Computing PageRank in a Distributed Internet Search System

Yuan Wang   David J. DeWitt

Computer Sciences Department, University of Wisconsin - Madison
1210 W. Dayton St.
Madison, WI 53706
USA
{yuanwang, dewitt}@cs.wisc.edu

Abstract

Existing Internet search engines use web crawlers to download data from the Web. Page quality is measured on central servers, where user queries are also processed. This paper argues that using crawlers has a list of disadvantages. Most importantly, crawlers do not scale. Even Google, the leading search engine, indexes less than 1% of the entire Web. This paper proposes a distributed search engine framework, in which every web server answers queries over its own data. Results from multiple web servers will be merged to generate a ranked hyperlink list on the submitting server. This paper presents a series of algorithms that compute *PageRank* in such framework. The preliminary experiments on a real data set demonstrate that the system achieves comparable accuracy on *PageRank* vectors to Google's well-known *PageRank* algorithm and, therefore, high quality of query results.

1. Introduction

Internet search engines, such as Google™, use web crawlers (also called web robots, spiders, or wanderers) to download data from the Web [3]. The crawled data is stored on centralized servers, where it is parsed and indexed. Most search engines employ certain connectivity-based algorithms to measure the quality of each individual page so that users will receive a ranked page list for their queries. For instance, Google computes *PageRank* [22] to evaluate the importance of pages. Thus, the size of the crawled web data repository has two impacts on the results of a query. First, more qualified results may be found in a larger data set. Second, more web pages will provide a bigger link graph which, in turn, will result in a more accurate *PageRank* computation.

However, there are several limitations of using web crawlers to collect data for search engines:

- ***Not Scalable***. According to a survey [21] released by Netcraft.com in February 2004, there are more than 47 million web servers hosting the contents in the Internet. Based on another study [19] released by Lyman et al. in 2003, it was estimated that the Web consisted of 8.9 billion pages in the "surface web" (public available static pages) and about 4,900 billion pages in the "deep web" (specialized Web-accessible databases and dynamic web sites) in year 2002[1]! The numbers have been growing even faster since. In comparison, Google indexes "only" 4.3 billion pages[2]. Even with a distributed crawling system [3], it is still impossible to consider downloading a large portion of the Web.
- ***Slow Update***. Web crawlers are not capable of providing up-to-date information in the Web scale. For instance, it is estimated that Google refreshes its data set once every two to four weeks, with the exception of Google™ News, which covers "only" 4,500 sources.
- ***Hidden (Deep) Web***. It is very difficult, if not impossible, for web crawlers to retrieve data that is stored in a database system of a web site that presents users with dynamically generated html pages.
- ***Robot Exclusion Rule***. Web crawlers are expected to observe the robot exclusion protocol [18], which advises crawlers not to visit certain

[1] 167 TB in surface web, 91,850 TB in deep web, 18.7 KB per page [19].

[2] Claimed on http://www.google.com as of June 2004.

Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004

directories or pages on a web server to avoid heavy traffic. Nevertheless, the protocol does not affect human beings surfing on the Internet. Thus, the crawled data set is not "complete" and conflicts with those connectivity-based page quality measures, such as *PageRank*, which is based on the "Random Surfer Model" [22]. Thus, an incomplete data set may result in a loss of accuracy in the *PageRank* computation.

- ***High Maintenance***. It is difficult to write efficient and robust web crawlers. It also requires significant resources to test and maintain them [3].

In fact, besides web crawlers, centralized Internet search engine systems also face other challenges. A successful search engine system requires a large data cache with tens of thousands of processors to create inverted text indices, to measure page quality, and to execute user queries. Also, centralized systems are vulnerable to point failures and network problems, and thus must be replicated. For example, Google employs a cluster of more than 15,000 PCs and replicates each of its internal services across multiple machines and multiple geographically distributed sites [1].

This paper proposes a distributed Internet search engine framework that addresses the above problems. With such a framework, there are no dedicated centralized servers. Instead, every web server participates as an individual search engine over its own (local) data so that crawlers are no longer needed. User queries are processed at related web servers and results will be merged at the client side.

Since Google is by far the most utilized search engine, The framework presented in this paper is based on Google's *PageRank* algorithm. This paper introduces a series of variants of this algorithm that are used in the system. The goal of this paper is to present an efficient strategy to compute *PageRank* in a distributed environment without having all pages at a single location. The approach employs of the following steps,

1. *Local PageRank* vectors are computed on each web server individually in a distributed fashion.
2. The relative importance of different web servers is measured by computing the *ServerRank* vector.
3. The *Local PageRank* vectors are then refined using the *ServerRank* vector. Query results on a web server are rated by its *Local PageRank* vector.

This approach avoids computing the complete global *PageRank* vector. (Consider the 4.3 billion pages indexed by Google, the *PageRank* vector itself is 17 GB in size, even without including the size of the web link graph.) When a user query is executed by a web server, the result is ranked by the server's *Local PageRank* vector. As results from multiple servers are received by the server to which the query was originally submitted, they are merged and ranked by their *Local PageRank* values and *ServerRank* values to produce the final result list.

A real web data set was collected and used to evaluate the different *PageRank* algorithms. The preliminary experiments demonstrate that the *Local PageRank* vectors are very "close" to their corresponding segments in the global *PageRank* vector computed using Google's *PageRank* algorithm. They also show that the query results achieved are comparable in quality to those obtained using the centralized Google Algorithm.

The remainder of the paper is organized as follows. Section 2 describes the data set that is used throughout the paper for *PageRank* computation and query execution. The collection of algorithms is formulated in Section 3, along with experiments for each step and evaluation of Local PageRank and ServerRank vectors against the "true global" PageRank vector computed using the standard Google's *PageRank* algorithm. Section 4 presents more query results and evaluation. Section 5 summarizes the conclusions and discusses future research directions.

2. Experimental Setup

The following sections use some real web data to evaluate the proposed distributed search engine scheme. Since the authors do not have control over the web servers from which the pages were collected, a local copy of the data had to be made.

Since the scope of Internet search engines, such as Google, is the entire Web, ideally, the experiments would be conducted using all the data on the Web. This is obviously impossible. What is needed is a relatively small subset that resembles the Web as a whole. For the experiments described in this paper, the authors crawled over the Stanford.edu domain. The major characteristics of the data set are:

- The crawl was performed in October 2003, starting from "http://www.stanford.edu", in the breadth-first fashion as described by Najork and Wiener [20], in an effort to obtain high-quality pages.
- The crawl is limited to the stanford.edu domain and all out-of-domain hyperlinks are removed.
- If the data set is viewed as a breadth-first search tree, it has 8 levels of pages and thus, 9 levels of URLs[3].
- In the raw data set (15.8 GB), there are 1,168,140 unique pages. Since the experiments only performed PageRank computation and title search queries (explained in Section 4) over the dataset, only those pages of "text/html" type, all 1,168,140

[3] It was not an exhaustive crawl because it was asked to stop when it just finished downloading 8 levels of pages.

of them, were obtained from the crawl. Other pages, such as PDF files, images, etc., only appear as hyperlinks in the data set.

- The crawler visited 1,506 different logical domains that are hosted by 630 unique web servers (identified by their IP addresses) within the Stanford.edu domain.
- The crawler did not observe the robot rule in an effort to try to get the complete page set of the domain.

In order to create an accurate web link graph, certain data cleaning procedures were applied to the raw data set. For example, URLs that are literally different but lead to the same page were identified[4]. For such URLs, only one is retained throughout the data set in order to avoid duplicates. Also, URL redirections had to be recognized so that corresponding URLs could be corrected.

