Appears in the 12^{th} International Symposium on High Performance Distributed Computing

Distributed Pagerank for P2P Systems

Karthikeyan Sankaralingam Simha Sethumadhavan James C. Browne

Department of Computer Sciences
The University of Texas at Austin
(karu,simha,browne)@cs.utexas.edu

Abstract

This paper defines and describes a fully distributed implementation of Google's highly effective Pagerank algorithm, for "peer to peer"(P2P) systems. The implementation is based on chaotic (asynchronous) iterative solution of linear systems. The P2P implementation also enables incremental computation of pageranks as new documents are entered into or deleted from the network. Incremental update enables continuously accurate pageranks whereas the currently centralized web crawl and computation over Internet documents requires several days. This suggests possible applicability of the distributed algorithm to pagerank computations as a replacement for the centralized web crawler based implementation for Internet documents. A complete solution of the distributed pagerank computation for an in-place network converges rapidly (1% accuracy in 10 iterations) for large systems although the time for an iteration may be long. The incremental computation resulting from addition of a single document converges extremely rapidly, typically requiring update path lengths of under 15 nodes even for large networks and very accurate solutions.

This implementation of Pagerank provides a uniform ranking scheme for documents in P2P systems, and its integration with P2P keyword search provides one solution to the network traffic problems engendered by return of document hits. In basic P2P keyword search, all the document hits must be returned to the querying node causing large network traffic. An incremental keyword search algorithm for P2P keyword search where document hits are sorted by pagerank, and incrementally returned to the querying node is proposed and evaluated. Integration of this algorithm into P2P keyword search can produce dramatic benefit both in terms of effectiveness for users and decrease in network traffic. The incremental search algorithm provided approximately a ten-fold reduction in network traffic for two-word and three-word queries.

1 Introduction

"Peer to peer" (P2P) networks [18, 20, 22] are emerging as potentially important structures for information and content management and distribution. Effective keyword search is a crucial method for information and content management. Recent papers on P2P implementations of keyword search for documents [8, 15, 19] have called attention to the need for a document ranking system for documents stored in such network systems. In the absence of a ranking system, a P2P keyword search must return all of the qualified documents to the node which initiated the query. The amount of traffic generated by a typical search would flood the network. It seems probable that P2P keyword search will not be practical in the absence of an effective P2P implementable document ranking algorithm. However, a crawler based centralized server approach for computation of such ranks is incompatible with a P2P implementation of keyword search.

Google's *pagerank* computation [11, 17] and other relevance rankings for web pages have made Internet keyword searches orders of magnitude more effective. Google's pagerank algorithm reduces to computation of the eigenvectors of a very large matrix which represents document linkages of web pages. Google uses a web crawler which traverses the Internet tabulating links among documents, returning the link structure to a central server which generates a graph of this link structure. The ranks of the documents are obtained as the eigenvalues of a sparse matrix representing the graph, using an iterative numerical method on a computational server. This process of explicitly generating and then solving of this very large (order three billion or so [16]) matrix takes days [1]. A Google search can return just the pages assessed as most relevant to the search by ranking the results based on the computed pagerank, fetching additional pages incrementally as required.

This paper reports on a method for generating pageranks where the matrix representing the relationships among the documents is never explicitly generated and the solution of the matrix is distributed across all of the nodes which hold the documents. The method is developed in the context of keyword search in P2P networks but could be implemented as an Internet service without the formal structure of a P2P network.

This paper defines and describes a fully distributed, “peer to peer” equivalent of Google’s highly effective pagerank algorithm and provides promising results on the performance of the distributed implementation. The solution phase of the pagerank computation is based on chaotic (asynchronous) iterative solution of linear systems [5]. A complete initial computation of a set of pageranks converges rapidly (1% accuracy in 10 iterations) for large systems. The algorithm enables incremental computation of pageranks as new documents are entered into the network. Incremental updates mean that pagerank computations are continuously updated as compared to the weekly and monthly updates characteristic of centralized pagerank computations. By using distributed computation on the P2P system itself, the need for large computational server complexes is eliminated.

For keyword searches in P2P networks, we define an approach based on incremental return of documents with highest rank, and evaluate its effectiveness. Integration of the distributed pagerank algorithm into P2P implementations of keyword search, along with incremental return of documents, produces dramatic benefit both in terms of effectiveness for users, and decrease in network traffic engendered by document transfer. This scheme has the potential to make keyword search on P2P systems as efficient as Internet keyword search.

The rest of the paper is organized as follows: Section 2 defines and describes a static version of the algorithm. In Section 3 we describe caching of document locations and protocols for joining and leaving for both peers and documents. Section 4 provides a detailed evaluation of the proposed distributed pagerank algorithm. Section 5 briefly discusses centralized alternatives to the distributed algorithm. Section 6 describes future directions, Section 7 discusses related work, and Section 8 concludes.

2 Static Algorithm Design

In this section, we describe the following: document link structure, formulation of queries and the static version of the distributed formulation of the pagerank algorithm where pageranks are computed for an “in-place” network.

