

A number of P2P systems provide keyword search, including Gnutella [1] and KaZaA [4]. These systems use the simple and robust technique of flooding queries over some or all peers. The estimated number of documents in these systems is 500 million [3]; documents are typically music files, and searches examine only file meta-data such as title and artist. These systems have performance problems [6] even with workloads much smaller than the Web.

Another class of P2P systems achieve scalability by structuring the data so that it can be found with far less expense than flooding; these are commonly called distributed hash tables (DHTs) [19, 16, 18, 23]. DHTs are well-suited for exact match lookups using unique identifiers, but do not directly support text search. There have been recent proposals for P2P text search [17, 20, 11, 10] over DHTs. The most ambitious known evaluation of such a system [17] demonstrated good full-text keyword search performance with about 100,000 documents. Again, this is a tiny fraction of the size of the Web.

This paper addresses the question *Is P2P Web search likely to work?* The paper first estimates the size of the problem: the size of a Web index and the rate at which people submit Web searches. Then it estimates the magnitude of the two most fundamental resource constraints: the capacity of the Internet and the amount of disk space available on peer hosts. An analysis of the communication costs of naive P2P Web search shows that it would require orders of magnitude more resources than are likely to be available. The paper evaluates a number of existing and novel optimizations, and shows that in combination, they should reduce costs to within an order of magnitude of available resources. Finally, the paper outlines some design compromises that might eliminate the last order of magnitude difference.

The main contribution of this paper is an evaluation of the fundamental costs of, and constraints on, P2P Web search. The paper does not claim to have a definitive answer about whether P2P Web search is likely to work, but it does provide a framework for debating the question.

2 Background

A query consists of a set of *search terms* (words) provided by a user. The result is usually a list of documents that contain the terms, ranked by some scoring mechanism. Search engines typically precompute an *inverted index*: for each word, a *posting list* of the identifiers of documents that contain that word. These postings are intersected in a query involving more than one term. Since the intersection is often large, search engines usually present only the most highly ranked documents. Systems typically combine many ranking factors; these may include the importance of the documents themselves [15], the frequency of the search terms, or how close the terms occur to each other within the documents.

3 Fundamental Constraints

Whether a search algorithm is feasible depends on the workload, the available resources, and the algorithm itself. We estimate the workload first. Google indexes more than 2 billion Web documents [2], so to be conservative we assume 3 billion. Assuming 1000 words per document, an inverted index would contain $3*10^9*1000$ document identifiers (docIDs). In a DHT, a docID would likely be a key with which one could retrieve the document; typically this is a 20-byte hash of the document's content. The large docID space simplifies collision avoidance as different peers independently generate docIDs when inserting documents into the P2P network. The total inverted index size for the Web is about $6 * 10^{13}$ bytes. We assume the system would have to serve about 1000 queries per second (Google's current load [2]).

A P2P search system would have two main resource constraints: *storage* and *bandwidth*. To simplify subsequent discussion, we present concrete estimates based on informed guesses.

- **Storage Constraints:** Each peer host in a P2P system will have a limit on the disk space it can use to store a piece of the index; we assume one gigabyte, a small fraction of the size of a typical PC hard disk. It is not atypical today for some large desktop applications to have an installed size of around 1GB. An inverted index of size $6 * 10^{13}$ bytes would require 60,000 PCs, assuming no compression.

- **Communication Constraints:** A P2P query consumes bandwidth on the wide-area Internet; the total bandwidth consumed by all queries must fit comfortably within the Internet's capacity.

  Given the importance of finding information on the Web, we assume that it is reasonable for it to consume a noticeable fraction of Internet capacity. DNS uses a few percent of wide-area Internet capacity [21]; we optimistically assume that Web search could consume 10%. One way to estimate the Internet's capacity is to look at the backbone cross-section bandwidth. For example, the sum of the bisection bandwidths of Internet backbones in the U.S. was about 100 gigabits in 1999 [7]. Assuming 1,000 queries per second, the per-query communication budget is 10 megabits, or roughly one megabyte. This is a very optimistic assessment.