The cleaned data set consists of 630 hosts (i.e. web servers), 1,049,901 pages, and 4,979,587 unique hyperlinks. Figure 1 shows the distribution of the size (number of pages and number of hyperlinks) of these web servers[5]. For instance, 0.5% of servers host more than 100,000 pages and 4.6% of servers host more than 100,000 URLs.

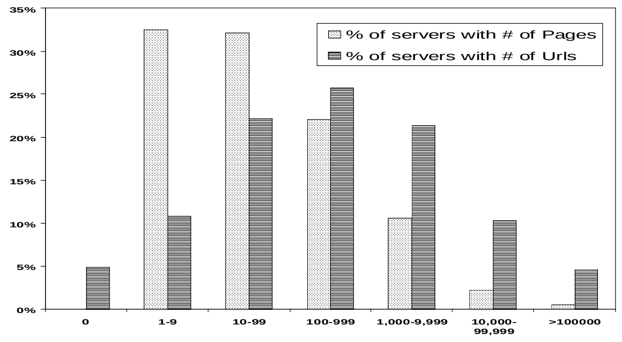

Figure 1: Histogram of Distribution over server size. The *x*-axis gives the magnitude of the number of pages or urls hosted by a web server, and the *y*-axis shows the fraction of web servers of the size.

3. The Framework

The goal is to distribute the search engine workload to every web server in the Internet, while still obtaining high-quality query results compared to those that a centralized search engine system obtain. This goal would be achieved by installing a shrunk version of the Google search engine on every web server which only answers queries against the data stored locally. Results from different web servers are merged locally to produce a ranked hyperlink list. Ideally this list would be identical to the result returned by a centralized system for the same data set.

It takes three steps to process a user query, namely query routing, local query execution, and result fusion.

Query Routing. In the distributed search engine scenario, every web server is equipped with a search engine, so users can submit their queries to any web server. For example, a Stanford computer science graduate student might submit his query on www.cs.stanford.edu, which, in turn, sends the query to other web servers that host the relevant pages.

(Local) Query Execution. When a web server receives a query that has been relayed from another web server, it processes the query over its local data and sends the result, a ranked URL list, back to the submitting web server.

Result Fusion. Once results from other web servers have been obtained, they are merged into a single ranked URL list to be presented to the user.

This paper focuses on how queries are executed on each web server and how to generate a ranked result list. Later in this section, related issues will be briefly discussed, which include typical routing strategies and how to improve them in order to obtain top-*k* results faster.

Section 3.1 briefly reviews the original *PageRank* algorithm. Section 3.2 explores the web link structure and explains the data locality feature that enables the distributed execution of search engine queries. A new *PageRank* computation strategy is introduced in Section 3.3 and Section 3.4 describes the metrics that are used to evaluate the algorithms. In the following sections, 3.5 through 3.8, a series of modified *PageRank* algorithms are proposed, accompanied by experiments for evaluation. Section 3.9 discusses a few other issues in the framework, such as query routing and data updates.

3.1 PageRank Review

Google uses *PageRank*, to measure the importance of web pages, which is based on the linking structure of the Web. Page and Brin [3][22] consider the basis of *PageRank* a model of user behavior, the "Random Surfer Model", where a web surfer clicks on links at random with no regard towards content. The random surfer visits a web page with a certain probability which is derived from the page's *PageRank*.

In fact, the *PageRank* value of a page is defined by the *PageRank* values of all pages, $T_1, \ldots, T_n$, that link to it, and a damping factor[6], d, that simulates a user randomly jumping to another page without following any hyperlinks [3], where $C(T_i)$ is the number of outgoing links of page T_i.

$$PR(A) = (1-d) + d\left(\frac{PR(T_1)}{C(T_1)} + \ldots + \frac{PR(T_n)}{C(T_n)}\right)$$

The *PageRank* algorithm is formally defined in

[4] E.g., "www.stanford.edu", "www.stanford.edu/" and "www.stanford.edu/index.html" represent the same page.

[5] Based on the crawled data set.

[6] The value of d was taken to be 0.85 in [22]. We use this value in all experiments in this paper.

[14][15] as follows,

```
Function **pageRank** $(G, \vec{x}^{(0)}, \vec{v})$ {
    Construct $P$ from $G$: $P_{ji} = 1/\deg(j)$;
    **repeat**
        $\vec{x}^{(k+1)} = dP^T \vec{x}^{(k)}$;
        $w = \left\|\vec{x}^{(k)}\right\|_1 - \left\|\vec{x}^{(k+1)}\right\|_1$;
        $\vec{x}^{(k+1)} = \vec{x}^{(k+1)} + w\vec{v}$;
        $\delta = \left\|\vec{x}^{(k+1)} - \vec{x}^{(k)}\right\|_1$;
    **until** $\delta < \varepsilon$;
    **return** $\vec{x}^{(k+1)}$;
}
```

Figure 2: The PageRank Algorithm.

G is the directed web link graph of *n* pages (i.e. *n* URLs), where every vertex represents a unique URL in the Web. Let $i \to j$ denote the existence of a link from page *i* to page *j*, i.e. URL *j* appears on page *i*. Then, P is the $n \times n$ stochastic transition matrix describing the transition from page *i* to page *j*, where P_{ij}, defined as 1/deg(*i*), is the possibility of jumping from page *i* to page *j*. Let $\vec{v}$ be the n-dimensional column vector representing a uniform probability distribution over all pages:

$$\vec{v} = \left[\frac{1}{n}\right]_{n\times 1}$$

The standard *PageRank* algorithm starts with the uniform distribution, i.e., $\vec{x}^{(0)} = \vec{v}$. The algorithm uses the power method to converge, when the L_1 residual, δ, of vectors of two consecutive runs is less than a preset value ε[7]. The result vector is the principal eigenvector of a matrix derived from *P* (see details in [15]). Let $\vec{\gamma}_g$ denote the global *PageRank* vector computed over *G*, the global web link graph. $\vec{\gamma}_g$ is also referred as the "true global" *PageRank* vector in this paper.

Note that *PageRank* is not the only factor in determining the relevance of a page to a query. Google considers other factors, including the number of occurrences of a term on a page, if terms in the query appear in the page title, or anchor text, if the terms are in large font, etc., to produce an IR score for the page. Then, the IR score for a page is combined with its *PageRank* value to produce the final rank value for the page. The algorithm for computing the IR scores is a secret. Since it is orthogonal to the *PageRank* problem, it will not be discussed in this paper.

3.2 The Server Linkage Structure

Intuitively, it would seem that all pages within a domain (e.g. www.cs.stanford.edu) have a stronger connection with each other through their intra-domain hyperlinks than their connections with pages out of their domain. Bharat et al. [2] investigated the topology of the Web link graph, focusing on the linkage between web sites. They introduced a notion of "hostgraph" to study connectivity properties of hosts and domains over time. Kamvar et al. [14] studied the block structure of the Web. They found that there are clearly nested blocks corresponding to domains, where the individual blocks are much smaller than the entire Web.

Out of the 1,049,271 pages in our test data set, 865,765 (82.5%) pages contain intra-server hyperlinks and only 255,856 (24.4%) pages contain inter-server hyperlinks. 96.6% (26,127,126) of the links are intra-server while 3.4% (908,788) are inter-server. After removing duplicates, there are 21,604,663 (96.3%) intra-server links and 833,010 (3.7%) inter-server links. Figure 3 shows that most servers have very few inter-server links while servers with large numbers of intra-server links are very common.