2.1 Documents and Links

Hyperlinks are the fundamental structuring elements for Internet documents (web pages) and are the core data used in Google’s computation of pageranks. Files (documents) stored in peer to peer file systems also have an equivalent link structure. The documents make references to other documents in the file system. In systems with bounded search such as CAN [18], Pastry [20] or Chord [22] the GUID (Global Unique Identifier) implements a pointer to each document. All these systems are distributed hash table (DHT) based systems. Systems like Freenet [7] while providing no bounded search guarantees, also maintain a unique identifier, the SSK (a subspace key). Throughout this paper we focus on DHT based systems, but indicate how the mechanisms proposed can be extended for Freenet like systems. For specifying document link structure an HTML-like syntax is assumed. The underlying assumption is that the document was originally an HTML document. Documents on Freenet already use such a syntax with Freenet browsing possible using a web browser with the localhost:8888 FProxy.

2.2 Google’s Formulation of Pageranks

The Google Pagerank algorithm assigns a numeric rank to every document in a system which denotes its importance. 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. Formally the pagerank is defined as:

$$P(a) = (1-d) + d * \sum_{\forall \text{ inlinks i}} P(i)/C(i) \qquad (1)$$

where, $P(i)$ is the pagerank of a document i, $C(i)$ is the number of outbound links out of any page i, and d is a damping factor. Due to lack of space, we restrict the treatment of the pagerank algorithm to this brief discussion. More details are found in [11, 17].

In a matrix form, Equation 1 can be rewritten as:

$$R = d(AR + D) \qquad (2)$$

where the matrix A is an N*N link matrix, R is a column vector of length N representing the ranks of pages, D is a constant column vector(=$(1-d)/d$). It can be shown that, the solution for R corresponds to the eigenvectors of $(A + D \times I)$. The formulation shown in (3) is an iterative solution to (2).

$$R_{i+1} = d * AR_i + D \qquad (3)$$

In the Google pagerank system implemented for Internet web pages, a crawler first crawls all the web pages on the Internet and writes them to a central database. This database is then examined to extract the link structure from which the A matrix can be obtained. The ranks are calculated by a central server as the dominant eigenvector of $(A + D \times I)$. Page et al. [17] showed formally that this algorithm converges.

2.3 Distributed Pagerank Formulation

In a P2P system, different documents reside on different peers. The links in a document can point to documents on other peers[1]. The distributed algorithm for computing the pagerank for documents on P2P systems is as follows:

1. Every peer initializes all its documents with an initial pagerank.
2. Each peer sends a pagerank update message to all the documents to which the documents it stores are linked(out-links). Pageranks of nodes present in the same peer are updated without need for network update messages.
3. Upon receiving an update message for a document, the receiving peer updates the document's pagerank.
4. Update of pagerank of a document in the system results in generation of further pagerank update messages for all out-links.
5. The difference between successive pageranks of a particular document is used as a measure of convergence and when this absolute difference falls below an error threshold ϵ, no more update messages are sent for that document.

Different documents will attain their final pagerank at different times. The algorithm is given in pseudo-code in Figure 1.

This algorithm is essentially a distributed "peer to peer" implementation of a chaotic (asynchronous) iterations linear equation solver, with the peers acting as simple state machines exchanging messages. Asynchronous iteration is the natural algorithm for solving linear systems on distributed networks since no central control or global synchronization is required, and each peer executes the same process. There is substantial literature on mathematical properties of chaotic interactions. As shown in 1969 by Chazan and Miranker in their Asynchronous/Chaotic Iterations paper [5], such systems can be proven to converge, albeit, typically rather slowly.

On P2P systems which maintain only references (just the GUIDs for example) and not the full documents, the documents need to be first fetched by the peers, the references of the outlinks extracted from the contents, and stored along with the reference of the document (the documents themselves need not be saved). A second issue is replication and document caching that some P2P systems use to reduce retrieval time. On such systems, for the distributed pagerank computation to work accurately, pointers need to be maintained at document sources to point to cached copies, so that

[1]Throughout this document we use the term peer or peer-node to refer to a peer computer on the P2P system. And document or node to refer to documents on the P2P system.

```
/* initialize */
Set all pageranks in all nodes to an
initial value;

At time = 0:
Concurrently on all peers:

for all documents in this peer {
   compute newrank based on inlinks;
   relerr = abs(oldrank - newrank)/newrank;
   if (relerr > epsilon) {
     a) send pagerank update message to
        out-links on other peers;
     b) weights of out-links in same peer
        updated;
   }
}
/* first pass is done */
/* every peer listens for pagerank
update messages */

while pagerank update message received {
   recompute newrank based on message;
   relerr = abs(oldrank - newrank)/newrank;
   if (relerror > epsilon) {
     a) send pagerank update message to
        out-links on other peers;
     b) weights of out-links in same peer
        updated;
   }
}
```

Figure 1. Distributed Pagerank. epsilon is a user defined error threshold used for testing convergence.

all copies of the document can contain the correct computed pagerank.

2.4 Integration with Peer to Peer Search Algorithms

The exact method of use of the pagerank with search algorithms varies based on the underlying P2P layer. This section sketches briefly how the pagerank algorithm might be utilized in a keyword search system in FASD/Freenet-like systems, and DHT based systems like CAN, Pastry, and Chord. We then propose an incremental search algorithm that can used for efficient search using the pagerank on DHT based systems.

