Journal of Grid Computing **1:** 291–307, 2003.

Pagerank Computation and Keyword Search on Distributed Systems and P2P Networks

Karthikeyan Sankaralingam, Madhulika Yalamanchi, Simha Sethumadhavan and James C. Browne
Department of Computer Sciences, The University of Texas at Austin, 1 University Station C0500, Austin, TX 78712-0233, USA
E-mail: karu@cs.utexas.edu

Key words: asynchronous iteration, Internet search, keyword search, pagerank, peer-to-peer computing, web server

Abstract

This paper presents a fully distributed computation for Google's pagerank algorithm. The computation is based on solution of the matrix equation defining pageranks by a distributed implementation of asynchronous iteration. Pageranks for the documents stored on a web server or on a host in a peer-to-peer network are computed in place and stored with the documents. The matrix is never assembled and no crawls of the web are required. Continuously accurate pageranks are enabled by incremental computation of pageranks for documents as they are inserted onto a network storage host and incremental recomputation of pageranks when documents are deleted. Intrahost and intradomain dominance of document link structure is naturally exploited by the distributed asynchronous iteration algorithm.

Three implementations: (i) a simulation which was previously reported, (ii) an implementation of the algorithm in a peer-to-peer computational system and (iii) an embedding of the computation in web servers, are described. Application of the three implementations to three different workloads, two constructed following power law network models for link distributions and one derived from the Government document database are reported. Convergence for computation of a complete set of pageranks is rapid: 1% accuracy in 10 or fewer messages per document. Incremental computation of pageranks resulting from addition or deletion of documents also converges rapidly, usually requiring 10 or fewer messages per document.

Coupling locally stored pageranks with the documents in a peer-to-peer network dramatically diminishes the volume of data which must be transmitted to satisfy keyword searches in peer-to-peer networks.

The web server implementation shows that the distributed algorithm can be used to enable web servers to compute pageranks for the documents they store and thus potentially enable effective keyword searches for the documents stored on the web servers of intranets by utilizing unused processing power of the web servers.

1. Introduction

This paper presents a fully distributed computation for Google's pagerank algorithm [30, 18] based on solution of the matrix equation defining pageranks by asynchronous iteration [9]. This paper evaluates the effectiveness of the distributed asynchronous computation and shows how it enables effective keyword search in 'peer-to-peer' systems. The computations are executed by the storage hosts for the documents (web servers or peers in a 'peer-to-peer' network) in place and stored with the documents. The matrix is never assembled and no crawls of the web are required. Continuously accurate pageranks are enabled by incremental computation of pageranks for documents as they are inserted onto a network storage host and incremental recomputation of pageranks when documents are deleted. Intrahost and intradomain dominance of document link structure is naturally exploited by the distributed asynchronous iteration algorithm.

Three implementations, a simulation of a peer-to-peer network, an actual peer-to-peer system and an embedding of the computation in network of web

servers, are described and results from these implementations are given. Convergence for computation of a complete set of pageranks is rapid: 1% accuracy requires about 10 or fewer messages per document. Incremental computation of pageranks resulting from addition or deletion of documents (not surprisingly) also converges rapidly, usually requiring 10 or fewer messages per document.

Coupling locally stored pageranks for documents with incremental return based on pageranks dramatically diminishes the volume of data which must be transmitted to satisfy keyword searches in peer-to-peer networks.

The web server implementation shows that the distributed algorithm can be used to enable networks of web servers to compute pageranks for the documents they store and thus potentially enable effective keyword searches for the documents stored on the web servers of intranets by utilizing unused processing power of the web servers.

1.1. *Pageranks and Centralized, Crawler-Based Computation of Pageranks*

Google's pagerank computation and other relevance rankings for web pages have made Internet keyword searches orders of magnitude more effective. 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. 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 eigenvector of the largest eigenvalue of the sparse matrix representing the link graph, using an iterative numerical method on a computational server. There are however several drawbacks with the web crawl/central server process for computation of page ranks. Explicitly generating and then solving of this very large (order three billion or so [29]) matrix takes days [31] and requires a very large configuration of computational and storage servers. These issues have led to several recent studies on methods for accelerating computation of pageranks [22, 2, 21, 19] through improvements in the computational algorithms. The approach defined and described in this paper does not require a web crawl, uses the unused processing power of the hosts storing the documents to compute the pageranks and provides continuously accurate pageranks.

1.2. *Motivation for a Distributed Implementation of Pageranks*

The original motivation for development of the distributed algorithm was to enable effective keyword search in peer-to-peer (P2P) networks. P2P networks [33, 35, 37] are emerging as potentially important structures for information and content management and distribution. A network of web servers is also an instance of a P2P system where the relationships among peers are defined by the URL links in the documents stored on the web servers. Effective keyword search is a crucial method for information and content management. Recent papers on P2P implementations of keyword search for documents [15, 28, 34] have called attention to the need for a document ranking system compatible with P2P systems to enable practical keyword search in such 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 will flood the network. It seems probable that keyword search will not be practical in P2P-like systems in the absence of an effective P2P implementable document ranking algorithm. A crawler-based, centralized server approach for computation of such ranks is incompatible with a P2P implementation of keyword search while the distributed asynchronous algorithm is natural for P2P-like systems.

Embedding the distributed pagerank computation in P2P systems, including networks of web servers, was straightforward and the computations consume very modest processor and communication resources. This suggests that the distributed algorithm may enable effective Internet keyword search without web crawls or specialized server complexes.

