SiteRank: Link-Based Relevance Computation for Persistent Search

Erich R. Schmidt and Jaswinder Pal Singh
Department of Computer Science
Princeton University
Princeton, NJ 08544
eschmidt@cs.princeton.edu

Report number: TR-745-06
February 2006

Abstract

Existing search services rely heavily on citation-based authority (e.g. PageRank) to assess the quality of publications. The quality and relevance of results is particularly important in persistent search, but the current rank computations are strongly biased against new pages. We propose SiteRank, a new ranking mechanism that handles new publications well and also dramatically reduces the communication and computation costs.

This performance improvement is especially valuable when authority is computed in a persistent search service. Current systems, whether small-scale notifiers (e.g. CNN Alerts) or persistent queries on traditional search engines (e.g. Google Alerts), suffer from limited coverage and/or low refresh rates. We propose Distributed Persistent Search, a new architecture based on a publish-subscribe framework that achieves linear improvement in publication processing and notification routing, as a function of the number of servers used.

1 Introduction

Keeping track of new information on the web can be a difficult task using existing services. One can repeatedly query search engines, or use their newly deployed persistent query feature [3, 4]. There are also similar services being deployed directly by information providers [2, 11, 12] or aggregators [7, 13, 22, 26]. While dedicated services work well for specialized content (e.g. news), there is still need for good, scalable solutions for Internet-scale general purpose persistent search. The same way that general purpose web search engines were needed in addition to lower scale specialized search systems, there are numerous examples that illustrate the need for a general purpose persistent search system: companies can keep track of competitors through news, blogs,

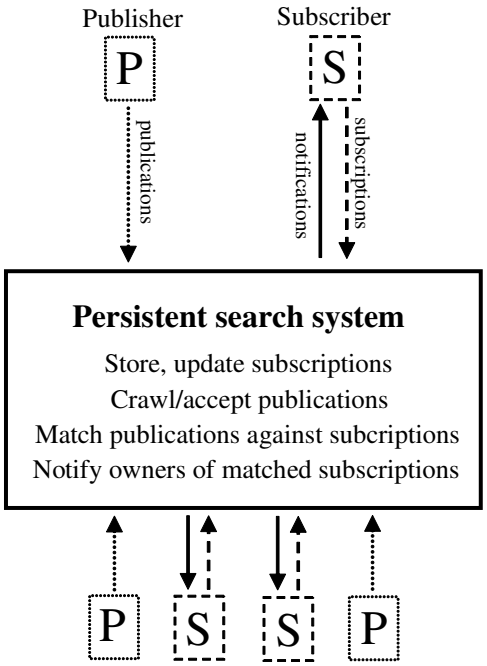


Figure 1: Persistent search system

and more obscure publications; researchers can discover new publications, grants, announcements on topics of interest etc.

Figure 1 shows a high-level view of a general persistent search system. The underlying architecture is that of a publish-subscribe service (pubsub). The main elements are publishers, subscribers and the persistent search system. Publishers provide publications (text-based files in our case: txt, html, rss, xml), either by actively pushing/advertising them (via RSS or other protocols) or by allowing the system to crawl them periodically. Subscribers submit queries and expect notifications of matching publications. The persistent search system accepts/crawls the publications, stores and updates the subscriptions, matches every incoming publication against all subscriptions and notifies the owners of matched subscriptions. A notification consists of a reference to the publication together with perhaps a summary and some metadata; the actual document is not sent to the user, but can be accessed at the publisher.

Existing persistent search systems are limited in scale. Data throughput is always limited in a single-location architecture (single-server and parallel/cluster solutions). While keyword-based subscriptions are small and relatively static, the size and growth rate of the web present a bigger challenge (publications come in various sizes and are very dynamic).

The goal of our work is to develop a persistent search service that can effectively cover a large and highly distributed publication base, with a high degree of freshness in covered publications. To increase publication throughput, we distribute the underlying

pubsub system geographically, effectively dividing the publication base by the number of servers and significantly reducing the publication latency. We propose mechanisms to address some of the key challenges imposed by distribution. We use simulation to evaluate the distributed design and compare it against two existing approaches: single-server persistent search and distributed persistent search filtered through a single server. The cost of this distribution is an increase in subscription management communication, as measured in Section 4.2.

Since it actively notifies users, a persistent search system arguably has an even higher pressure to provide only high quality and highly relevant results than a traditional search engine. Along with content-based relevance, traditional search engines have successfully relied on citation-based authority for quality of results; PageRank [8, 23] is a good example of using the web's link structure to assign importance to well-cited pages. This rank computation is done periodically in a centralized manner.