2.4.1 FASD/Freenet

In FASD, a metadata key representing the document as a vector is associated with every document. These metadata keys are stored in a distributed manner. Search queries are also represented as vectors and documents that match a query are "close" to the search vector. We make a modification to the original FASD algorithm to incorporate pagerank into the search scheme. Results are forwarded based on a linear combination of document closeness and pagerank. Details can be found in [21].

2.4.2 DHT systems

In this paper, we focus on computing the pageranks on DHT based systems and coupling it with keyword search algorithms. Keyword search on DHT based systems is typically implemented by using a distributed index, with the index entry for each keyword pointing to all documents containing that particular keyword. We propose adding an extra entry in the index to store the pageranks for documents. When the pagerank has been computed for a node, an index update message is sent, and the pagerank is noted in the index. When a search is performed the pagerank is used for relevance sorting.

Boolean multi-word queries are inefficient on DHT based systems, because documents ids of all the hits for the keywords need to be passed from peer to peer for each word in the query. To avoid this excessive traffic Bloom filter [3] based solutions have been proposed [19].

2.4.3 Incremental Searching

We propose incremental searching which addresses the issue of network traffic caused by multi-word queries, by incrementally returning documents sorted by pagerank. The search scheme works as follows: When a boolean multi-word query is received, the first term in the query is examined and is routed to the peer which owns the part of the index that contains this term. The index is accessed and a list of documents that the index entry points to are examined and sorted by the pagerank. Instead of forwarding all these documents to the peer responsible for the next term, only the top x% of hits are forwarded. So, the next peer receives only a small fraction, albeit *encompassing the most important documents*. The second peer also follows the same procedure. It finds the documents that match the second term and performs the boolean operation on the two sets of documents. The resulting set is again sorted by pagerank and the top x% of hits are forwarded to the next peer. This procedure is repeated for each term in the query.

In practice we found that this approach provided an order of magnitude reduction in traffic. Furthermore, this approach can be coupled with a Bloom filter based method [19] to provide further reduction in traffic.

3 Extension to Dynamic Systems

In this section, we extend the distributed pagerank computation to handle dynamic behavior. The following two basic dynamic behaviors are handled — a) documents and peers entering and leaving the system and b) caching. It will be shown that after an initial convergence the pageranks can be incrementally updated as peers and documents get added and deleted.

3.1 Entering and Leaving Protocols

So far we described our system assuming all peers are alive at all times. In a P2P system this is rarely the case. Furthermore, documents are regularly added and deleted. Logically peers joining and leaving, and documents being added and deleted are identical — except for the fact that when peers join and leave a large set of documents appear and disappear respectively. The other difference being that when a document is deleted it is gone forever, whereas when a peer leaves it is likely to rejoin the network at a later time.

Peer leaves and joins: When peers leave the P2P system, they take away with them (until they reappear) all their documents. This transient behavior could possibly result in pagerank updates to documents in unavailable peers being lost forever. To avoid this, when a peer is detected as unavailable, update messages are stored at the sender and periodically resent until delivered successfully. In the worst case, the amount of state saved scales linearly with the sum of outlinks in all documents in a peer.

Document inserts and deletes: When a new document is inserted into the network, its pagerank is initialized to some fixed constant value and update messages to its outlinks are sent; it is thus immediately integrated into the distributed pagerank computation scheme. Similarly, when a document is removed, a pagerank update message is sent with the value of the pagerank negated. When the pagerank update message is received by the outlinked documents they update their pageranks and the system eventually reconverges.

3.2 Caching

On DHT based systems with no specific anonymity guarantees the network traffic generated from the pagerank update messages can be reduced by caching IP addresses of peers. When the first pagerank update message is sent for a document, the P2P layer's routing mechanism is used to find the location of the document. Once its location has been found the IP address is cached at the source node, and subsequent update messages can be exchanged directly between source and destination. Storage requirement for this scheme scales linearly with the sum of the outlinks in all documents in a peer.

Graph size (in 1000s)	# of passes		
	% of peers present		
	100	75	50
10	74	134	166
100	88	137	196
500	118	139	196
5000	120	141	241

Table 1. Convergence rate of the distributed pagerank algorithm for 500 peers. Error threshold of 10^{-4}.

On Freenet, in order to honor anonymity guarantees IP addresses cannot be cached in such a manner, and every update message has to be individually routed through intermediate nodes.

4 Evaluation

In this section, we evaluate the distributed pagerank algorithm proposed. There are three key components to be modeled — the graph structure of the document links, the underlying P2P system, and the search queries. In the subsequent sections, we describe the modeling of document link structure, details of our simulation infrastructure and discuss results under the following categories — 1) convergence rate, 2) quality of the pagerank, 3) amount of pagerank message traffic generated, 4) execution time, 5) the quantitative effects of document insertion and deletion, and 6) the traffic reduction obtained from using the pagerank in search queries.

4.1 Graph Structure

Broder et al. [4] studied the graph structure of Internet documents by performing a crawl of approximately 200 million nodes. They concluded that the link structure is similar to small world networks i.e. the number of nodes with degree i is proportional to $1/i^k$. They numerically estimated k for in-degree and out-degree and determined it to be 2.1 and 2.4 respectively. We hypothesize that files on P2P storage systems will show similar link structure, and we synthesized graphs based on this model with 10,000, 100,000, 500,000 and 5 million nodes for our experiments - each node representing a document. We use only the link structure among the documents.