1.3. *Pageranks + Incremental Return = Keyword Search in Peer-to-Peer Networks*

Since the distributed computation leaves the pageranks for the documents on the storage host, incremental return of documents based on pageranks is straightforward. Coupling of pageranks and incremental document return is found (see Section 3.4 for details) to produce dramatic benefit both in terms of effectiveness for users, and decrease in network traffic engendered

by document transfer. In-place computation of pageranks has the potential to make keyword search on P2P-like systems as efficient as the current centralized Internet keyword search.

1.4. *Summary of Contents*

The rest of the paper is organized as follows: Section 2 defines and describes the static version of the distributed asynchronous algorithm. Section 3 describes the implementation of the algorithm on peer-to-peer systems and on web servers. Section 4 analyzes the basic properties of the chaotic iterative algorithm for the pagerank computation. Section 5 gives some experimental evaluation of the distributed pagerank algorithm based on the implementation in a simulated P2P system. Section 6 provides a detailed evaluation of the peer-to-peer implementation and web server implementation of the distributed pagerank algorithm. Section 7 discusses related work, and Section 8 concludes.

2. The Algorithm

This section defines and describes: document link structure, formulation of queries and the static version of the distributed formulation of the pagerank algorithm where pageranks are computed in-place on the processors of a 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 [33], Pastry [35] or Chord [37] the GUID (Global Unique Identifier) implements a pointer to each document. All these systems are distributed hash table (DHT) based systems. Systems like Freenet [11] while providing no bounded search guarantees, also maintain a unique identifier, the SSK (subspace key). Throughout this paper we focus on DHT based systems, and web pages on web server, 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 denoting its importance to every document in a system. 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:

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

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 [18, 30].

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

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

where the matrix R is a column vector of length N representing the ranks of pages, D is a constant column vector$(= (1-d)/d)$, and A is an $N \cdot N$ link matrix defined with entries a_{ij} such that:

$$\begin{aligned} a_{ij} &= \frac{1}{C(j)} && \text{if there is a link from } j \text{ to } i, \\ &= 0 && \text{otherwise.} \end{aligned}$$

It can be shown that the solution for R corresponds to the eigenvectors of $(A + D \times E)$, where E is a constant $N \cdot N$ row-stochastic matrix. The formulation shown in Equation (3) is an iterative solution to Equation (2).