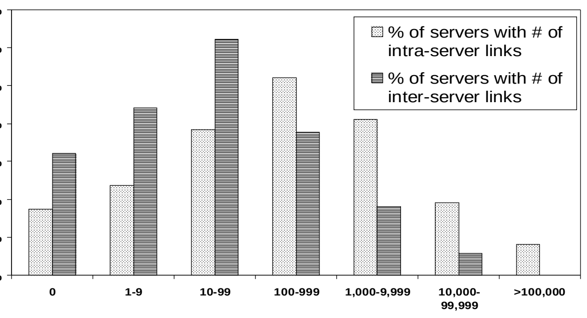

Figure 3: Histogram of Distribution over server hyperlinks. The *x*-axis gives the magnitude of the number of intra-server and inter-server hyperlinks hosted by a web server, and the *y*-axis shows the fraction of web servers of that size.

Notice that an inter-server hyperlink often points to the top page (or entry point page) of a domain, such as http://www.cs.stanford.edu. Among the 908,788 inter-server links, there are 222, 393 (24.5%) top-page links, or 187,261 (22.5%) links after removing the duplicates. Such inter-server links do not affect the relative importance of pages within a web site.

It is possible for a web server to host multiple independent web sites that do not interlink each other to a significant extent. For instance, one server in the data set, "proxy-service.lb-a.stanford.edu", hosts as many as 285 web sites. Such case is treated as a single server to avoid an explosion in the number of servers when doing *Server-Rank* calculation. Notice that it will generate more accurate result if those web sites are treated individually.

3.3 Overview of Distributed PageRank Algorithms

The topology of the web linkage structure suggests that connectivity-based page importance measures can be computed at individual web servers, i.e., every web server can independently compute a "*Local PageRank*" vector

[7] It is set to be 0.0001 in all experiments in this paper.

over its local pages. Since the majority of links in the web link graph are intra-server links, the relative rankings between most pages within a server are determined by the intra-server links. So the result of local query execution is likely comparable to its corresponding sublist of the result obtained using the global *PageRank* algorithm.

The inter-server links can be used to compute "*ServerRank*", which measures the relative importance of the different web servers. Both *Local PageRank* and *ServerRank* are used in combination to merge query results from multiple sites into a single, ranked hyperlink list.

The outline of the algorithm follows:

1. Each web server constructs a web link graph based on its own pages to compute its "*Local PageRank*" vector (Section 3.5).
2. Web servers exchange their inter-server hyperlink information with each other and compute a "*ServerRank*" vector (Section 3.6).
3. Web servers use the "*ServerRank*" vector to refine their "*Local PageRank*" vectors, which are actually used for local query execution (Section 3.7).
4. After receiving the results of a query from multiple sites, the submitting server uses the "*ServerRank*" vector and the "*Local PageRank*" values that are associated with the results to generate the final result list (Section 3.8).

Each step is described in detail in the following sections. Notice, that for static data sets, both the *Local PageRank* vectors and the *ServerRank* vector need to be only computed once. As shown later, all algorithms are efficient and can be exercised frequently in case of updates.

3.4 Evaluation Metrics

The goal is to apply the *PageRank* algorithm in a distributed Internet search engine system, where it should be able to provide users the same quality results as what the original algorithm does, without incurring the cost of a centralized search system. Judging the search quality of Google is not the focus of this paper; rather *PageRank* vectors and query results, generated by the algorithms presented in this paper, can be compared against those computed using the Google algorithm. Basically, given the same data set, the distributed search engine system is expected to return a very similar, if not identical, ranked page list to the results obtained in a centralized fashion using the Google *PageRank* algorithm. In this section, several metrics are described, which can be used to compare two ranked lists.

Suppose a *PageRank* vector $\vec{\gamma}$ is computed over a page set (domain) D, and p is the corresponding ranked page list, which is a permutation from D. Let $p(i)$ denote the position (or rank) of page i in p, and page i is "ahead" of page j in p if $p(i) < p(j)$. $P_D = \{\{i, j\} \mid i \neq j \textit{ and } i, j \in D\}$ is also defined to be the set of unordered pairs of all distinct pages in the domain D.

Given two *PageRank* vectors $\vec{\gamma}_1$ and $\vec{\gamma}_2$ on D, and their respective ranked page lists, p_1 and p_2, Kendall's τ metric [16] is then defined as:

$$K(p_1, p_2) = \sum_{\{i,j\} \in P_D} K_{\{i,j\}}(p_1, p_2),$$

where $K_{\{i,j\}}(p_1, p_2) = 1$ if i and j are in different order in p_1 and p_2; otherwise, $K_{\{i,j\}}(p_1, p_2) = 0$.

To measure the similarity between p_1 and p_2, Kendall's τ–distance [9][16] is defined as follows:

$$KDist(p_1, p_2) = \frac{K(p_1, p_2)}{|D| \times (|D| - 1)/2}$$

Notice the maximum value of $KDist(p_1, p_2)$ is 1 when p_2 is the reverse of p_1.

In practice, people are usually more interested in the top-k results of a search query. Kendall's τ–distance, however, cannot be computed directly between two top-k ranked page lists because they are unlikely to have the same set of elements. In [9], Fagin et al. generalize the Kendall's τ–distance to be able to handle this case.

Suppose $p_1^{(k)}$ and $p_2^{(k)}$ are the top-k lists of p_1 and p_2. The minimizing Kendall's τ metric is defined as:

$$K_{\min}(p_1^{(k)}, p_2^{(k)}) = \sum_{\{i,j\} \in P_D} K_{\{i,j\}}^{\min}(p_1^{(k)}, p_2^{(k)})$$

where $K_{\{i,j\}}^{\min}(p_1^{(k)}, p_2^{(k)}) = 0$ if both i and j appear in one top-k list but neither of them appears on the other list[8]; otherwise, $K_{\{i,j\}}^{\min}(p_1^{(k)}, p_2^{(k)}) = K_{\{i,j\}}(p_1, p_2)$.

Then, the minimizing Kendall's τ–distance [9] is defined as:

$$K_{\min}^{(k)} Dist(p_1, p_2) = \frac{K_{\min}(p_1^{(k)}, p_2^{(k)})}{k \times (k-1)/2}.$$

Another useful metric is the L_1 distance between two *PageRank* vectors, $\|\vec{\gamma}_1 - \vec{\gamma}_2\|_1$, which measures the absolute error between them.

3.5 Local PageRank

Since the majority of hyperlinks within a web site are intra-server links, intuitively, the *PageRank* vector calculated over the site's local page set may resemble its corresponding segment of the true global *PageRank* vector.

A straightforward way to compute a *Local PageRank* vector on a web server is to apply the *PageRank* algorithm on its page set after removing all inter-server hyper-

[8] This is the only case that there's not enough information to compare the ordering of i and j in p_1 and p_2. $K_{\min}$ is the optimistic choice.

links [14]. Given server-*m* that hosts n_m pages, $G_{(m)}$ ($n_m \times n_m$), the web link graph of server-*m*, is first constructed from the global web link graph G_g, where for every link $i \rightarrow j$ in G_g, it is also in $G_{(m)}$ if and only if both *i* and *j* are pages in server-*m*. That is, $G_{(m)}$ contains intra-server links only. Then, a *Local PageRank* vector of server-*m* is computed as follows:

$$\vec{\gamma}_{l(m)} = pageRank\ (G_{(m)}, \vec{v}_{n_m}, \vec{v}_{n_m}) \qquad (\textbf{\textit{LPR-1}})$$

where $\vec{v}_{n_m}$ is the n_m-dimensional uniform column vector as defined in Figure 2.