4.2 Distributed Computation

Our simulation methodology is explained below. First the graph representing the documents is constructed as described in the previous subsection. Each document in the graph is then randomly assigned to a peer. We model the logic of distributed computation of the pagerank assuming an underlying DHT based P2P system. The computation of the pagerank is done as follows: concurrently all peers compute the pagerank using the available pagerank for in-links (from the previous iteration). Once all pageranks have been computed, we assume that pagerank messages are sent and received instantaneously and all peers start their next iteration concurrently. In between such passes, sets of peers randomly leave and join the network. Multiple iterations are performed until the computation converges. The computation converges when the error (absolute difference between successive values of the pagerank of a document), in all the documents is less than the error threshold defined in Section 2.3. This is a very strong convergence criterion. Network latency effects, message routing, and other system overheads are not modeled in the simulation. The experiments in subsections 4.3 through 4.7 simulate 500 peers.

4.3 Convergence

Table 1 shows the number of passes required for convergence for the four graph sizes we examined using an error threshold $\epsilon = 10^{-4}$ for checking convergence, with the nodes distributed on 500 peers in the P2P system. When all peers are present (column 2), the number of passes for convergence is of the order of 100. Convergence rate grows slowly with the problem size — a factor of 500 increase in the graph size (10k to 5000k) increases number of passes by only 60%.

To test the quality of the pagerank, we computed the pageranks using a conventional synchronous iterative solver and compared the error between the pagerank from our distributed asynchronous scheme(P_d) and the pagerank from the conventional approach(P_c). In practice, the pagerank, P_d converges to within 0.1% of P_c in as few as 30 passes. Furthermore, we observed that for all the graphs, more than 99% of the nodes converged to within 1% of P_c in less than 10 passes. A more detailed analysis of the pagerank quality is given in Section 4.4.

Dynamic effects: To evaluate the effect of peer joins and leaves, we simulate a fixed fraction of randomly selected nodes leaving and joining the network at the end of every iteration. In Table 1, in columns 3 and 4, we show the results when only three quarters of the peers and half of the peers are available at any given time. The algorithm converges in the presence of these transient effects, albeit at a slower rate. With only half the peers present at any given time only a factor of two slowdown is seen in the convergence rate for all the graph sizes.

	Threshold						
	0.2	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
	Scale						
	10^{-3}	10^{-4}	10^{-5}	10^{-5}	10^{-6}	10^{-6}	10^{-6}
% pages	Relative error for 10k nodes						
50	0.9	4.4	4.5	0.6	1.9	1.1	0.7
75	2.0	10.3	10.7	1.2	3.0	1.7	1.2
90	4.0	21.0	21.6	2.4	4.4	2.3	1.9
99	11.8	66.4	74.0	7.9	8.3	4.5	3.9
99.9	29.9	164.9	195.4	20.3	19.0	5.9	4.9
Max.	102.6	478.1	1166.4	64.1	41.6	6.5	5.9
Avg.	1.7	8.9	9.5	1.1	2.3	1.2	0.9
Relative error for 100k nodes							
50	5.3	27.5	27.6	2.9	4.2	1.5	0.9
75	16.2	86.9	87.3	9.0	9.9	2.4	1.5
90	39.2	212.9	216.3	22.7	23.7	3.8	2.2
99	128.4	717.9	773.0	80.0	85.9	9.1	3.9
99.9	275.0	1589.9	1946.9	216.3	225.6	23.8	5.3
Max.	663.3	4579.9	12714.1	777.7	5202.2	300.8	12.5
Avg.	14.9	81.5	84.4	8.9	10.1	1.9	1.1
Relative error for 500k nodes							
50	0.0	0.1	0.3	0.2	1.5	1.1	1.0
75	0.1	0.7	1.5	0.5	2.5	1.9	1.8
90	0.6	3.4	5.8	1.2	3.5	2.7	2.6
99	4.1	25.6	38.6	6.5	8.6	4.1	3.4
99.9	11.4	74.8	116.7	15.9	19.5	5.7	4.7
Max.	98.2	604.9	849.4	79.6	98.8	9.5	6.2
Avg.	0.3	1.6	2.7	0.6	1.9	1.3	1.2
Relative error for 5000k nodes							
50	1.6	14.1	0.5	0.4	2.8	2.2	1.3
75	11.4	79.7	2.2	1.3	7.4	3.9	2.2
90	43.1	258.9	3.5	2.0	11.5	6.4	2.9
99	345.7	2591.2	7.5	3.0	17.5	10.1	4.2
99.9	1091.2	10912.4	21.3	4.3	21.9	12.9	5.4
Max.	1353.0	12204.9	1241.0	68.5	190.0	24.6	7.8
Avg.	21.5	158.4	1.4	0.8	4.7	2.8	1.5

Table 2. Relative error distribution with different error thresholds ϵ. Relative error = $(P_d - P_c)/P_c$. Relative error in table is *not* expressed as a percentage. Scale shown in row 4.