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

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 $(A + D \times E)$. This algorithm has been formally proven to converge for appropriate values of d [30].

```
/* 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)/oldrank;
   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)/oldrank;
   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.

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 of the documents with updated pagerank.
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 pseudocode in Figure 1.

This algorithm is essentially a distributed 'peer-to-peer' implementation of a chaotic (asynchronous) iterative 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 (see Section 4.2).

3. Implementation

In this section we describe the implementation of the proposed distributed pagerank computation algorithm on P2P systems and also on an Internet-scale web server based environment. While P2P systems have well defined underlying communication protocols for exchanging messages, web servers can communicate with each other using standard HTTP messages. Following the description of the basic pagerank computation, we extend it to implement the dynamic behaviors exhibited in P2P systems. We then describe methods for keyword search with incremental return using document ordering based on the computed pageranks.

3.1. *Pagerank Computation*

3.1.1. *Peer-to-Peer Environments*

P2P systems typically reference each document using a GUID (globally unique identifier), constructed by hashing the document contents. Each peer is assigned a portion of the GUID space and owns any document that falls into that space. Distributed hash table (DHT) based systems provide a well structured

[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.

way of assigning these GUIDs to peers and hence provide guaranteed bounded access times. Other less structured systems like Freenet provide no such guarantees. All P2P systems provide a messaging layer which uses the mapping of these GUIDs to peers to achieve peer-to-peer communication, so that any peer can effectively send a message to any other. Figure 2 shows a high level description of such a P2P system.

To implement the pagerank computation, we maintain the pagerank for each document along with its GUID. As outlined in the algorithm in Section 2, when a peer first comes 'alive,' it sends pagerank update messages corresponding to the outlinks for the documents it stores. Each peer then listens for pagerank update messages on the network, and performs pagerank updates on receipt of such messages. The pagerank update messages are all delivered through the underlying P2P messaging layer. A pagerank update message contains the GUID of the destination document and the incremental value for the pagerank of the receiving document. The underlying P2P routing layer, will deliver this message to the appropriate peer. Since the pagerank computation is an iterative process, a single document may send messages to its outlinks multiple times before eventual convergence. To reduce pagerank routing traffic, the first time a pagerank update message is delivered from source to destination, a reply message with the destination peer's IP address may be sent to the source. The source node then maintains a table mapping GUIDs to IP addresses and delivers subsequent update messages point-to-point, thus reducing routing overheads.

Some P2P systems implement document caching, by replicating documents and storing copies on different peers, to reduce retrieval time. The caching process is augmented to mark one copy as the master copy, and only this copy is allowed to send and receive pagerank update messages. Pointers need to be maintained at the master copy of document to point to cached copies, so that all copies of the document can be updated to the correct computed pagerank. The storage requirement for this scheme scales linearly with the sum of the outlinks in all documents in a peer. When the pagerank converges for a document, the value is sent to all the cached copies. In the incremental algorithm where approximate pageranks are continuously maintained, the pagerank value is sent to the cached copies periodically as the pagerank computation proceeds.

3.1.2. *Web Server Environments*

We now describe the pagerank computation scheme in a web server environment. We foresee such systems being utilized in two possible scenarios: (1) to build an Internet-wide search engine without a centralized crawler and computation farm, and (2) to build search engines for intranets without using a separate computation server, instead pageranks are computed for documents hosted on the intranet using the nodes hosting the documents.

In both cases, the basic implementation is the same. A simple pagerank computation engine executes on the web server as a service. When documents are added to a web server, the pagerank for the document is initialized to a default value, pagerank update messages are sent to the web servers corresponding to its outlinks using the standard HTTP protocol. When the web servers receive an update message (implemented through a CGI-BIN interface), the pageranks are updated by the computation engine, and any further update messages generated are delivered to other web servers. The URLs embedded in the documents

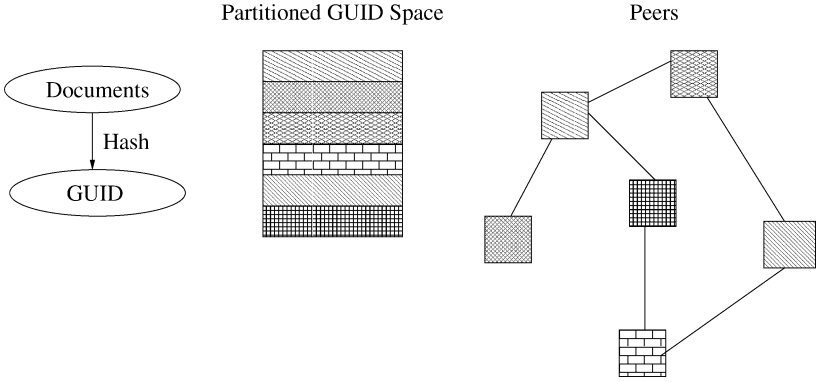


Figure 2. Typical P2P system organization.

directly provide the name of the web server to contact for each pagerank message.

Compared to current centralized implementation of pagerank computation this web server computed pagerank approach has several advantages:

- Pageranks for the documents it hosts are known at each web server,
- Pageranks are computed without any massive, periodic crawl of the network,
- Pageranks are computed in place without separate computation servers,
- Pageranks are incrementally and automatically computed when new documents are inserted and deleted.

3.2. *Extension to Dynamic Systems*

We now extend the distributed pagerank computation to handle dynamic behaviors which fall into two basic categories. First, in any document database, documents are regularly added and deleted. Second, in a P2P system, and to a lesser extent with web servers, peers (or servers) are constantly entering and leaving the system. 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 may 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. Below, we describe protocols to incorporate these dynamic behaviors into the distributed algorithm, and describe them in the context of P2P networks (they can be directly applied to web servers as well). We show that after an initial convergence the pageranks can be incrementally updated as peers and documents get added and deleted.

Document inserts and deletes: When a new document is inserted into the network, its pagerank is initialized to some default 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 re-converges.

Peer (or server) leaves and joins: When peers (servers) 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 the outlinks in all documents in a peer.

3.3. *Integration with P2P Search Algorithms*