The main problem with PageRank for persistent search systems is that it ascribes very low authority to new pages - since they will by definition have few inlinks - while a key focus of persistent search systems is to provide new matching content quickly. Previous studies [5, 10] show that PageRank is biased against new pages; popular pages become increasingly popular due to their high link rank, and new pages are often ignored regardless of their quality.

For distributed persistent search, another problem is introduced by the PageRank computation's requirement that the entire link structure be known at a single location, even if document processing is distributed.

We propose SiteRank, a method that assigns authority to pages based in large part on computed authorities of the sites that publish them, together with characteristics of the pages within the sites. SiteRank not only handles new pages well, but also has much decreased computational cost and communication overhead compared with PageRank. Our proposed mechanism uses a single dedicated server for rank computation, but the communication overhead required by this approach is just a very small percentage of overall publication management costs.

The rest of this paper is structured as follows: Section 2 presents existing work on link-based authority computation, persistent search and distributed publish-subscribe systems; Section 3 describes the design of our SiteRank method, along with the experiments and evaluation; Section 4 presents the design and evaluation of the Distributed Persistent Search system; Section 5 concludes the paper and discusses our plans for future work.

2 Related work

Our goal is to build a highly scalable general-purpose persistent search system, based on a distributed publish-subscribe architecture. One particularly important component of this system is its ability to rank documents using citation-based authority. We therefore first look at some of the more popular approaches currently in use, and establish a number of desirable properties for our system.

2.1 Citation-based authority

There is a lot of research addressing link-based authority. The most popular and most frequently used method is Google's PageRank [8, 23]. We mention here a number of approaches to the PageRank computation in single- or multi-server architectures.

Far from being random, the web exhibits a nested block structure, where pages on the same site preferentially link to one another; Kamvar et al. [19] take advantage of this structure and propose a 3-stage algorithm that speeds up the PageRank computation by a small constant factor. Kamvar et al. [20] also introduce a power extrapolation method to accelerate the convergence in computing PageRank. Haveliwala [17] uses a block-based strategy to compute PageRank efficiently and proposes topic-sensitive PageRank vectors [18].

A fully distributed implementation of PageRank for peer-to-peer systems is proposed by Sankaralingam et al. [29]. Their algorithm converges rapidly, requiring a smaller number of iterations compared to the classic single-server algorithm. However, high communication costs required during each of the iterations result in reduced overall performance. Wang and DeWitt [34] propose a distributed solution requiring much less communication.

All these approaches, while addressing different issues and improving on the original PageRank algorithm in various ways, do not handle new pages well. Each page's rank is directly determined by its links alone. Recently, ranking of sites has also been taken into consideration. Google just announced that it plans to rank pages by the quality and reliability of the publishing site (source), for its Google News service [14]. Wu and Aberer [35] use the site graph to compute site ranks and observe that page ranks (as given by PageRank [23]) and site ranks follow very similar distributions; however, these site rank values are not used to assign individual page authority values.

2.2 Persistent search

Highly specialized services (such as financial data [22] or news [2, 11]) use active and passive publishers to cover a relatively small and highly specific publication base. Real-time coverage is therefore possible, and clients are informed of new or updated data quickly. Due to the limited publication base, a single-server or single-location (cluster) architecture (*1S*) can be used successfully; all operations are performed at the same location (publication and subscription handling, matching, notification) - this approach is illustrated in Figure 2.

The same architecture is used by persistent search services deployed by traditional search engines [3,4]. While a single-location system can easily handle documents from a small number of publishers, this architecture does not scale well when the number of publishers and subscribers grows significantly. Earlier estimates on the size of the web [30] and search engine coverage [31] show that search engines cannott easily scale up with the growing web. These numbers do not even take into consideration the deep web - information used to create dynamic pages on demand, until recently completely hidden from search engines. This data would expand the size of the web by an estimated factor of 500 [6], considerably widening the gap between the available publication base and the search engines' coverage. Existing search engines do not fare

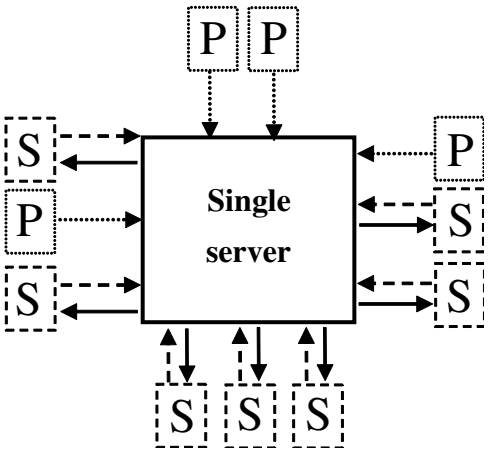


Figure 2: Single-server architecture (1S)

well when it comes to data refresh rate, either. On average, the bulk of most search engine databases is about one month old [31]. Few pages are indexed more frequently (1-2 days, hours), but there are also a large number of pages that have not been re-indexed for a long time (50+ days).