To evaluate the accuracy of *Local PageRank*, the *Local PageRank* vectors $\vec{\gamma}_{l(m)}$ of each of the web servers in the data set are computed. The true global *PageRank* vector $\vec{\gamma}_g$ is also computed on the entire data set. Let $p_{l(m)}$ be the corresponding ranked page list of $\vec{\gamma}_{l(m)}$, and p_g the global ranked page list. For every server-*m*, the elements corresponding to all of its pages from $\vec{\gamma}_g$ are taken to form $\vec{\gamma}_{g(m)}$, the corresponding vector of $\vec{\gamma}_{l(m)}$. Note that, in order to compare with $\vec{\gamma}_{l(m)}$, $\vec{\gamma}_{g(m)}$ is normalized so that its L_1 norm (the sum of all element values) is 1. Let $p_{g(m)}$ be the according ranked page list of $\vec{\gamma}_{g(m)}$.

First, the average L_1 distance between $\vec{\gamma}_{l(m)}$ and $\vec{\gamma}_{g(m)}$, $\left\|\vec{\gamma}_{l(m)} - \vec{\gamma}_{g(m)}\right\|_1$, is 0.0602. In comparison, the average L_1 distance between $\vec{v}_{n_m}$, the uniform vector, and $\vec{\gamma}_{g(m)}$ is 0.3755.

Second, the average[9] Kendall's τ–distance, $KDist(p_{l(m)}, p_{g(m)})$, is 0.00134, which is a short distance. If a server hosts 40 pages, it means that there is only 1 pair of pages mis-ordered in the *Local PageRank* list, where they are next to each other.

Finally, the average minimizing Kendall's τ–distance, $K_{\min}^{(k)}Dist(p_{l(m)}, p_{g(m)})$, is measured, which is shown in Figure 4.

The accuracy between top-*k* page lists seems worse than the accuracy between two full lists, though the distance declines quickly as *k* increases. On the one hand, because of the small size of the top-*k* lists, even one mis-ordered pair has a big impact on the distance, e.g., a Kendall's τ–distance of 0.022 corresponds to only 1 mis-order in a top-10 list. On the other hand, the most important pages in a web site usually have more incoming and outgoing inter-server links with other domains which are not considered by *LPR-1*. These links, in turn, affect the accuracy. Table 1 lists the average number of incoming and outgoing inter-server links on the lists of the top-10 and top-100 pages, compare to the number of all pages in the data set. The relatively large number of incoming links indicates that the true *PageRank* value of a top-*k* page is significantly affected by the number of out-of-domain pages that link to it.

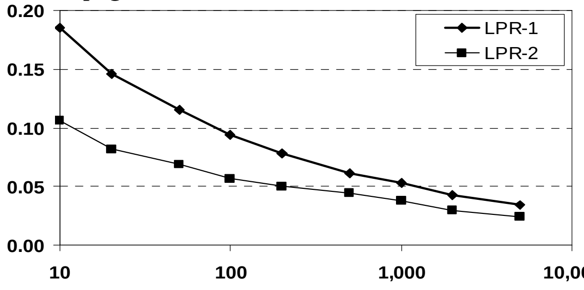

Figure 4: The minimizing Kendall's τ-distance for top-*k* page lists between $p_{l(m)}$ and $p_{g(m)}$ (k = 10 to 5,000) of algorithms *LPR-1* and *LPR-2*.

	Top-10	Top-100	All
Outgoing links	2.10	1.58	0.17
Incoming links	70.72	13.30	0.17

Table 1 Average number of inter-server links that involve top-*k* pages.

To improve the accuracy of the *Local PageRank* vectors, the authors present a slightly more complicated algorithm. As described in the next section, web servers need to exchange information about their inter-server hyperlinks to compute the *ServerRank* vector. The link information can also be used to compute more accurate *Local PageRank* vectors.

Given server-*m*, this algorithm introduces an artificial page, *ξ*, to its page set, which represents all out-of-domain pages. First, a local link graph, $G'_{(m)}$ ($(n_m + 1) \times (n_m + 1)$), is constructed from the global web link graph G_g, where for every link $i \rightarrow j$, it turns into (1) $i \rightarrow j$ if both *i* and *j* are local pages; or (2) $i \rightarrow \xi$ if *i* is a local page but *j* is not; or (3) $\xi \rightarrow j$ if *j* is a local page but *i* is not. Second, a *PageRank* vector $\vec{\gamma}'_{l(m)}$ is calculated as follows,

$$\vec{\gamma}'_{l(m)} = pageRank\ (G'_{(m)}, \vec{v}_{n_m+1}, \vec{v}_{n_m+1}) \qquad (\textbf{\textit{LPR-2}})$$

Then, the *Local PageRank* vector $\vec{\gamma}_{l(m)}$ is derived by removing *ξ* from $\vec{\gamma}'_{l(m)}$ and normalizing it.

For *LPR-2*, the average L_1 distance between $\vec{\gamma}_{l(m)}$ and $\vec{\gamma}_{g(m)}$ is only 0.0347, less than half that of algorithm *LPR-1*. The average Kendal's τ–distance, $KDist(p_{l(m)}, p_{g(m)})$, is also reduced to 0.00081, approximately 1 mis-ordering in 50 pages. Figure 4 also shows the improvement on the accuracy of the top-*k* page lists.

In order to construct $G'_{(m)}$, servers need to inform each other of their inter-server hyperlinks. For instance, server-*m* has to send a message to server-*n* that there are *x* links from *m* to *n*. Within the data set of 630 web servers,

[9] Weighted by server size.

on the average a server has outgoing links to 10.0 others, which means it needs to send 10 point-to-point messages. More details of the associated communication cost are discussed in the following sections.

3.6 ServerRank

It is encouraging that the accuracy of the *Local PageRank* vectors is improved significantly by the inclusion of some simple inter-server link information. This section presents algorithms to compute *ServerRank* that measure the relative importance of different servers. *ServerRank* is useful in two ways, refining *Local PageRank* vectors and weighing the importance of the result pages from different servers.

Similar to how the relative importance of different pages is measured by their connections with each other, *ServerRank* can be computed on the inter-server links between servers. However, unlike the *Local PageRank* computation, which can be performed individually by each server without contacting others (algorithm *LPR-1*), to calculate the *ServerRank*, servers must exchange their inter-server hyperlink information. The communication cost will be discussed after the algorithms are presented.

First, the server link graph, G_s, is constructed, in which there are n_s servers and every server is denoted by a vertex. Given servers m and n, $m \to n$ denotes the existence of a hyperlink from a page on m to a page on n. Then, a *ServerRank* vector can be simply computed as if it were a *PageRank* vector,

$$\vec{\gamma}_s = pageRank\,(G_s, \vec{v}_{n_s}, \vec{v}_{n_s}) \qquad (\textbf{\textit{SR-1}})$$

Let p_s be the corresponding ranked server list of $\vec{\gamma}_s$.

Since there is no such "*ServerRank*" concept in Google's search engine scheme and it is an intermediate step in the framework, there are no direct ways to measure its accuracy. Intuitively, the importance of a server should correlate with the importance of the pages it hosts. Here the authors suggest to construct two "benchmark" lists to approximately check the *ServerRank* vector $\vec{\gamma}_s$ against the true global *PageRank* vector $\vec{\gamma}_g$.