The 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 DHT based systems like CAN, Pastry, and Chord and on similarity based routing table systems such as Freenet, which do not provide bounded search guarantees. We propose an incremental search algorithm that can be used for efficient keyword search using the pagerank to reduce network traffic.

3.3.1. *DHT Systems*

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. Boolean multiword queries are inefficient on DHT based systems, because documents ids of all the hits for the keywords must be passed from peer-to-peer for each word in the query. 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 stored in the index. When a search is performed the pagerank is used as the relevance metric.

3.3.2. *Freenet*

Freenet is representative of systems which do not provide bounded search guarantees. FASD is a key word search mechanism for Freenet, where a metadata key representing the document as a vector is associated with every document [28]. 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.

3.3.3. *Incremental Searching*

We propose incremental searching to address the issue of network traffic in P2P systems caused by multiword queries. The search scheme which incrementally

returns documents sorted by pagerank 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. The peer with the index entries for the next keyword receives only a fraction of documents but the most relevant documents. The second peer finds the documents that match the second term and performs the user specified 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 [5] based method [34] to provide further reduction in traffic.

3.4. *Integration with Web Server Search Algorithms*

The straightforward way to build keyword search for a network of web servers is to implement a centralized index, and transfer the pageranks from the web servers to this centralized index, when they converge. This index server essentially is the search engine for the network of web servers and services search queries. Such a system computes the pageranks in-place at the web servers, but uses a stand-alone index server. Application of the web server implemented distributed computation of pageranks but utilizing a index server appears to be an efficient solution for keyword search in small to moderate size corporate intranets. To provide a completely distributed keyword system, the keyword index must be distributed to the web servers using a mechanism such as a DHT with search queries being routed according to the DHT until the correct web server is found. While a fully distributed system is feasible for small to moderate networks, the latency to answer search queries, when implemented at a Internet-wide scale, would make such a system unusable.

4. Analysis of Asynchronous Iteration-Based Computation of Pageranks

4.1. *Why Asynchronous Iteration?*

Conventional approaches to parallelization of pagerank computations are not appropriate for P2P or other heterogeneous systems of independent hosts such as collections of web servers. Heterogeneity of processing power and connectivity will cause synchronous parallel iterative algorithms for matrix solution to flounder on synchronization delay. Association of a powerful central server complex to compute pageranks is contrary to the motivation for P2P networks. The natural approach to consider for distributed computation of pageranks is asynchronous iterative solution [9] where each document or collection of documents computes its own pagerank following the algorithm given in Section 2. The issues are: (1) the effectiveness and convergence of asynchronous iterative for computation of the eigenvectors of matrices defining pageranks, particularly for dynamic sets of documents and (2) the performance of asynchronous iteration in a P2P-like execution environment.

4.2. *Convergence*

Frommer and Szyld [14] give an excellent survey of asynchronous iteration and the mathematical theory of the several models of asynchronous iteration including necessary and sufficient conditions for convergence. Strikwerda [38] gives weaker conditions for convergence based on a probabilistic analysis. There have been several experimental studies of the performance of asynchronous iterative solvers for linear system solution (and non-linear system solution) on clusters and multiprocessors. See, for example, [4].

Given that each document has at least one outlink, the link matrices defining pageranks are $N \times N$ row-stochastic matrices, and the matrices to be solved for pageranks have largest eigenvalue 1. The system to be solved for the eigenvectors defining pageranks meets the necessary and sufficient conditions for convergence. It has been established by Haveliwala and Kamvar [20] and Elden [13] that the second eigenvalue of the pagerank matrix is equal to or less than the damping factor, d, in Equation (1). d is usually chosen to be about 0.85, suggesting that an iterative solution will converge fairly rapidly.

Cheng and Zhang [10] had previously found that a parallel asynchronous iterative computation of pageranks converged more rapidly that a synchronous computation when both were executed on a cluster multicomputer.

4.3. *Matrix Structure*

It has been determined [3] that the distribution of links is highly localized. The outlinks of the documents on a web server are predominantly, almost 80%, to other documents on the same web server. This implies a

near block structure for the link matrix. This is very advantageous for solution by asynchronous iteration since most of the update messages are internal to a web server or (assuming the same locality property) to a host in a peer-to-peer network. The computation internal to a web server or host can iterate to local convergence while periodically interacting with messages originating in outlinks from other web servers.

4.4. *Validation and Performance Metric*

We implemented the pagerank computation as described in Section 3.1.1 on a locally developed P2P system called CoorSet [26, 8] which utilizes broadcast communication. We systematically evaluate the performance of asynchronous iteration for computation of pageranks including incremental computation of pageranks for dynamically changing document sets. We also validate the pageranks obtained from the simulated peer-to-peer implementation and from the web server implementation described in Sections 3.1.1 and 3.1.2. Where possible, the pageranks (eigenvectors of the largest eigenvalue) computed by asynchronous iteration were validated by comparision between these two implementations and also to pageranks for the same matrices computed using MatLab. The comparisons were accurate to the number of significant figures determined for the iteration. The pageranks computed through the P2P and web server implementations agreed to the number of significant figures sought in the iteration.

We used average number of update messages per document as a common metric for evaluating each implementation. The results show that the asynchronous iterations converge rapidly and provide a practical method for distributed computation of pagerank on P2P-like systems systems. The detailed experimental methodology and a summary of results are given in Section 6.