Some simple calculations based on estimates can show us the magnitude of the problem. Recent estimates put the size of the indexable web over 11.5 billion pages [16], a very conservative lower limit. In 2001, the size of the surface web was estimated at 4 billion pages, and the growth rate of the surface web at 7.3 million new pages per day [30]. Assuming that the growth rate did not decrease and the same number of pages are modified each day, the new or modified content adds up to about 15 million pages per day. It is estimated that the deep web grows faster than the surface web [6], so multiplying by 500 we estimate that there are at least 7.5 billion new or modified pages per day. Using a conservative estimate of 100 KB for the average page size, we need over 70 Gbps at one location to manage all this new content. However, we do not know in advance which pages are modified or which sites publish new pages, so we need 1000 times this bandwidth if we want to crawl the entire web once every day. While this is already a huge number, there is no reason to assume the web growth is slowing down, so the bandwidth requirements will keep growing as well.

Both coverage and refresh rate can be improved through parallelization or distribution of crawling. As seen in existing search engines, parallel crawling is still limited by bandwidth. Some work is being currently done on distributed search engines built around central control [15]. This architecture (*DCC*) is illustrated in Figure 3 - a central server assigns crawling to other servers, which then send the content back to central control. This approach effectively distributes the crawling load and increases the crawling-based refresh rate, but it fails to eliminate the publication traffic to a central server. Also, all subscription handling, matching and notification is done at the central server. Dealing with the large volume and freshness requirements of a web-scale persistent search engine requires a more fully distributed architecture.

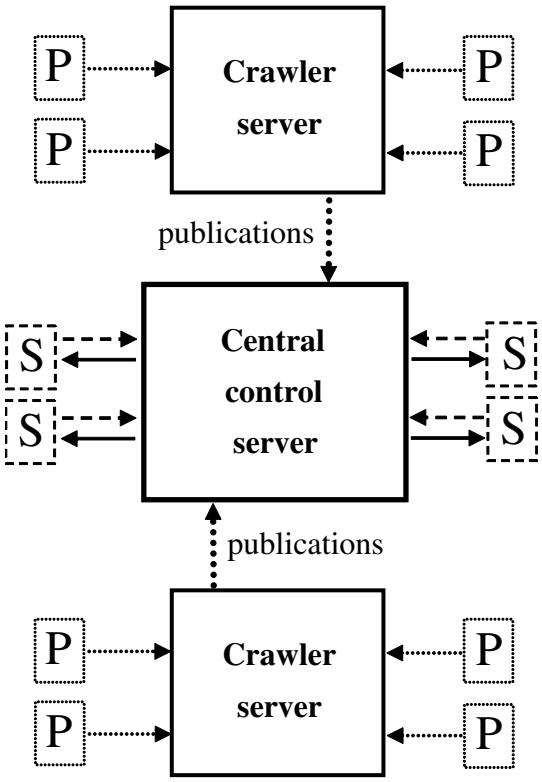


Figure 3: Distributed architecture with central control server (DCC)

Traditional search engines are deployed on carefully monitored and stable architectures. While peer-to-peer systems are usually less reliable, there are some persistent search systems proposed on top of structured peer-to-peer architectures. Unstructured peer-to-peer architectures are not usable due to their lack of guarantees and poor scalability [21]. Structured peer-to-peer systems based on distributed hash tables (DHT) scale better and can offer search guarantees. Workload and performance estimates [21] suggest that very large systems built on these architectures might be used for distributed search, but dynamic peer sets could easily make the overlay network too expensive to maintain.

Tang and Xu [33] propose pFilter, a global-scale persistent search system that uses a DHT to store queries and efficiently route publications to related queries. Due to the relatively large size and publication rate or pages on the web, this inter-server publication traffic can prove expensive. The use of latent semantic indexing (LSI) offers an efficient matching mechanism that eliminates the common problems of literal matching schemes. Subscriptions are aggregated for more efficient routing, building a single multicast tree for similar queries, but this might lead to unnecessary notification messages being sent. The lack of any additional document quality assessment can further increase non-relevant notification traffic.