- *Top_Page* server list, *Top*(p_s). Servers are ranked by the *PageRank* value of the most important page that they host, i.e. the page with the highest *PageRank* value in $\vec{\gamma}_g$.
- *PR_Sum* server list, *Sum*(p_s). Servers are ranked by the sum of the *RageRank* values of all pages that they host.

Both server lists are constructed using the global *PageRank* vector $\vec{\gamma}_g$. Table 2 shows p_s is near both of them and closer to *Top*(p_s). Notice, that the Kendall's τ–distance between *Top*(p_s) and *Sum*(p_s) is 0.025.

Algorithm *SR-1* does not distinguish the inter-server hyperlinks when constructing the server link graph. Since an outgoing hyperlink carries a certain portion of the importance (i.e. *PageRank* value) of the source page to the destination page, in turn, it also transfers some importance to the hosting server, which means that a link from a more important page likely contributes more to the destination server. In fact, the *ServerRank* vector can be computed using the *PageRank* information of the source pages of the inter-server links. Notice, that at this phase, the *Local PageRank* value of a page is the best measure of its "true" importance within its own domain, so it can be used to weight inter-server links [14].

	KDist(p_s, *Top*(p_s))	*KDist*(p_s, *Sum*(p_s))
SR-1	0.035	0.053
SR-2	0.022	0.041

Table 2: The Kendall's τ-distance between the ranked server lists.

The construction of the stochastic transition matrix *P*, in Figure 2 can be modified to accommodate any available *Local PageRank* information into the *ServerRank* computation. Given a link $m \to n$ in the server link graph G_s, its weight, denoted by $w_{m \to n}$, is defined as the sum of the *Local PageRank* values of all source pages in server-*m*,

$$w_{m \to n} = \sum_{i \in D_m, j \in D_n} lp_i, \quad \forall\, i \to j \in G_g$$

where lp_i is the *Local PageRank* value of page *i*. Then *P* is constructed as,

$$P_{mn} = \frac{w_{m \to n}}{\sum_k w_{m \to k}} \quad if\ m \to n \in G_s;\ or\ P_{mn} = 0,\ otherwise.$$

and *P* is still a stochastic transition matrix. Then the rest of the algorithm is applied to compute the *ServerRank* vector,

$$\vec{\gamma}_s = pageRank'\,(G_s, \vec{v}_{n_s}, \vec{v}_{n_s}) \qquad (\textbf{\textit{SR-2}})$$

The result ranked server list is also compared against *Top*(p_s) and *Sum*(p_s) in Table 2. It is clearly closer to those two lists than the list generated by *SR-1*.

Notice, that it is not necessary to share all inter-server hyperlinks explicitly across servers (and related *Local PageRank* values in *SR-2*) to compute the *ServerRank* vector. In fact, each server just needs to compose its row of the stochastic transition matrix *P*, which means that it sends only one message out. Suppose on average every server connects with *c* other servers through inter-server links (10.0 in the data set). In a message of algorithm *SR-1*, a server just needs to identify the *c* servers to which it connects with. In *SR-2*, the size of the message is a little bigger, which is a row in *P* as described above.

Since every server needs a copy of the *ServerRank* vector to refine its *Local PageRank* vector in the next step. One way to compute it is to let every server broadcast their messages to all other servers and compute the vector by themselves. A more efficient strategy is to elect a few capable servers to compute the vector and broadcast the result so most servers will not have to receive and process *n* messages to construct the corresponding server link graph.

3.7 Local PageRank Refinement

In this section, the *ServerRank* vector is used to refine the *Local PageRank* vectors on every web server.

The difference between a *Local PageRank* vector and its corresponding (normalized) sub-vector in the true global *PageRank* vector is caused by the lack of inter-server link information. The *ServerRank* vector indicates the relative importance of a server within the network. More inter-server hyperlink information can be added to it in order to make the *Local PageRank* vectors more accurate.

The smallest amount of information that server-*n* must share with server-*m* is the number of links from *n* to *m*, and which pages on *m* the links lead to. Then, the *Local PageRank* vector of server-*m* can be adjusted in the following way. Assuming that out of l_n inter-server hyperlinks hosted by server-*n* there are $l_{n(m_i)}$ links that lead to page *i* on server-*m*. The *Local PageRank* value of page *i*, $\gamma_{l(m)_i}$, is adjusted by transferring a portion of *PageRank* values of the links from server-*n*,

$$\gamma_{l(m)_i} = \gamma_{l(m)_i} + \frac{\gamma_{s_n}}{\gamma_{s_m}} \times \frac{l_{n(m_i)}}{l_n}$$

where γ_{s_n} and γ_{s_n} are the *ServerRank* values of *m* and *n* respectively. Then the updated *Local PageRank* vector is normalized, denoted by $\vec{\gamma}'_{l(m)}$, so that other pages will also be affected. Thus, the same web link graph $G_{(m)}$ of algorithm *LPR-1* can be used to further distribute the linkage information. Unfortunately, the loop operation in the *PageRank* algorithm (shown in Figure 2) cannot be performed here as $\vec{\gamma}'_{l(m)}$ will eventually converge to the old *Local PageRank* vector no matter what the initial input vector is because the *PageRank* vector is the principal eigenvector of a matrix, which is constructed over the web link graphs [15]. To avoid this problem, the *PageRank* algorithm can be performed for only one iteration, denoted by *pageRank_single*,

$$\vec{\gamma}_{l(m)} = pageRank\_single(G_{(m)}, \vec{\gamma}'_{l(m)}, \vec{v}_{n_m}) \quad (\textbf{\textit{LPR-Ref-1}})$$

Results from both algorithms *LPR-1* and *LPR-2* at the *Local PageRank* computation step and *ServerRank* vectors from both *SR-1* and *SR-2* are used to evaluate the refinement algorithms. Table 3 demonstrates that the accuracy of the *Local PageRank* vectors is improved, especially these vectors resulting from *LPR-1* because they do not have any inter-server link information. Only a small gain orcurs the *LPR-2*/*SR-1* combination because only a small amount of additional information is added by algorithm *SR-1* to the vectors from *LPR-2*. The improvement in vectors from algorithm *LPR-1* is greater because no inter-server link information is applied in the algorithm. Figure 5 demonstrates the improvement on the accuracy of the top-*k* page lists. Notice, that the ranking of the few most important pages improves significantly because they are usually affected by the inter-server links much more than other pages.

Local PageRank & ServerRank	$\left\Vert \vec{\gamma}_{l(m)} - \vec{\gamma}_{g(m)} \right\Vert_1$	$KDist(p_{l(m)}, p_{g(m)})$
LPR-1/SR-1	0.0391 (35%)	0.00093 (31%)
LPR-1/SR-2	0.0284 (53%)	0.00069 (49%)
LPR-2/SR-1	0.0303 (13%)	0.00071 (12%)
LPR-2/SR-2	0.0245 (29%)	0.00055 (32%)

Table 3: The Average L$_1$ distance and Kendall's τ–distance of *Local PageRank* vectors after being refined by algorithm LPR-Ref-1. (Percentage of improvement in parentheses).