Threshold 0.2

- 10k graphs: only 10 nodes have error $\geq 3\%$, max 10%
- 100k graph: only 100 nodes have error $\geq 27\%$, max 66%
- 500k graph: only 500 nodes have error $\geq 1\%$, max 10%
- 5000k graph: only 50,000 nodes have error $\geq 35\%$,
  only 5000 nodes have error $\geq 109\%$, max 135%

Threshold 0.001

- 10k graphs: only 10 nodes have error $\geq 0.02\%$, max 0.06%
- 100k graph: only 100 nodes have error $\geq 0.2\%$, max 0.7%
- 500k graph: only 500 nodes have error $\geq 0.02\%$, max 0.08%
- 5000k graph: only 5000 nodes have error $\geq 0.004\%$, max 0.07

Threshold	Number of Messages								Exec. Time (hrs)	
	Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.	32 KB/sec	200 KB/sec
	10k nodes		100k nodes		500k nodes		5000k nodes			
0.2	0.35	35	3.80	38	29.97	60	169.1	33.8	33.7	5.4
10^{-1}	0.39	39	4.16	42	31.14	62	183.8	36.7	36.7	5.8
10^{-2}	0.51	51	5.36	54	35.48	71	395.0	79.0	78.8	12.6
10^{-3}	0.63	63	6.53	65	39.87	80	440.3	88.1	87.9	14.1
10^{-4}	0.75	75	7.74	77	44.36	89	485.2	97.1	96.8	15.5
10^{-5}	0.90	90	9.06	91	48.84	98	533.2	106.6	106	17.0
10^{-6}	1.11	111	10.54	105	52.84	106	586	117.21	117	18.7

Table 3. Variation of message traffic with error threshold ϵ. Total number of messages shown in millions. Execution time for convergence shown in hours for 32Kbytes/sec and 200Kbytes/sec networks.

4.4 Quality of Pagerank

We now examine the quality of the pageranks generated by the distributed pagerank computation. The quality of the pageranks is characterized by the relative error in pagerank: $(P_d - P_c)/P_c$. Higher quality pageranks have lower relative error and are produced by using a lower error threshold for convergence. However, the primary disadvantage of having a low threshold is the increase in the number of pagerank update messages (and therefore execution time). Hence, reduction in network traffic and quality of pagerank are opposing goals. To accurately characterize the relationship between the error threshold and pagerank quality we simulated the distributed pagerank scheme for different graph sizes and for different error thresholds. We report the following statistical indicators: 1) the maximum error in pagerank when using different thresholds, 2) average error across all the documents, and 3) distribution of relative error across the document set.

In Table 2, the distribution of the error across the different documents is shown for the different thresholds. We examined threshold values of 0.2 and 10^{-1} through 10^{-6} for the four graphs. In the table we show collective data for 50%, 75%, 90%, 99% and 99.9% of the pages. The last two lines indicate the maximum relative error and average relative error respectively. The first column lists the different percentages. Columns 2 through 8 indicate the maximum error for that percentage of pages. Note that each of the columns has the error reported in a different scale, which is indicated in the fourth row in the table headings. Looking at the table, we can see for example that, with a threshold of 0.2, up to 50% of the pages in the 10k graph had a relative error of less than $0.9 * 10^{-3}$, up to 99.9% of the pages had a relative error of less than $29.9 * 10^{-3}$, indicating less than 10 pages had more than 3% relative error.

From the error distribution table we can see that a threshold as high 0.2 performs extremely well, producing extremely good quality pageranks for most of the pages (up to 99.9%). Examining the values down any particular column we can see that the pageranks of most documents is extremely close to P_c, even with moderately high thresholds. Examining the values for the four different graph sizes we see that the trends hold independent of graph size. A threshold of 10^{-3}, produces extremely good results for all graph sizes. A summary of the error results in Table 2 is shown to the left of the table. The quality of the pageranks achievable in a huge set of the documents, even with high error thresholds is remarkable.

4.5 Message traffic

We now examine the number of pagerank update messages generated during the pagerank computation. Using lower error thresholds for convergence produces higher message traffic. In Table 3, the number of pagerank update messages generated for different error thresholds is shown for the four graphs. Columns 2, 4, 6, and 8 show the total number of pagerank update messages in millions, generated for convergence, and columns 3, 5, 7, and 9 show the average number of pagerank messages per node - this is obtained by dividing columns 2, 4, 6, and 8 by the corresponding graph size. The average *number of messages per node* is a graph size independent metric of measuring message traffic.

From the Table, it can be seen that the increase in the message traffic with the threshold is approximately logarithmic. As the threshold decreases from 10^{-1} to 10^{-6}, a factor of 10^5, the message traffic increases by less than a factor of 3. The message traffic per node is largely independent of the graph size. This suggests the scalability of the algorithm to large problem sizes.

4.6 Execution Time

The total execution time to convergence for the distributed pagerank scheme is strongly dependent on the network characteristics (latency, bandwidth and congestion) and somewhat less dependent on the characteristics of the processors implementing the computations. Execution time is estimated for two execution environments, one modeling a typical peer to peer network and another modeling the case where the algorithm is implemented by web servers as a backbone service of the Internet as briefly sketched in Section 8.