  Another way to derive a reasonable query communication cost is to assume that the query should send no more data than the size of the document ultimately retrieved. Assuming that the average Web page size is about 10 kilobytes, this leads to a pessimistic query communication budget of 10 kilobytes.

  The rest of this paper assumes the more optimistic budget of one megabyte of communication per query.

4 Basic Cost Analysis

This section outlines the costs of naive implementations of two common P2P text search strategies: *partition-by-document* and *partition-by-keyword*.

4.1 Partition by Document

In this scheme, the documents are divided up among the hosts, and each peer maintains a local inverted index of the documents it is responsible for. Each query must be broadcast or flooded to all peers; each peer returns its most highly ranked document(s). Gnutella and KaZaA use partition by document.

Flooding a query to the 60,000 peers required to hold an index would require about 60,000 packets, each of size 100 bytes. Thus a query's communication cost would be 6 megabytes, or 6 times higher than our budget. Of course, if peers were able to devote more disk space to storing the index, fewer would be required, and the communication cost would be proportionately less.

4.2 Partition by Keyword

In this scheme, responsibility for the words that appear in the document corpus is divided among the peers. Each peer stores the posting list for the word(s) it is responsible for. A DHT would be used to map a word to the peer responsible for it. A number of proposals work this way [17, 10].

A query involving multiple terms requires that the postings for one or more of the terms be sent over the network. For simplicity, this discussion will assume a two-term query. It is cheaper to send the smaller of the two postings to the peer holding the larger posting list; the latter peer would perform the intersection and ranking, and return the few highest-ranking document identifiers.

Analysis of 81,000 queries made to a search engine for `mit.edu` shows that the average query would move 300,000 bytes of postings across the network. 40% of the queries involved just one term, 35% two, and 25% three or more. `mit.edu` has 1.7 million Web pages; scaling to the size of the Web (3 billion pages) suggests that the average query might require 530 megabytes, requiring a factor of $\mathbf{530}\times$ improvement.

Some queries, however, are much more expensive than this average. Consider a search for "the who". Google reports that $3 * 10^9$ documents contain "the", and $2 * 10^8$ contain "who". This query would send 4 GB over the network, exceeding our budget by $\mathbf{4000}\times$.

Our analysis seems to imply that partition-by-document is the more promising scheme, requiring only a factor of $6\times$ improvement. However, we focus instead on partition-by-keyword because it allows us to draw on decades of existing research on fast inverted index intersection. As we will see later, by applying a variety of techniques we can bring the partition-by-keyword scheme to the same order-of-magnitude bandwidth consumption as the partition-by-document approach.

5 Optimizations

In this section, we discuss optimization techniques for partition-by-keyword. We evaluate the optimizations using our data set of 81,000 `mit.edu` queries and 1.7 million web pages crawled from MIT.

5.1 Caching and Precomputation

Peers could cache the posting lists sent to them for each query, hoping to avoid receiving them again for future queries. This technique reduces the average query communication cost in the MIT query trace by 38%. The modest improvement can be attributed to the fact that many queries appear only once in the trace.

Precomputation involves computing and storing the intersection of different posting lists in advance. Precomputing for all term pairs is not feasible as it would increase the size of the inverted index significantly. Since the popularity of query terms follows a Zipf distribution [17], it is effective to precompute only the intersections of all pairs of popular query terms. If 7.5 million term pairs (3% of all possible term pairs) from the most popular terms are precomputed for the MIT data set, the average query communication cost is reduced by 50%.

5.2 Compression

Compression provides the greatest reduction in communication cost without sacrificing result quality.

5.2.1 Bloom Filters

A Bloom filter can represent a set compactly, at the cost of a small probability of false positives. In a simple two-round Bloom intersection [17], one node sends the Bloom filter of its posting list. The receiving node intersects the Bloom filter and its posting list, and sends back the resulting list of docIDs. The original sender then filters out false positives. The result is a compression ratio of 13 [1].