If server-*n* is willing to share more information with server-*m*, more specifically, the *Local PageRank* value of the individual pages that have hyperlinks to pages on *m*, it can help server-*m* understand better the relative importance of the incoming links. In this case, server-*m*'s *Local PageRank* vector can be adjusted as the end page of an inter-server link receives a certain amount of the *Local PageRank* value of the starting page. Specifically,

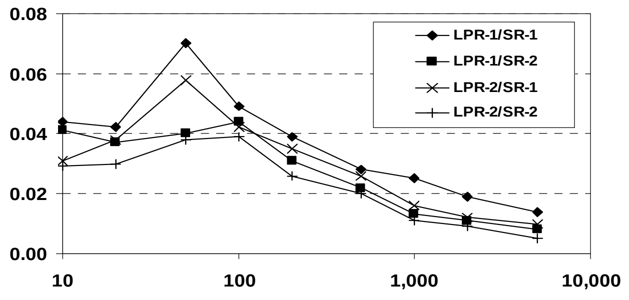

Figure 5: The minimizing Kendall's τ-distance for top-*k* page lists between refined $p_{l(m)}$ and $p_{g(m)}$ (k = 10 to 5,000), Algorithm *LPR-Ref-1*.

$$\gamma_{l(m)_i} = \gamma_{l(m)_i} + \frac{\gamma_{s_n}}{\gamma_{s_m}} \times \sum_{j \in D_n, j \to i \in G_g} \left(\gamma_{l(n)_j} \times 1 / \deg(j) \right)$$

where $\gamma_{l(n)_j}$ is the *Local PageRank* value of page *j*, which is on server-*n* and has a URL link to page *i*, which is on server-*m*. deg(*j*) is page *j*'s outgoing degree in the global link graph. Similarly, the new vector can be normalized, denoted by $\vec{\gamma}''_{l(m)}$ and applied through the single-iteration *PageRank* algorithm,

$$\vec{\gamma}_{l(m)} = pageRank\_single(G_{(m)}, \vec{\gamma}''_{l(m)}, \vec{v}_{n_m}) \quad (\textbf{\textit{LPR-Ref-2}})$$

Table 4 and Figure 6 show that with more link source information algorithm *LPR-Ref-2* significantly improves the accuracy of the *Local PageRank* vectors when compared with the results obtained using *LPR-Ref-1*.

In algorithm *LPR-Ref-1*, each server needs to send one message to every server to which it is connected with one or more hyperlinks. For instance, the message from server-*n* to server-*m* consists of all unique URLs hosted by *m* that occur on pages stored at *n*, along with the count of each link, and the count of all inter-server links on server-*n*. In the data set, it translates into an average 10 messages per server and the average message (uncompressed) size is 940 bytes. Because many URLs in a message share the same domain name, it is easy to reduce the size of each message using some prefix compression techniques. Algorithm *LPR-Ref-2* needs the same number of messages but requires more detailed information about the source page of each inter-server link, specifically the *Local PageRank* value of the source page of every link. In this case, the average message size increases to 2.1 Kbytes before compression. In both cases, the small message size means that the bandwidth requirement of the distributed *PageRank* algorithms is low. When scaled to the size of the Internet, the total number of messages will increase significantly due to the large number of web servers but the number of messages sent per server and the average message size will not increase significantly.

Local PageRank & ServerRank	$\left\|\vec{\gamma}_{l(m)} - \vec{\gamma}_{g(m)}\right\|_1$	$KDist(p_{l(m)}, p_{g(m)})$
LPR-1/SR-1	0.0282 (53%)	0.00061 (54%)
LPR-1/SR-2	0.0197 (67%)	0.00039 (71%)
LPR-2/SR-1	0.0205 (41%)	0.00049 (40%)
LPR-2/SR-2	0.0163 (53%)	0.00027 (67%)

Table 4: The Average L$_1$ distance and Kendall's τ–distance of *Local PageRank* vectors after being refined by algorithm *LPR-Ref-2*. (Percentage of improvement in parentheses).

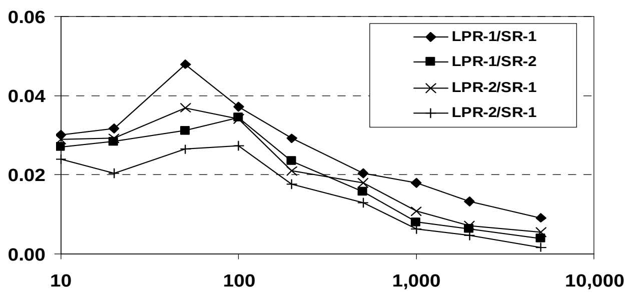

Figure 6: The minimizing Kendall's τ-distance for top-*k* page lists between refined $p_{l(m)}$ and $p_{g(m)}$ (k = 10 to 5,000), Algorithm *LPR-Ref-2*.

Since the computation of the *ServerRank* vector in algorithm *SR-2* is dependant on the *Local PageRank* vectors, the updated *Local PageRank* vectors can be used to further refine the *ServerRank* vector, etc. Multiple rounds of refinement can be applied in a relatively static environment, where hyperlinks on pages are not changed frequently.

3.8 Result Fusion

When a search query is submitted to a web server, it is forwarded to some other web servers that have relevant pages. On each receiving server, the query is executed using the server's *Local PageRank* vector, and the result, a ranked URL list, is sent back to the server to which the query was initially submitted. Then, the server performs *Result Fusion* (*RF*), which merges all the result lists into a single ranked URL list that is as close to the "true global" result list as possible, i.e. as if the query had been executed by a centralized search engine over the same data set.

Given a query *q*, let q_m, a ranked URL list, denote the result returned by server-*m*, where in general *m* ranges from 1 to n_s, which is the number of servers in the system. q_m is a sublist of *m*'s ranked page list, $p_{l(m)}$, and it is empty if there are no relevant pages on server-*m*. In q_m, every URL is associated with its *Local PageRank* value. Let $\vec{\gamma}_{q(m)}$ be the corresponding *Local PageRank* vector of q_m, which is also a sub vector of $\vec{\gamma}_{l(m)}$, the *Local PageRank* vector of server-*m*. The result lists from every server can be simply weighted by the *ServerRank* vector and merged together,

$$\vec{\gamma}_{q_{\lg}} = \begin{bmatrix} \gamma_{s1}\vec{\gamma}_{q(1)} \\ \vdots \\ \gamma_{sm}\vec{\gamma}_{q(m)} \\ \vdots \\ \gamma_{sn_s}\vec{\gamma}_{q(n_s)} \end{bmatrix} \qquad (\textbf{\textit{RF}})$$

where γ_{sm} is *ServerRank* value of *m*. Then, the result page list, denoted by q_{lg}, is sorted according to the values in $\vec{\gamma}_{q_{\lg}}$.

If the entire *Local PageRank* vector of every server is applied in *RF*, the result, p_{lg}, will be the sorted list of all pages in the system. Obviously, q_{lg} is a sub list of p_{lg}, i.e., for any page *i* and *j* in q_{lg}, if *i* is ahead of *j* in q_{lg}, then *i* is also ahead of *j* in p_{lg}. Let q_g denote the corresponding result list obtained using the centralized *PageRank* algorithm, it is also a sub list of p_g, the complete sorted page list based on the true global *PageRank* vector $\vec{\gamma}_g$. Notice, that the number of mis-ordered pairs between p_{lg} and p_g is the upper bound of that between q_{lg} and q_g. Also, that p_{lg} and p_g are identical is a sufficient condition of q_{lg} being identical to q_g. Thus, the algorithm RF can be evaluated by comparing p_g and p_{lg}, or $\vec{\gamma}_g$ and $\vec{\gamma}_{\lg}$, the result of *RF* after normalization. The evaluation of actual queries will be shown in Section 4.