2.3 Distributed publish-subscribe systems

Distributed publish-subscribe architectures have been proposed for more general applications other than text-based persistent search; in particular, for publish-subscribe on structured name-value pair messages. The three major approaches are content-based forwarding, best illustrated by SIENA [1, 32]; channelization [27]; and dynamic multicast [9].

In content-based forwarding [1, 32], the network of servers is organized in an acyclic overlay. Subscriptions are broadcast from each server on the tree. Every server builds a forwarding table for each of its neighbors by summing up all the subscriptions from that neighbor's direction. Publications are matched at each server against all the forwarding tables stored at that server, and forwarded to neighboring servers based on matches. This approach requires repeated matching of each publication introducing unnecessary publication traffic through nodes that are not interested in those publications.

Channelization [27] partitions the publication space into a fixed, limited number of channels, replicating this partitioning on all servers. A multicast tree is built for each channel, connecting all the servers containing subscriptions that might be matched by publications from that channel/partition. The advantage is that publications are quickly forwarded in the corresponding channel based on simple lookups of the publication space partitioning. However, since the possible number of keyword-based subscriptions is much larger than any practically possible number of channels, nodes will belong to many channels if even one of their subscriptions overlaps with the channels' event spaces, so a large number a non-matching publications will reach each server.

A major drawback of both options above is the high communication cost required by publication forwarding [9]. Given the high publication rate on the web, and the relatively large size (compared to subscriptions) of an average publication, a good approach would minimize publication traffic through the system. In MEDYM [9], publications are matched against subscriptions at the first pubsub server they reach. Any notifications are then multicast to the servers containing all the matched subscriptions, using dynamically constructed multicast trees for those servers [9]. The disadvantage is that subscriptions must be broadcast to all the servers. The MEDYM architecture is very well suited to our problem because publications are much larger and more dynamic than subscriptions.

Building on a distributed publish-subscribe framework introduces a major problem by removing the single publication repository that can be used for quick relevance computations. The next section presents our solution to this problem, while further details of our persistent search system are illustrated in Section 4.

3 Citation-based authority

The problem we set out to solve initially is building a scalable persistent search system. Link-based authority is a very important component of such a system, and we chose to address it first individually. Section 4 presents our design and evaluation of our Distributed Persistent Search system, where SiteRank is used as the link-based ranking

component.

In addition to content-based relevance to queries, an important component of the quality of a page is its rank (citation-based authority), computed from the page's incoming links and the rank of the pages these links originate from. Link-based ranks are widely used by current search engines. However, for this mechanism to be useful in a persistent search system, we must address a number of challenges:

- As mentioned earlier, new pages do not have many incoming links, and therefore they are assigned low rank values. Drawing on Wu and Aberer's [35] observations that the ranks of pages and the ranks of sites hosting them follow the similar distributions, we derive the ranks of individual pages from their site's rank.
- Search engines compute PageRank periodically, after refreshing their index; this rank computation is also computationally intensive. Fresh pages can arrive much more frequently in a scalable persistent search system, so we need a faster solution. Computing site ranks is much faster than computing page ranks due to the much reduced link graph.
- The need for a central repository of all link information makes it hard to adapt PageRank in a distributed system. We use a dedicated server for rank computation, where we send only the necessary information for link graph maitenance and rank computation.

Our solution to the above listed challenges is the SiteRank algorithm, described in the next section. Experimental evaluation is then presented in Section 3.2.

3.1 SiteRank overview

We propose a modification of the classic PageRank algorithm [23] that handles new publications well, even in the absence of inlinks; this algorithm also significantly reduces the required computation time.

3.1.1 Sites

Fresh pages have usually a very small number of incoming links. Their citation-based authority score is therefore low, making it difficult for these pages to be noticed by users (they fall to the bottom of the result list in traditional search, and may be excluded from notifications in persistent search). It was observed however that there is a correlation between the ranks of pages and the ranks of the sites hosting them. We take advantage of this by deriving the pages' rank values from the rank values of the sites, using additional information specific to each individual page.

The first step of SiteRank is to compute authority scores for the sites. To do this, we reduce the page-link graph to a site-link graph by collapsing all pages from the same site into a single node; links between these site-nodes are then weighted based on the number of inter-page links between sites. Similar to PageRank, this new graph's principal eigenvector is computed in an iterative calculation; these values constitute the site authority scores.

As an added bonus, this significant reduction in the size of the web graph results in several orders of magnitude of speedup compared to the basic PageRank algorithm; each iteration is much shorter on a smaller graph, and the overall computation converges faster. This allows rank scores to be recomputed more often, a welcome option in a system processing a high volume of new publications.