5. Algorithm Evaluation

This section presents an evaluation of the distributed pagerank algorithm executing on a model P2P system simulated on a multiprocessor cluster. This execution environment facilitated measurement of the properties and behavior of the algorithm. A subset of these results have been previously reported in [36]. There are three key components to be modeled – the graph structure of the document links, the basic P2P system, and the search queries. Section 5.1 defines the document link structure and Section 5.2 details the simulation infrastructure. The results: quality of the pagerank, amount of pagerank message traffic generated, execution time, the quantitative effects of document insertion and deletion, and the traffic reduction obtained from using the pagerank in search queries are presented in Sections 5.3–5.7.

5.1. *Graph Structure*

Researchers have studied the graph structure of Internet documents by performing a crawl of a significant fraction of the Internet [7]. 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.

5.2. *Distributed Computation: Simulation Methodology*

The 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. The simulator executes multiple passes, on each pass, evaluating the peers in some order. For each peer, we process its update messages and send any update messages resulting from the recomputation. We assume that pagerank messages are sent and received instantaneously and all peers start concurrently. If peer A sends update messages to peer B, and if B is evaluated *after* A in the execution order, B will see the update messages in that same pass. Sending of pagerank update messages continues until the computation converges. The computation converges when the error (absolute relative difference between successive values of the pagerank of a document), for 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 Sections 5.3 through 5.6 are based on simulating 500 peers.

5.3. *Quality of Pagerank*

We now examine the quality of the pageranks generated by the distributed pagerank computation. The quality metric for pageranks is the relative error in

pagerank. Higher quality pageranks have lower relative error and are produced by using a lower error threshold for convergence. The primary disadvantage of having a low threshold is an increase in the number of pagerank update messages. 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 metrics: the maximum error in pagerank when using different thresholds, average error across all the documents, and distribution of relative error across the document set.

In Table 1 the distribution of 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 pagerank 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 is shown in Table 2. The quality of the pageranks achievable in a huge set of the documents, even with high error thresholds is surprising.

5.4. *Message Traffic*

Message traffic is an important metric for cost of execution of a distributed algorithm. 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. The document structure used in the simulated executions did not incorporate the host and domain locality properties observed for actual document link structures while these locality properties were used and exploited in the direct implementations. From Table 3, 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 distributed algorithm to large problem sizes.

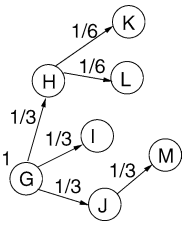


Figure 3. Propagation of pagerank increments on document inserts.

5.5. *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 have outlinks but will have no inlinks. 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,

Table 1. 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	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 100 k 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 500 k 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 5000 k 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. Error distribution summary.

Threshold 0.2	Threshold 0.001
– 10 k graphs: only 10 nodes have error $\geqslant$ 3%, max 10%; – 100 k graph: only 100 nodes have error $\geqslant$ 27%, max 66%; – 500 k graph: only 500 nodes have error $\geqslant$ 1%, max 10%; – 5000 k graph: only 50,000 nodes have error $\geqslant$ 35%, only 5000 nodes have error $\geqslant$ 109%, max 135%.	– 10 k graphs: only 10 nodes have error $\geqslant$ 0.02%, max 0.06%; – 100 k graph: only 100 nodes have error $\geqslant$ 0.2%, max 0.7%; – 500 k graph: only 500 nodes have error $\geqslant$ 0.02%, max 0.08%; – 5000 k graph: only 5000 nodes have error $\geqslant$ 0.004%, max 0.07.

Table 3. Variation of message traffic with error threshold ϵ. Total number of messages shown in millions. Avg. is average number of messages per document.

Threshold	Number of messages							
	10 k nodes		100 k nodes		500 k nodes		5000 k nodes	
	Total	Avg.	Total	Avg.	Total	Avg.	Total	Avg.
0.2	0.35	35	3.80	38	29.97	60	169.1	33.8
10^{-1}	0.39	39	4.16	42	31.14	62	183.8	36.7
10^{-2}	0.51	51	5.36	54	35.48	71	395.0	79.0
10^{-3}	0.63	63	6.53	65	39.87	80	440.3	88.1
10^{-4}	0.75	75	7.74	77	44.36	89	485.2	97.1
10^{-5}	0.90	90	9.06	91	48.84	98	533.2	106.6
10^{-6}	1.11	111	10.54	105	52.84	106	586	117.21

Table 4. Path lengths and node coverage.

Threshold	Path length			
	10 k	100 k	500 k	5000 k
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			
	10 k	100 k	500 k	5000 k
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

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 3, 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 10 k graph and the node coverage is limited 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.

5.6. *Pagerank Summary*

In the previous subsections, we evaluated the proposed distributed pagerank scheme over a wide rangeof graph sizes and convergence error thresholds. The results on convergence, quality of pageranks, message

Table 5. Distributed pagerank computation summary.

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

	Average traffic reduction	
	2 term queries	3 term queries
Top 10% forwarded	12.2	11.9
Top 20% forwarded	6.5	6.9
	Average # hits returned	
	2 term queries	3 term queries
Top 10% forwarded	55.3	41.7
Top 20% forwarded	66.8	27.7
Baseline	1603.9	835.6

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.

5.7. *Incremental Search*

This section presents our 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 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.

5.7.1. *Documents and Search Queries*

We built a document corpus consisting of around 11,000 documents amounting to 99 MB of storage. After removing common stopwords and thresholding based on most frequently appearing terms, the document corpus was reduced to 1880 keywords. 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.

5.7.2. *Search Results*

We simulated these automatically synthesized search queries to measure the performance of incremental search. We made the worst case assumption 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 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.

6. System Evaluation: P2P and Web Server Implementation