The set of *Local PageRank* vectors used in the following experiments were generated using algorithm *LPR-2*, refined by *LPR-Ref-2* using the *ServerRank* vector produced with *SR-2*. The L$_1$ distance between $\vec{\gamma}_{\lg}$ and $\vec{\gamma}_g$ is 0.0198. The Kendal's τ–distance between p_{lg} and p_g, is 0.00105, similar accuracy to the Local PageRank page lists. Figure 7 shows the minimizing Kendall's τ–distance between the top-*k* lists of p_{lg} and p_g, which illustrates a good match between two lists on the ranking of the most important pages.

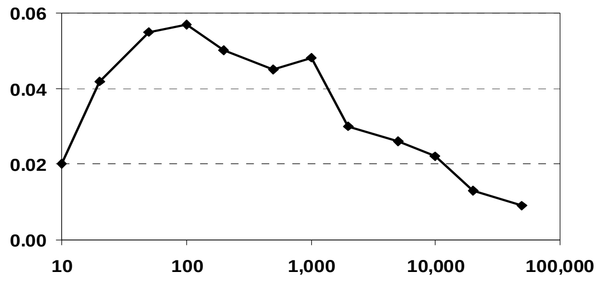

Figure 7: The minimizing Kendall's τ-distance for top-*k* page lists between p_{lg} and p_g (k = 10 to 50,000).

3.9 Query Routing and Data Updates

In this section, the query routing and data updates issues are briefly discussed.

In general, search engine queries are multi-term boolean text queries that request ranked results. On the one hand, more results are desired for purpose of completeness. On the other hand, the most important results need to be on the top of the result list. In the previous sections, this paper focuses on the ranking problem, assuming that queries are sent to and executed at all relevant web servers.

Recently there have been many studies on indexing techniques to help route queries in Peer-to-Peer systems. Several of these approaches could be applied to the search engine framework, such as Chord [25], one of a number of DHT-based indexing methods that have been proposed recently. Chord was designed to associate a key with each shared file in a P2P system, in which all keys are hashed into a common space and every participating peer is responsible for a portion of the key-space. DHT mechanisms like Chord can be adapted for use in a distributed search engine, where every keyword can be paired with a list of server ids that indicate which servers host pages containing that keyword. Thus, given a user's search query, the submitting server first retrieves the server lists of all query terms from the system. Second, it performs an intersection operation on the server lists to determine what servers host relevant pages. Then, it sends out the user's search query to the servers and waits for the results. This strategy guarantees the complete result if the query ends up being executed on every submitted server.

It may be expensive to send a query to all relevant web servers if the query contains some popular keywords that can be found on many websites. Furthermore, in most cases, people are more likely to be interested in the top-*k* results of a query, which requires only that the top-*k* lists are accurate. Intuitively, the query should not be sent to a server that cannot return pages whose PageRank value is not high enough to make the top-*k* list. Thus, a more efficient strategy can be used once the submitting server has the list of all relevant servers, denoted by S_q, as shown in Figure 8.

```
Set result list $p_q$ = ();
Sort $S_q$ by their ServerRank values, with the highest one on the top;
While ($S_q$ is not empty)
{
  Pop s servers out of $S_q$ and forward the search query q to them;
  Wait for results;
  Perform RF and merge the results into $p_q$.
  If ($p_q$ has at least k pages)
  {
    Find the PageRank value of the k-th page, $\gamma_{qk}$;
    Remove all servers in $S_q$ whose ServerRank value is lower than $\gamma_{qk}$;
  }
}
Return $p_q$.
```

Figure 8: The Query Routing Algorithm.

The submitting server can also stop forwarding the query when the user is satisfied with the top-*k* result list. Furthermore, it can attach the current threshold *PageRank* value to the query so that the receiving servers do not need to return less relevant results.

Since most search queries contain more than one term, whose corresponding indices are likely to be distributed into multiple servers using DHT-based approaches. *Direct peer indices* and *indirect peer indices*, proposed in the GALANX system [26], can be also applied in the framework, which tend to group indices of frequently co-occurring terms together so that fewer server contacts need to be made to discover the relevant server list,

To handle data updates, such as pages added/deleted on a server, or hyperlinks added/deleted on a page, every web server can periodically update their *Local PageRank* vector based on the frequency and extent of the changes. Updated inter-server link information can also be exchanged in the same fashion for the *ServerRank* computation.

4. Query Evaluation

In the previous section, the true global *PageRank* vector is used to evaluate the accuracy of the *Local PageRank* vectors in the different steps and the merged *PageRank* vector in the final fusion phase. In this section, queries are used to investigate the performance of the proposed algorithms.

As mentioned in Section 3.1, where Google's *PageRank* algorithm was reviewed, there are a few other factors in ranking search results in Google besides *PageRank*, including some standard IR measures, the appearance of query terms in page titles and anchor text, and text font size, etc. For each page, an IR score is computed for a given query and combined with its *PageRank* value to form its final page ranking value. Although the formula of the IR score is not publicly known, it is orthogonal to the *PageRank* computation. Thus, title search queries [22] can be used to further evaluate the algorithms presented in this paper. The results of a title search query are all web pages whose titles contain all the query words. So every word in the query can be treated equally and the result list is sorted by *PageRank* only.

In the over one million pages in the data set, there are about 150,000 unique terms in their <title> section. a set of 100 queries is selected, which include “Stanford University”, “engineering”, “research overview”, “news”, etc. On average, the complete result list of each query has about 3,450 pages from 11 web servers. The combination of algorithms *LPR-2*, *SR-2*, and *LPR-Ref-2* are used to construct and refine *Local PageRank* vectors on each server.

In the first experiment, the complete result list is returned for every query and it is compared against the result list produced using the true global *PageRank* vector. The average Kendal’s τ–distance between two result lists is 0.00047, which corresponds to approximately 2,800 mis-ordered pairs (out of a possible total of 6 million) between two 3,450-page lists. Figure 9 shows the accuracy of the top-*k* result lists. Notice that the important pages are in good order – The minimizing Kendall’s τ–distance between the top-10 list and its counterpart generated using the true global *PageRank* vector is 0.27, or only 1.2 mis-ordering pairs out of 45 possible pairs.

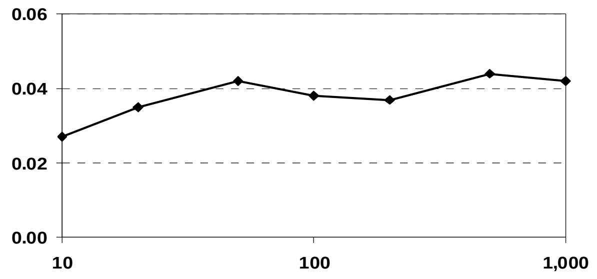

Figure 9: The minimizing Kendall’s τ-distance for top-*k* query result quality measures (k = 10 to 1,000).

If only the top-*k* result lists are wanted, the progressive routing approach described in Section 3.9 can be applied. The second experiment asks for top 100 page URLs for every query in the same query set. In each round, a query will be forwarded to 3 servers. As a result, on average, each query is executed on only 7 servers out of possible 11 servers, and only about 1,500 page URLs (of 3,450 potential results) are received.

5. Related Work

Among the existing Internet search engines, Google is the most frequently used, accounting for more than 35% of searches done by US web surfers[10]. Google employs the *PageRank* algorithm, designed by Page and Brin [3][22], to measure the importance of web pages. There are also a few other hyperlink connectivity based algorithms, such as the HITS algorithm by Kleinberg et al. [17][4] used by the IBM CLEVER Searching project [6].