4.6.1 Peer to Peer Implementation

To simplify the analysis the following assumptions are made:

- A homogeneous network in which all peers are always present.
- A network transfer model where the peers collect together all the pagerank messages for each other generated during one pass into a single message, and transmit this message in one network call per peer.
- Each peer serializes sending of these messages to other peers instead of sending them concurrently to all peers.
- The IP address caching scheme proposed in Section 3.2 is used, hence the pagerank update messages can be directly exchanged between peers without having to be routed every time.
- The computational work to be done is constant over all passes.

The execution times estimated in this model should be conservative, given the assumptions of serialized network transfer between peers and constant computational work per pass.

Let c_i be the computation time for a pass at peer i, L_{ij} be the number of document links from peer i to peer j, the size of a pagerank message be m and the average transfer rate in the network be t. The execution time per pass at peer i can then be estimated as:

$$\text{Execution Time per Pass} = c_i + \sum_j \frac{Lij * m}{t} \ \forall \ \ \{\text{Peers j}\} \tag{4}$$

Based on simulations on Pentium III and Pentium 4 class machines we estimate the computation required per pass for the 5000k node graph to be of the order of a minute or less. The transfer rate between peers in peer to peer networks may vary over a considerable range. We used a conservative transfer rate of 32 Kbytes/sec to estimate an upper bound on execution time and also a more aggressive transfer rate of 200 Kbytes/sec. A message size of 24 bytes per message is used (128 bits for GUID, 64 bits for pagerank value). The execution time for the convergence of the 5000k node graph is shown in the last two columns in Table 3. In all cases the execution times are reasonably measured in hours, totally dominated by communication time and decrease rapidly with faster transfer rates.

4.6.2 Web Server Implementation on Internet Scale

Assume the distributed algorithm is implemented as a function of a web server. The average transfer rates of the networks connecting web servers can reasonably be expected to be at the rate of at least a T3 line (about 5.6 Megabytes per second). We estimate the execution time for a network with 3 billion documents to be about 35 days for a convergence threshold of 10^{-3} and about 14 days for a convergence threshold of 10^{-1}. These execution times are of the same order of magnitude as the current web crawler based centralized method. The distributed pagerank scheme incrementally incorporates document insertions and deletions into the pageranks and does not require frequent recomputations. (Quantitative results for document insertions are discussed in the next section.) Furthermore, 99% of the graph converges in as few as 10 passes which would correspond to approximately 4 days.

4.7 Document insertions and deletions

To evaluate the effect of document insertions and deletions we measure the total number of network messages that can be generated when a document is inserted. When a new document is inserted, it can only have outlinks. Since this

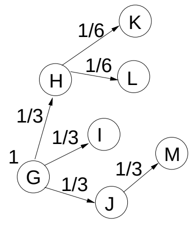

Figure 2. Propagation of pagerank increments on document inserts.

is a new document, there cannot be inlinks already pointing to it. Adding a node is equivalent to adding an extra column and row to the A matrix and one extra entry to the R matrix of Equation 3. The row added to the A matrix is all zeroes (since a new node cannot have inlinks coming into it). The column added will have values of $1/n_o$ when a link is present from this node to another node and zero elsewhere (n_o is the number of outlinks out of this node).

We measure the network traffic in terms of number of pagerank update messages, by performing the following experiment. For each of the four graph sizes, we pick a random node and set its pagerank to the initial pagerank value (1.0 in our case). We then propagate this pagerank to all its outlinks. Each outlink will get only a $1/n_o$ contribution. When these messages reach the outlinks, they will in turn send out messages to their outlinks. As shown in Figure 2, when *G* sends an update message to *H*, *H* will in turn send an update message to *K* and *L* - incrementing *K*'s and *L*'s pagerank by some amount. In our example *G*'s increment to *H* will be $1/3$. *H*'s increment to *K* and *L* will in turn be $1/6$ and so on. At some point the increment will be smaller than the error threshold, at this point no more pagerank update messages are generated. We measure the path length and the total number of nodes to which an update message is sent (called the *node coverage*). This *node coverage* is an upper bound on the number of messages a document insert can generate. Adding multiple documents simultaneously may generate fewer messages than separately adding them, because the inserted new documents could have links to the same documents. This effect will be less pronounced as graph sizes grow.

In Table 4, the path length and node coverage are shown for the four graph sizes and error thresholds of 0.2 and 10^{-1} through 10^{-5}. These numbers were obtained by averaging the results over 1000 randomly picked nodes from the graphs. Both the path length and the node coverage are largely independent of, or grow extremely slowly with the graph size, indicating the scalability of the algorithm. The large anomaly in node coverage for the 10^{-5} threshold is because at such a low threshold almost the entire graph is reachable for the 10k graph and the node coverage is limited

Threshold	Path length			
	10k	100k	500k	5000k
0.2	2.0	2.2	3.2	2.7
10^{-1}	2.9	3.1	4.8	3.4
10^{-2}	5.8	6.3	9.1	7.4
10^{-3}	8.7	9.3	14.5	11.1
10^{-4}	11.6	12.6	19.3	15.2
10^{-5}	14.7	16.0	24.3	19.5
	Node coverage			
0.2	14	19	17	34
10^{-1}	30	39	40	61
10^{-2}	293	388	330	602
10^{-3}	2139	3338	3625	6073
10^{-4}	7067	20544	24234	52888
10^{-5}	9863	62115	91736	326702

Table 4. Path lengths and node coverage.

by graph size. Examining the values down a column, we can see that node coverage grows reasonably rapidly with the threshold, and is almost linearly dependent as expected.