We implemented the pagerank computation on two systems – a broadcast based peer-to-peer system called CoorSet [26, 8] and on a set of web servers running thttpd [32], exchanging messages and executing a pagerank computation engine. In this section we briefly describe the implementation of these systems, and then present our experimental results. We first describe the implementations, experimental methodology and the datasets used. We then describe the results of the experiments for the P2P system and the web servers.

6.1. *Implementations*

6.1.1. *P2P System*

CoorSet is a development environment based on a broadcast-based coordination system. The components (peers) in this system communicate through associative broadcast (dynamically addressed multicast). Each message carries with it the target set specification and is broadcast into the network. The target set is determined for each message by the recipients whose local state satisfies the target specification. Thus, associative broadcast enables targeting of messages to processes in specific states and enables each process to select the properties of messages it requires. For the pagerank computation we use CoorSet to build a P2P system. Each peer consists of the following: (a) a unique name which is a part of its profile, and (b) a set of documents, with each document having a unique id, prefixed by the peer name and links to the documents to which it refers, and (c) the implementation of the distributed pagerank algorithm.

The pagerank computation proceeds as follows. At every peer, for each document, pagerank update messages are broadcast into the network with a target set specification which is the name of the intended recipient. The peers execute the local iteration continuously and send update messages when document pageranks change more than the threshold. Update messages are propagated in the network until the system stabilizes. All the experiments were run on a Linux based intranet on a 100 M-bit Ethernet network with up to 40 peers.

Table 7. TREC .GOV dataset properties.

Number of links	11164830
Number of documents	1227038
Number of hosts	7792
Link distribution	77% links within hosts
	23% links across hosts

6.1.2. *Web Server*

We used a simple and lightweight web server called thttpd in our web server experiments. The pagerank computation was written as a separate program that listens for pagerank update messages, updates the pageranks at the web server, and sends subsequent update messages to other web servers. A pagerank update message is simply an HTTP POST request which executes a CGI-BIN program on the recipient web server transferring the actual pagerank values as input. The update messages corresponding to all the outlinks from one web server to another are sent in a single message. The computation is initiated by starting the pagerank computation programs at each web server. All our experiments were run on a Linux based intranet on a 100 M-bit network. All machines were Pentium III or Pentium IV class machines, running Debian Linux. We used 64 such machines, running 4 web servers on each machine, effectively giving us a total of 256 web servers.

6.2. *Methodology and Datasets*

For all our P2P experiments we synthesized graphs representing the link structure using the power law relation: number of nodes with degree i is proportional to $1/i^k$. The documents are randomly assigned to peers approximately following the host and domain locality properties observed by Bharat et al. [3]. The graphs generated and the methodology used here is the same as in the previous section. We use the number of pagerank update messages per document generated as our performance metric, and where applicable report the execution times also.

For our web server experiments we used the datasets from the TREC .GOV [12] dataset to obtain the document link structure. This dataset is a crawl of .gov sites of the Internet, and its properties are shown in Table 7. We randomly mapped the 7792 hosts down to our available 256 web servers and assigned the documents to web servers accordingly. For all our experiments we used an error threshold of 0.001. This

Table 8. CoorSet P2P system experimental results.

Exponent		
	1000 documents, 8 peers	
	Execution time (seconds)	Number of messages per document
1.9	75.7	16.7
2.1	44.4	9.8
2.3	26.5	5.8
	5000 documents, 40 peers	
	Execution time (seconds)	Number of messages per document
1.9	326.8	26.2
2.1	85.9	13.0
2.3	58.2	7.5

Table 9. CoorSet P2P system experimental results with dynamic activity. Experiments run on 40 peers.

Average number of update messages per document for convergence of 2000 documents	17
Additional update messages per document generated when doubling document set *instantaneously*	3.2
Additional update messages per document generated when doubling document set *gradually*	8.6
Update messages per document generated when adding 1 document (averaged over 25 additions)	0.5

Table 10. Web server experimental results.

Execution time	57 minutes
Total number of update messages	5854101
Number of update messages per document	4.8

dataset was too large to be hosted on our P2P implementation and hence the two dataset strategy. A reduced version of this dataset was simulated in our P2P system and the results were found to match.

6.3. *Results*

6.3.1. *P2P Implementation*

We studied document sets of size 1000 and 5000, with the number of peers per document fixed at 125 in both cases. We evaluated 3 different graphs, with the exponent in the power law set to 1.9, 2.1, and 2.3. Table 8 summarizes our results. As the exponent increases, the graph becomes denser, hence the message traffic and the execution times rapidly fall.

To evaluate a dynamic system, we started with a document set size of 2500 distributed across 40 peers and simulated three types of controlled dynamic activity as shown in Table 9. All experiments were performed for graphs with exponent 2.1, and we measure performance again in terms of number of update messages number documents in the network.

First, we doubled the document size instantaneously by adding a set of 2500 documents after the initial 2500 documents converged. This is equivalent to computing the pageranks with a close initial solution. An average of 3.2 additional update messages per document was generated, compared to 13 messages per document, when computing the pageranks for all 5000 documents from scratch.

Second, we gradually increased the document set to 5000 by adding documents at random intervals between 20 and 40 seconds. As documents get added they generated update messages, because of the outlinks they contain and because they modify the ranks of the documents previously present on the host which in turn generate their own messages. Doubling the document size in this fashion created an average of 8.6 additional update messages per document.

To isolate and quantify the effect of adding a single document entering the network, we measured the number of messages generated when a single document is added to an existing 2000 document set. We run 25 executions, and for each execution, randomly added one document, and counted the number of messages generated until convergence. On average a total 1109 messages was generated, amounting to 0.5 messages per document in the document set.