Several other authors have considered the computation of *PageRank* vectors. Haveliwala [10] presents a block-based strategy for efficiently compute *PageRank* on computers with limited memory. Kamvar et al. [15] propose a power extrapolation algorithm to accelerate the convergence in computing *PageRank* vectors. Haveliwala [11] and Jeh et al. [13] discuss approaches to compute topic-sensitive or personalized *PageRank*. Bharat et al. [2] and Kamar et al. [14] observed the nested block structure of the Web. Kamar et al. [14] proposed *Local PageRank* and *BlockRank* algorithms (listed as algorithms *LPR-1* and *SR-2* in Section 3) to accelerate the computation of *PageRank* vectors.

While related, the main thrust of this work is orthogonal to these earlier efforts whose primary goal is to compute the *global PageRank* vector more efficiently. Specifically, the objective is to avoid computing the *global PageRank* vector altogether while still being able to provide quality ranking functions and quality results in a distributed search environment.

Web crawling is a critical part of Internet search engines. Cho et al. [3] defined ordering schemes to direct crawling, and evaluation metrics to measure their efficiency. Najork et al. [20] studied different crawling strategies and their impact on page quality. They found that crawling in a breadth-first search order tends to discover high-quality pages early on in the crawl, which was applied when the authors downloaded the experimental data set. Raghavan et al. [23] propose a layout-based information extraction technique to extract semantic information from web search forms and result pages in an effort to crawl the “hidden” web, the database-backed automatically generated web pages.

In early Peer-to-Peer file sharing systems, query routing is fairly simple. For instance, Gnutella [12] does not have any object indices. A query is simply relayed to all neighbor peers if it cannot be answered. In contrast to the flooding-based routing approaches, several research systems such as CAN [24], Chord [25], Pastry [8], and Tapestry [27], proposed independently, construct a distributed hash table (DHT) over the peer network in an effort to provide efficient query routing. In a DHT-based system, every shared file is associated with a key, either its name or a system id. All keys are hashed to a common key-space. Every peer is responsible for a portion of the key-space and stores the files whose keys fall into that key-space. Thus, every file request can be forwarded to the specific peer that corresponds to the file’s key.

6. Conclusions and Future Work

Existing Internet search engines use web crawlers to download data to their central servers to process queries. This paper describes and evaluates an alternative distributed approach in which every web server acts as an indi-

[10] Measured by the comScore Media Metrix gSearch service in November 2003, www.searchenginewatch.com.

vidual search engine on its own pages, eliminating the need for crawlers and centralized servers. In such a system, a query is taken by a web server of the user's choice, and then forwarded to related web servers. It is executed on those servers and results are returned to the submitting server where they are merged into a single ranked list.

Measuring the importance of web pages and ranking results are critical parts of a search engine. This paper focuses on how to apply ranking algorithms, more specifically Google's *PageRank* algorithm, in a distributed environment. The authors propose a series of *PageRank* variants, including *Local PageRank*, *ServerRank*, *Local PageRank Refinement*, and *Result Fusion*. A real web data set is used in the experiments, which shows a distributed approach can produce *PageRank* vectors that are comparable to the results of the centralized *PageRank* algorithm. Although it is premature to apply such distributed approach to the Internet scale which involves many other complicated research and engineering problems, the experiments, using a real-world domain of data, demonstrate it is promising to be adapted in an enterprise intranet environment.

Apart from improving the ranking algorithms, the authors plan to implement the framework in a real system in order to further investigate query routing issues and system performance such as query response time.

7. Acknowledgement

We would thank the anonymous reviewers for their valuable comments. The authors are supported by the NSF under grant number ITR-0086002.

8. References

[1] L. A. Barroso, J. Dean, U. Hölzle. "Web Search for a Planet: The Google Cluster Architecture", *IEEE Micro*, 23(2): 22-28, March/April, 2003.

[2] K. Bharat, B.-W. Chang, M. R. Henzinger, M. Ruhl. "Who Links to Whom: Mining linkage between Web Sites", in *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM'01)*, 2001.

[3] S. Brin, L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", in *Proceedings of the 7th International World Wide Web Conference (WWW7)*, 1998.

[4] S. Chakrabarti, B. Dom, D. Gibson, S.R. Kumar, P. Raghavan, S. Rajagopalan and A. Tomkins. "Spectral Filtering for Resource Discovery", *ACM SIGIR workshop on Hypertext Information Retrieval on the Web*, 1998.

[5] J. Cho, H. Garcia-Molina, L. Page. "Efficient Crawling Through URL ordering", in *Proceedings of the 7th International World Wide Web Conference (WWW7)*, 1998.

[6] The IBM CLEVER Searching project. Available at http://www.almaden.ibm.com/cs/k53/clever.html.

[7] P. Diaconis. "Group Representation in Probability and Statistics", Number 11 in IMS Lecture Series. Institute of Mathematical Statistics, 1998.

[8] P. Druschel, A. Rowstron. "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems", in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[9] R. Fagin, R. Kumar, D. Sivakumar, "Comparing top k lists", SIAM J. Discrete Mathematics 17, 1 (2003), pp. 134 – 160.

[10] T. H. Haveliwala. "Efficient Computation of PageRank", *Stanford University Technical Report*, 1999.

[11] T. H. Haveliwala. "Topic-Sensitive PageRank", in *Proceedings of the 11th International World Wide Web Conference (WWW11)*, 2002.

[12] The Gnutella website, http://www.gnutella.com.

[13] G. Jeh, J. Widom. "Scaling Personalized Web Search", in *Proceedings of the 12th International World Wide Web Conference (WWW12)*, 2003.

[14] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, G. H. Golub. "Exploiting the Block Structure of the Web for Computing PageRank", *Stanford University Technical Report*, 2003.

[15] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, G. H. Golub. "Extrapolation Methods for Accelerating PageRank Computations", in *Proceedings of the 12th International World Wide Web Conference (WWW12)*, 2003.

[16] M. G. Kendall, J. D. Gibbons. "Rank Correlation Methods", *Edward Arnold*, London, 1990.

[17] J. Kleinberg. "Authoritative Sources in a Hyperlinked Environment", in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[18] M. Koster, "A Standard for Robot Exclusion", available at http://www.robotstxt.org/wc/norobots.html.

[19] P. Lyman, H. R. Varian, K. Swearingen, P. Charles, N. Good, L.L. Jordan, J. Pal, "How Much Information 2003?", School of Information Management and Systems, University of California at Berkeley, 2003. Available at *http://www.sims.berkeley.edu/how-much-info-2003*.

[20] M. Najork, J. L. Wiener. "Breath-First Search Crawling Yields High-Quality Pages", in *Proceedings of the 10th International World Wide Web Conference (WWW10)*, 2001.

[21] Netcraft Ltd. "Web Server Survey", Available at *http://news.netcraft.com/archives/web_server_survey.html*.

[22] L. Page, S. Brin, R. Motwani, T. Winograd. "The PageRank Citation Ranking: Bringing Order to the Web", *Stanford Digital Libraries Working Paper*, 1998.

[23] S. Raghavan, H. Garcia-Molina. "Crawling the Hidden Web", in *Proceedings of the 27th International Conference on Very Large Dta Bases (VLDB'01)*, 2001.

[24] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, S. Shenker. "A Scalable Content-Addressable Network", in *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 2001.

[25] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", in *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01)*, 2001.

[26] Y. Wang, L. Galanis, D. J. DeWitt. "Galanx: An Efficient Peer-to-Peer Search Engine System", Available at http://www.cs.wisc.edu/~yuanwang.

[27] B. Y. Zhao, J. D. Kubiatowicz, A. D. Joseph. "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing", *UC Berkeley Computer Science Division Report No. UCB/CSD 01/1141*, 2001.