When the result set is small, we propose multiple rounds of Bloom intersections. In this case, the

[1] This is a best case compression ratio which assumes that the intersection is empty and that the two posting lists have similar sizes.

compression ratio is increased to 40 with four rounds of Bloom filter exchange [2]. Compressed Bloom filters [14] give a further 30% improvement, resulting in a compression ratio of approximately 50.

5.2.2 Gap Compression

Gap compression [22] is effective when the gaps between sorted docIDs in a posting list are small. To reduce the gap size, we propose to periodically remap docIDs from 160-bit hashes to dense numbers from 1 to the number of documents. In the MIT data set, gap compression with dense IDs achieves an average compression ratio of 30. Gap compression has the added advantage over Bloom filters as it incurs no extra round-trip time, and the compression ratio is independent of the size of the final intersection.

5.2.3 Adaptive Set Intersection

Adaptive set intersection [8] exploits structure in the posting lists to avoid having to transfer entire lists. For example, the intersection $\{1, 3, 4, 7\} \cap \{8, 10, 20, 30\}$ requires only one element exchange, as $7 < 8$ implies an empty intersection. In contrast, computing the intersection $\{1, 4, 8, 20\} \cap \{3, 7, 10, 30\}$ requires an entire posting list to be transferred.

Adaptive set intersection can be used in conjunction with gap compression. Based on the MIT data set, an upper bound of 30% improvement could be achieved on top of gap compression, resulting in a compression ratio of 40.

5.2.4 Clustering

Gap compression and adaptive set intersection are most effective when the docIDs in the posting lists are "bursty". We utilize statistical clustering techniques to group similar documents together based on their term occurrences. By assigning adjacent docIDs to similar documents, the posting lists are made burstier. We use Probabilistic Latent Semantic Analysis (PLSA) [13] to group all the MIT Web documents into 100 clusters. Documents within the same cluster are assigned contiguous docIDs. Clustering improves the compression ratio of adaptive set intersection with gap compression to 75.

[2]More than 4 rounds yield little further improvement.

Technique	Improvement
Caching	$1.5\times$
Precomputation	$2\times$
Bloom Filters	$50\times$
Gap Compression (GC)	$30\times$
Adaptive Set (AS) + GC	$40\times$
Clustering + GC + AS	$75\times$

Table 1: Optimization Techniques and Improvements

6 Compromises

Table 1 summarizes the performance gains of different techniques proposed so far. The most promising set of techniques result in a $75\times$ reduction in average communication costs. However, achieving this improvement would require distributed renumbering and clustering algorithms which are rather complex. Even a $75\times$ reduction leaves the average query communication cost an order of magnitude higher than our budget. An extra $7\times$ improvement is still needed. This leads us into the softer realm of accepting compromises to gain performance.

6.1 Compromising Result Quality

Reynolds and Vahdat suggest streaming results to users using *incremental intersection* [17]. Assuming users are usually satisfied with only a partial set of matching results, this will allow savings in communication as users are likely to terminate their queries early. Incremental intersection is most effective when the intersection is big relative to the postings so that a significant number of matching results can be generated without needing to transfer an entire posting list [3].

While incremental results are useful, the likelihood that users will terminate their queries early will be increased if the incremental results are prioritized based on a good ranking function. To achieve this effect, Fagin's algorithm (FA) [9] is used in conjunction with a ranking function to generate incremental ranked results. The posting lists are sorted based on the ranking function, and the top ranked docIDs are incrementally transferred from one node to another for intersection. Unfortunately, not all ranking func-

[3]This suggests that it might be preferable to precompute term pairs with big posting lists and small intersections, which would also reduce the storage overhead of precomputation.

tions are applicable. Examples of applicable ranking functions include those based on PageRank, term frequencies or font sizes. An example of a ranking function that can *not* be used with FA is one based on proximity of query terms. By limiting the choices of useful ranking functions, we are left with incremental results that are not as well-ranked compared to the results of commercial search engines. To alleviate this shortcoming, we propose the use of mid-query relevance feedback [12] that allows users to control and change the order in which the posting list intersections are performed. This leads to potential improvements in user experiences, and may result in earlier query termination. However, incorporating user feedback in the middle of a search query introduces a number of challenges in designing appropriate result browsing and feedback interfaces.