Document deletions: Document deletions are very similar to document inserts. A pagerank message is sent with a negative increment. Mathematically removing a document is equivalent to deleting its row and its corresponding column from the A matrix.

4.8 Pagerank Summary

In the previous subsections, we evaluated the proposed distributed pagerank scheme over a wide range of graph sizes and convergence error thresholds. The results on convergence, quality of pageranks, message traffic and the execution time demonstrate the scalability and performance. Dynamic effects affected the convergence rate only to a small extent. Based on these results we conclude that an error threshold of 10^{-3} seems ideal — pageranks have a maximum error of less than 1%, with reasonably low message traffic generated for all the graphs sizes. Table 5 summarizes these conclusions.

4.9 Incremental Search

This section presents our preliminary results on measuring the effectiveness of the incremental search mechanism in reducing network traffic while executing keyword search queries. There are two key factors that contribute to reducing traffic. Firstly, the presence of the pagerank greatly reduces traffic on multi-word queries. Secondly, the presence of a ranking scheme ensures that the user sees the most

Convergence	Fast convergence, high tolerance and adaptability to peer leaves and joins, good scalability with graph size.
Pagerank Quality	Very high, typically $\ll$ 1% error, good scalability with graph size.
Message Traffic	Reasonably low, message traffic per node nearly constant, logarithmic growth with accuracy.
Execution Time	Reasonably low, dominated by network transfer time.
Document Insertion, Deletion	Handled naturally, no global recomputes required, pageranks continuously updated.

Table 5. Distributed pagerank computation summary.

important documents first, while other documents can be fetched incrementally if requested.

We first built our own document corpus by performing a crawl of a set of news web pages. We then computed the pagerank of these pages using our distributed pagerank scheme. Automatically synthesized search queries were then simulated to measure the reduction in traffic.

Documents and search queries: We built a document corpus consisting of around 11,000 documents amounting to 99MB of storage. After removing common stopwords and thresholding based on most frequently appearing terms, the document corpus was reduced to 1880 dimensional data. Two-word and three-word search queries were generated by randomly combining the top 100 most frequent terms. We randomly distributed these documents on 50 peers and computed the pagerank using our distributed scheme. The search queries were simulated on this 50 node peer to peer system.

Search results: We simulated these automatically synthesized search queries to measure the performance of incremental search. We assumed that each search term in the query, was always present in a different peer. Therefore in the baseline case, for every two word query, a set of document IDs have to be transferred between two peers (the one's owning the first term and second term). Finally the document IDs have to be transferred to the user. Twenty each of the two and three word queries were used in our experiments.

We simulated two instances of the incremental search algorithm — one where the top 10% (based on pagerank) of the hits are transferred to the next peer, and one where the top 20% of the hits are transferred to the next peer. The results are shown in Table 6. The reduction in traffic is measured in terms of number of document IDs that must

	2 Term queries	3 Term queries
Average Traffic Reduction		
Top 10% forwarded	12.2	11.9
Top 20% forwarded	6.5	6.9
Average # hits returned		
Top 10% forwarded	55.3	41.7
Top 20% forwarded	66.8	27.7
Baseline	1603.9	835.6

Table 6. Network traffic reduction when using Incremental search with pagerank.

be transferred between peers, and finally back to the user. The baseline we compare against is a system where there are no pageranks and hence all the document IDs need to be transferred between peers. When the *top 10%* of the hits are forwarded, more than a factor of 10 reduction in traffic is obtained for both two- and three-word queries. When the *top 20%* of hits are forwarded, more than a factor of 6 reduction is obtained. In both cases the number of results returned is a very manageable amount unlike in the baseline case. The reason why there are fewer 3 term hits with top 20% forwarding than with top 10%, is because of a simulation artifact. When the top x% of the documents falls below a threshold (we used 20), then all the results are forwarded along. In some cases the top 20% amounts to a number just over 20. But in the top 10% cases it does not, but the entire set of hits is far greater than 20 and they are all forwarded.

5 Centralized Crawler Implementation

We briefly address the issue of using a centralized crawler on DHT based P2P storage systems. Firstly, a centralized crawler is against the philosophy of P2P systems. Secondly, a rudimentary centralized crawler would generate an extremely large amount of traffic - basically it amounts to fetching all the files on the P2P system to a central storage server; such a scheme is undesirable. A more efficient crawler based system would only transmit the link structure to the central server which would compute the pageranks and redistribute the ranks to the peers owning the documents. Document inserts/deletes can be handled by having the peers directly communicate with the pagerank server — this avoids the need for a recrawl every so often (as is done on the Internet).

A centralized crawler would be unworkable on a Freenet based system because of the anonymity guarantees.

6 Future Work

In our current work, we have focussed on the design, computation, scalability and feasibility aspects of the pagerank computation for P2P systems. We simulated a simple P2P system model to perform our experiments. In future work, we will implement the distributed computation of the pagerank on a P2P system.