6.3.2. *Web Server*

Our results for our web server implementation also closely correlate with our algorithmic evaluation in Section 5. The total execution time for the .GOV dataset with approximately 1.25 million documents, distributed across 256 servers, was 57 minutes. The computation was dominated by the communication time, with the actual computation time at each web server being of the order of seconds. The total number of update messages sent was 5.8 million, resulting in an average of about 5 messages per document. Table 10 summarizes our experimental results. This message count is much lower compared to what we obtained in Section 5, and is due to the link distribution. In the TREC .GOV dataset most of the links are

within the same host (77%) and hence do not result in update messages. Such a distribution has been independently observed by other researchers as well [3]. The link distribution used in the simulated P2P system in Section 5 did not model link locality.

The documents in web servers follow a Zipf distribution with a few web servers accounting for a large number of documents [6]. Correspondingly the pagerank update messages showed a pattern consistent with such a structure, with a few set of web servers accounting for most of the update messages. Figure 4 shows the distribution of the communication between the 256 web servers, which have a total of 65536 pair-wise server links. The x-axis shows the percentage of such communication links that account for the total percentage of the messages. As seen from the graph, about 4% of these pair-wise links are never used, and less than 5%, i.e. not more 25 web servers account for more than half the total message traffic, and less 8 web servers (1.5%) account for more than 30% of the total message traffic.

The pagerank algorithm implemented used two optimizations to reduce message traffic. First, instead of sending the update message, generated by each document as a separate message, all the update messages from one server to another are grouped together and sent as one HTTP request. Second, when a group of update messages is received, each web server performs 30 internal passes, aggregates the update messages, and then sends them to the remote web servers. This results in a tremendous amount of traffic reduction, as opposed to doing just one local iteration at each web server.

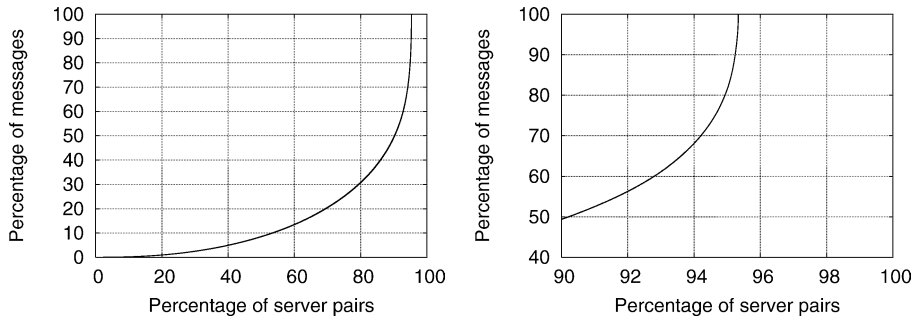
6.4. *Summary*

The results from the simulation model execution reported in Section 5 and the two implementations described and evaluated in this section suggest the following. First, distributed pagerank computation is a practical and feasible solution for assigning document relevance in peer-to-peer networks and can be effectively combined with incremental keyword search algorithms to provide high quality P2P search. Second, the web server implementation demonstrates the feasibility of the algorithm in web server environments, our prototype implementation uses a readily available lightweight web server and the standard HTTP protocol and CGI-BIN interface and consumes minimal processor and communication resources.

7. Related Work

The papers reporting research most closely related to the distributed asynchronous computation of pagereanks reported here are *Adaptive On-line Page Importance Computation* [1] and *The Eigentrust Algorithm for Reputation Management in Peer-to-Peer Networks*' [25]. Abiteboul et al. [1] report a crawler-based, distributed sequential implementation of pageranks. The algorithm executes the computation of pageranks by crawling the Web. As the crawler encounters a page, it finds the outlinks on the page and sends update messages distributing the current pagerank of the page among its outlinks. The pageranks are therefore computed in place. This algorithm is a sequential implementation of asynchronous iteration. These authors did not connect their algorithm with asynchronous iteration. In the second paper mentioned above, Kamvar et al. [25] report a distributed synchronous algorithm for computation of a quantity similar to pagerank, a reputation trust. The algorithm is operationally similar to a synchronous version of a distributed computation pagerank. The results of the computation are left in place in the network.

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, 19, 21, 22]. It appears on the basis of our limited results that the asynchronous iteration may converge more rapidly than the acceleration methods studied in [22]. Cheng and Zhang [10] studied computation of pagerank using asynchronous iterations, compared synchronous and asynchronous iteration on 128 processors, and found that asynchronous iteration was more efficient. The algorithm in [24] exploiting the block structure of document links to compute 'local pagerank vectors' which are then used as the starting solution for the standard pagerank computation suggests asynchronous iteration. Arasu et al. [2] use values from current iterations as they become available rather than using only values from the previous iteration which again suggests asynchronous iteration. The algorithm in [23] where the pagerank matrix is partitioned such that converged pageranks are not included in future iterations occurs naturally in the distributed asynchronous algorithm. There have recently been many reports [40, 39, 17, 27, 16] of systems supporting numerical computations on P2P networks or P2P-like systems but these have apparently not been applied to distributed computation of pageranks.



(a) (b)

Figure 4. Message traffic distribution between web servers. Graph (b) shows same data as graph (a), with more x-axis detail.

8. Summary, Conclusions and a Question

This paper proposes and evaluates a distributed algorithm based on asynchronous iteration which enables computations of pageranks in P2P and P2P-like networks and applies it to the problem of multiple word keyword searches in peer-to-peer networks. Evaluation of the algorithm is accomplished though three implementations: on a simulated P2P system, on a broadcast based P2P system, and on a network of web servers. The algorithm is simple and straightforward to implement and computationally lightweight. For P2P systems small modifications are required in the messaging layer to allow the sending and receiving of pagerank messages. To implement the distributed pagerank algorithm in a web server environment, each web server is augmented with the pagerank computation engine and the HTTP protocol is used to exchange messages.

The distributed algorithm converges rapidly, produces high quality pageranks, and enables incremental and continuous computation of pageranks as documents are added and deleted. The distributed implementation removes the need for periodic crawls of the network and requires no central server farm for pagerank computation.

The pageranks are computed in place on the peers of the P2P system (or network of web servers) so that the computed pageranks can be coupled with distributed keyword indexes to enable effective keyword search in P2P systems. An incremental search mechanism for P2P networks based on coupling pageranks with distributed indexes which provides a ten-fold network traffic reduction on multi-keyword search queries in P2P networks was defined and evaluated.