As we mentioned earlier, incremental intersection results are more effective when the final result set is big relative to the intersecting posting lists. To illustrate, consider two posting lists X and Y, and the corresponding intersection Z where $|X| > |Y| > |Z|$. Computing 10 matching results will require transferring an average of $\frac{10*|Y|}{|Z|}$ elements from the smaller posting list Y. We quantify the savings of incremental results based on the MIT data set. On average, computing 10 results using incremental intersection results in a $50\times$ reduction in communication cost [4]. We would expect even greater performance gains for the larger Web corpus. The savings of incremental intersection is especially significant for expensive queries such as "the who". Google reports that there are 10^7 results, hence roughly $\frac{2*10^8}{10^7} * 10 = 200$ docIDs need to be shipped to retrieve the top 10 ranked documents containing "the" and "who". This reduces the communication cost significantly to $200 * 20B \approx 4KB$ which is well within our budget of one megabyte per query.

6.2 Compromising P2P Structure

The one megabyte communication budget is derived from the bisection backbone bandwidth of the Internet. The *aggregate* bandwidth summed over all links is probably much larger than the bisection. We could compromise the P2P network structure to exploit Internet aggregate bandwidth for better performance. One proposal is to replicate the entire inverted index, with one copy per ISP. As a rough analysis, if the entire inverted index can be replicated at 10 ISPs, there is a $10\times$ increase in the communication budget per query.

[4] Incremental ranked intersection can be combined with compression, but unfortunately the compression ratio will be reduced as a result.

7 Conclusion

This paper highlights the challenges faced in building a P2P web search engine. Our main contribution lies in conducting a feasibility analysis for P2P Web search. We have shown that that naive implementations of P2P Web search are not feasible, and have mapped out some possible optimizations. The most effective optimizations bring the problem to within an order of magnitude of feasibility. We have also proposed two possible compromises, one in the quality of results, and the other in the P2P structure of our system. A combination of optimizations and compromises will bring us within feasibility range for P2P Web search.

8 Acknowledgments

This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

References

[1] Gnutella. `http://gnutella.wego.com`.

[2] Google Press Center: Technical Highlights. `http://www.google.com/press/highlights.html`.

[3] Ingram: Record Industry Plays Hardball with Kazaa. `http://www.globeandmail.com/`.

[4] Kazaa. `http://www.kazza.com`.

[5] The Deep Web: Surfacing Hidden Value. `http://www.press.umich.edu/jep/07-01/bergman.html`.

[6] Why Gnutella Can't Scale. No, Really. `http://www.darkridge.com/~jpr5/doc/gnutella.html`.

[7] Boardwatch Magazine's Directory of Internet Service Providers, 1999.

[8] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive Set Intersections, Unions, and Differences. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, January 2000.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Symposium on Principles of Database Systems*, 2001.

[10] O. D. Gnawali. A Keyword Set Search System for Peer-to-Peer Networks. Master's thesis, Massachusetts Institute of Technology, June 2002.

[11] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, March 2002.

[12] J. M. Hellerstein, R. Avnur, A. Chou, C. Olston, V. Raman, T. Roth, C. Hidber, and P. Haas. Interactive Data Analysis with CONTROL. In *IEEE Computer*, 1999.

[13] T. Hofmann. Probabilistic Latent Semantic Analysis. In *Proc. of Uncertainty in Artificial Intelligence, UAI'99*, Stockholm, 1999.

[14] M. Mitzenmacher. Compressed Bloom Filters. In *Twentieth ACM Symposium on Principles of Distributed Computing*, August 2001.

[15] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of the 2001 ACM SIGCOM Conference*, Berkeley, CA, August 2001.

[17] P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. In *Unpublished Manuscript*, June 2002.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218, 2001.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[20] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In *HotNets-I*, October 2002.

[21] K. Thompson, G. Miller, and R. Wilder. Wide-area Traffic Patterns and Characteristics. In *IEEE Network, vol. 11, no. 6, pp. 10-23*, November/December 1997.

[22] I. H. Witten, A. Moffat, and T. C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. May 1999.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.