Many interesting extensions and optimizations are possible. One is to address whether the link structure in documents can be used for mapping documents to peers, and whether this will alleviate network overheads in the computation of the pagerank. Arasu et. al. [2] noted that utilizing the structure of the web can materially speedup convergence of the iteration. This approach is closely related to the Fast Multipole method for solving linear systems [9, 10]. Secondly, we propose to investigate adaptation of the fast multipole solver to pagerank computations. Thirdly, we will investigate the effectiveness of distributed asynchronous linear solutions executing on P2P systems in other problem domains, where the generation of the elements of the matrices can be, or are distributed across a network. And finally, the Internet wide scalability of distributed pagerank, with web servers computing pageranks as a service will be explored.

7 Related work

We are not aware of any other distributed implementations for computation of pagerank or other document relevance metrics. There has been a great deal of research on speeding up the centralized computation of the eigenvectors of the Markov matrices which underlie pagerank computation [2, 12, 13, 14]. It appears on the basis of our limited results that the asynchronous iteration may converge more rapidly than the acceleration methods studied in [14]. Verbeke et al. [23] proposed a framework for peer-to-peer computation for coarse-grain parallelization using the JXTA protocols. More closely related work is the conventional parallelization of pagerank computation on multiprocessors. The only results we have found were by Chen and Zhang [6]. These authors compared synchronous and asynchronous iteration on 128 processors and found that asynchronous iteration was more efficient.

8 Conclusions and a Question

This paper proposes and evaluates a distributed algorithm which enables computations of pageranks in peer to peer networks and applies it to the problem of multiple word keyword searches in peer to peer networks. The distributed algorithm converges rapidly, produces high quality pageranks, and enables incremental and continuous computation of pageranks as documents are added/deleted.This formulation can be coupled with standard information retrieval methods to enable effective keyword search in P2P systems. An incremental search mechanism which provides a ten-fold network traffic reduction on multi-keyword search queries in P2P networks is defined and evaluated.

While the primary target for the distributed pagerank computation was peer to peer networks, the algorithm can be extended in a straightforward manner to the World Wide Web and its vast corpus of web pages. By augmenting web servers and the HTTP protocol to exchange messages, web servers can be collectively responsible for computing the pageranks for documents they host. Web servers would play the role of peers in a peer to peer network and exchange pagerank update messages. Application of the algorithm at the WWW level, if practical, removes both the need for a central server computing pageranks, and the need for periodic recrawls to reflect changes. Assuming that the distributed algorithm can be applied on the WWW scale, which seems at first glance to be plausible, a very interesting possibility arises — can the entire Internet keyword index be computed and stored in a distributed manner? Efficient keyword search requires both components, pagerank and keyword index. Coupling such a distributed keyword index with the web server computed distributed pagerank, could enable a fully distributed pagerank based keyword search for the Internet.

Acknowledgments

We thank the anonymous reviewers for their suggestions that helped improve the quality of this paper. We would like to thank Mike Dahlin for his insightful comments and suggestions. This research was supported in part by the National Science Foundation under grant 0103725 “Performance-Driven Adaptive Software Design and Control”.

References

[1] PageRank Explained. http://www.webrankinfo.com/ english/pagerank/.

[2] A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. Pagerank computation and the structure of the web: Experiments and algorithms. Nov 2001.

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

[4] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, , and J. Wiener. Graph structure in the web: experiments and models. In *Proceedings of the 9th World Wide Web Conference*, 2000.

[5] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra Applications*, 2:199–222, 1969.

[6] Y. Cheng and H. Zhang. Parallelization of the page ranking in the google search engine. http://manip.crhc.uiuc.edu/ chen/pagerank.ps.

[7] I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, and B. Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[8] O. D. Gnawali. *A Keyword-Set Search System for Peer-to-Peer Networks*. M.E. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 2002.

[9] L. Greengard and V. Rokhlin. A Fast Algorithm for Particle Simulations, 1987.

[10] L. Greengard and V. Rokhlin. A new version of the fast multipole for the LaPlace Equation in Three Dimensions, 1997.

[11] T. Haveliwala. Efficient computation of pagerank. Technical Report 1999-31, Stanford Digital Library Technologies Project, 1999.

[12] T. Haveliwala. Topic senstive pagerank. In *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002*, 2002.

[13] G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the Twelfth International World Wide Web Conferemce, WWW 2003, To Appear*, 2003.

[14] S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating pagerank computations. In *Proceedings of the Twelfth International World Wide Web Conferemce, WWW 2003, To Appear*, 2003.

[15] A. Z. Kronfol. FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine. Technical report, Princeton, 2002.

[16] C. Moler. The world’s largest matrix computation. http://www.mathworks.com/company/newsletter/ clevescorner/oct02_cleve.shtml.

[17] 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.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

[19] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. Technical report, Rice University, 2002.

[20] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[21] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Initial specification of a distributed pagerank scheme for p2p systems. http://www.cs.utexas.edu/ karu/docs/ projectreport.pdf.

[22] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[23] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. In *Proceedings of the 3rd International Workshop on Grid Computing*, November 2002.