As far as we can tell the computation of pageranks done for this paper are the first substantial numerical computations done on a P2P network using a purely distributed algorithm.

The primary goal for the distributed pagerank computation was to enable effective keyword search peer in peer networks. The web server implementation suggests an effective, low-cost implementation of keyword search for intranets to implement a search capability for their local document datasets.

The distributed pagerank computation algorithm can also be extended in a straightforward manner to the World Wide Web and its vast corpus of web pages. But efficient keyword search requires both distributed pageranks and a distributed keyword index. This raises the interesting research question: ‘Can the entire Internet keyword index be computed and stored in a distributed manner?’ Coupling such distributed keyword index with the web server computed distributed pagerank, could enable a fully distributed, pagerank-based, keyword search for the Internet.

Acknowledgments

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. S. Abiteboul, M. Preda and G. Cobna, “Adaptive On-Line Page Importance Computation”, in *Proceedings of the 12th World Wide Web Conference*, 2003, pp. 280–290.
2. A. Arasu, J. Novak, A. Tomkins and J. Tomlin, “PageRank Computation and the Structure of the Web: Experiments and Algorithms”, in *Proceedings of the 11th International World Wide Web Conference*, Poster Track, 2002.

3. K. Bharat, B.-W. Chang, M.R. Henzinger and M. Ruhl, "Who Links to Whom: Mining Linkage between Web Sites", in *Proceedings of the IEEE International Conference on Data Mining*, 2001, pp. 51–58.
4. K. Blathras, D.B. Szyld and Y. Shi, "Timing Models and Local Stopping Criteria for Asynchronous Iterative Algortihms", *Journal of Parallel and Distributed Computing*, Vol. 58, pp. 446–465, 1999.
5. B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors", *Communications of the ACM*, Vol. 13, No. 7, pp. 422–426, 1970.
6. L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", in *INFOCOM (1)*, 1999, pp. 126–134.
7. 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.
8. J.C. Browne, K. Kane and H. Tian, "An Associative Broadcast Based Coordination Model for Distributed Processes", in *Proceedings of 5th International Conference on Coordination Models and Languages 2002* (also to appear in a volume in the Springer-Verlag LNCS Series), 2002, pp. 96–110.
9. D. Chazan and W. Miranker, "Chaotic Relaxation", *Linear Algebra Applications*, Vol. 2, pp. 199–222, 1969.
10. Y. Cheng and H. Zhang, "Parallelization of the Page Ranking in the Google Search Engine", 2003, http://manip.crhc.uiuc.edu/chen/pagerank.ps
11. I. Clarke, T.W. Hong, S.G. Miller, O. Sandberg and B. Wiley, "Protecting Free Expression Online with Freenet", *IEEE Internet Computing*, Vol. 6, No. 1, pp. 40–49, 2002.
12. CSIRO, "TREC .GOV Collection", 2002, http://www.ted.cmis.csiro.au/ TRECWeb/govinfo.html
13. L. Elden, "A Note on the Eigenvalues of the Google Matrix", Technical Report LiTH-MAT-R-04-01, Linkoping University, 2004.
14. A. Frommer and D.B. Szyld, "On Asynchronous Iterations", *Journal of Computational and Applied Mathemathics*, Vol. 123, No. 1, pp. 201–216, 2000.
15. O.D. Gnawali, "A Keyword-Set Search System for Peer-to-Peer Networks", M.E. thesis, Department of Electrical Engineering and Computer Science, MIT, 2002.
16. M. Govindaraju, S. Krishnan, K. Chiu, A. Slominski, D. Gannon and R. Bramley, "XCAT 2.0: A Component-Based Programming Model for Grid Web Services", Technical Report TR562, Department of Computer Science, Indiana University, 2002.
17. J. Hau, W. Lee and S. Newhouse, "Autonomic Service Adaptation in ICENI using Ontological Annotation", in *Proceedings of 4th International Workshop on Grid Computing, Grid 2003*, 2003.
18. T. Haveliwala, "Efficient Computation of PageRank", Technical Report 1999-31, Stanford University, 1999.
19. T. Haveliwala, "Topic Senstive Pagerank", in *Proceedings of the 11th International World Wide Web Conference*, 2002.
20. T. Haveliwala and S. Kamvar, "The Second Eigenvalue of the Google Matrix", Technical Report 2003-20, Stanford University, 2003.
21. G. Jeh and J. Widom, "Scaling Personalized Web Search", in *Proceedings of the 12th International World Wide Web Conference*, 2003.
22. S. Kamvar, T. Haveliwala, C. Manning and G. Golub, "Extrapolation Methods for Accelerating PageRank Computations", in *Proceedings of the 12th International World Wide Web Conference*, 2003.
23. S.D. Kamvar, T.H. Haveliwala, C.D. Manning and G.H. Golub, "Adaptive Methods for the Computation of PageRank", *Linear Algebra and its Applications*, Special Issue on the Numerical Solution of Markov Chains, 2003, November (to appear).
24. S.D. Kamvar, T.H. Haveliwala, C.D. Manning and G.H. Golub, "Exploiting the Block Structure of the Web for Computing PageRank", Technical Report, Stanford University, 2003.
25. S.D. Kamvar, M.T. Schlosser and H. Garcia-Molina, "The Eigentrust Algorithm for Reputation Management in P2P Networks", in: *Proceedings of the 12th World Wide Web Conference*, pp. 640–651, 2003.
26. K. Kane and J.C. Browne, "CoorSet: A Development Environment for Associatively Coordinated Components", in *Proceedings of Coordination 2004* (also to appear in a volume in the Springer-Verlag LNCS Series), 2004.
27. Kelly, W. and L. Frische, "G2 Remoting: A Cycle Stealing Framework Based on .NET Remoting", in *Proceedings of the 2003 APAC Conference on Advanced Computing, Grid Applications and eResearch*, 2003.
28. A.Z. Kronfol, "FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine", Technical Report, Princeton, 2002.
29. C. Moler, "The World's Largest Matrix Computation", 2002, http://www.mathworks.com/company/newsletter/clevescorner/oct02_cleve.shtml
30. L. Page, S. Brin, R. Motwani and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web", Technical Report, Stanford University, 1998.
31. Pagerank, "PageRank Explained", 2002, http://www.webrankinfo.com/english/pagerank/
32. J. Poskanzer, "thttpd", 2003. http://www.acme.com/software/thttpd/
33. S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A Scalable Content Addressable Network", in *Proceedings of ACM SIGCOMM*, 2001.
34. P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching", in *Proceedings of Middleware 2003, LNCS 2672, Springer*, 2003, pp. 21–40.
35. 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)*, 2001, pp. 329–350.
36. K. Sankaralingam, S. Sethumadhavan and J.C. Browne, "Distributed Pagerank for P2P Systems", in *Proceedings of the 12th International Symposium on High Performance Distributed Computing*, 2003, pp. 58–68.
37. 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*, 2001, pp. 149–160.
38. J.C. Strikwerda, "A Probabilistic Analysis of Asynchronous Iteration", *Linear Algebra and its Applications*, Vol. 349, pp. 125–154, 2002.
39. I. Taylor, M. Shields, I. Wang and R. Philp, "Distributed P2P Computing within Triana: A Galaxy Visualization Test Case", in *Proceedings of IPDPS*, 2003.
40. 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*, 2002.