3.1.2 Pages

The second part of SiteRank consists of computing the actual page rank values. For a page p_j on site s_i, we use the notations: *r(p_j)* and *r(s_i)* for the rank of the page and the site, respectively; *n(s_i)* for the number of pages on the site; *ol(p_j)* and *ol(s_i)* for the number of outlinks from the page and site, respectively.

At this stage, we take into account multiple factors:

- The rank of the site containing this page - this is the starting value. All pages from the same site are initially assigned a rank value equal to the site's authority value divided by the number of pages on the site.

  This first step is summarized as *r(p_j) := r(s_i) / n(s_i)*, for all pages p_j on site s_i, for every site s_i.

- The page's position on the site, given by the directory structure - pages further away from the root have lower rank values compared to pages closer to the root. While not necessarily always true, we will see that this is a good indicator when only a small number of links are available.

  For each site, we redistributed the total rank (the rank of the site, equal to the sum of the ranks of all the pages on the site) based on the site's directory structure. A constant reduction factor *f* is applied between each consecutive levels, i.e. a page at level *k* is assigned a rank value *f* times larger than a page at level *k+1*. Our experiments show that a value of 1.2 for *f* gives best results.

  While this approach works quite well for vertically structured websites with a deep directory structure, it is not so helpful for sites exhibiting a flat structure, or dynamic pages.

- By storing the entire page-link graph (and not only the site-link graph used for the SiteRank computation), we can adjust individual page ranks based on incoming links. Since we apply this step after adjusting page ranks using the directory structure, we can take advantage of the fact that page rank values are different from site rank values and ajust link contributions between pages accordingly.

  We define the link contribution of a site (or page) as the rank value of the site (or page) divided by the number of outlinks from that site (or page): *lc(s_i) := r(s_i) / ol(s_i)* and *lc(p_j) := r(p_j) / ol(p_j)*.

  The rank adjustment is computed as follows: for page p_1 on site s_1 with an inlink from page p_2 on site s_2, we first determine the fraction of the site-rank *r(s_1)* coming from site s_2 as the link contribution of site s_2 divided by the sum of all link contributions to site s_1. Multiplying this fraction by the current rank

of page p_1 gives us the contribution of page p_2 to the rank of page p_1. Finally, this inter-page contribution is scaled by the ratio between page p_2's contribution and site s_2's contribution, i.e. by *lc(p_2) / lc(s_2)*. The final rank of page p_1 is computed as the sum of the scaled inter-page contributions over all its inlinks.

While it would be possible to use PageRank inside each site to compute individual pages' ranks, our proposed heuristics handle new pages much better, especially in the absence of inlinks. Using PageRank would also significantly increase the computation cost of this step; while only an approximation, the proposed computation allows immediate link authority computation for a page.

We derive SiteRank from a basic version of PageRank [23] with no optimizations, using all inlinks for every page (including intra-site links). During evaluation, the same basic iterative algorithm is applied to both the page graph and the site graph.

3.2 SiteRank evaluation

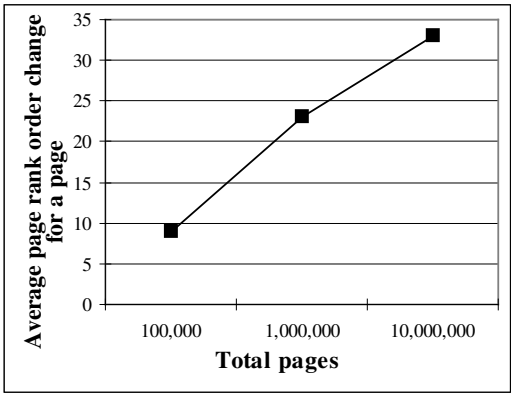
We measured the performance and accuracy of the SiteRank-based pagerank computation, compared with the classic PageRank algorithm (further experiments evaluating SiteRank as a component of a distributed persistent search system are summarized in Section 4.2). The publication dataset used for these experiments is a webcrawl from the Stanford WebBase project [25], containing 11 million text documents (html pages). Random subsets of 100,000, 1 million and 10 million pages were used. The experiments were repeated 1000 times for each subset size, and results averaged; however, different runs for the same subset size show very similar results, with no major variations.

In order to test the accuracy of this new approach, we compared the rank order of page authority values as given by the SiteRank-based computation versus the classic PageRank computation. For every page, the rank order change is defined as the absolute difference of the page's rank order number according to the classic PageRank algorithm and that according to the SiteRank method. We first computed the average rank order change over all pages in the dataset. Figure 4 (a) shows that there is very little loss of accuracy as measured by this experiment; it shows a logarithmic increase in the average rank order change when the dataset grows from 100,000 to 10 million pages.

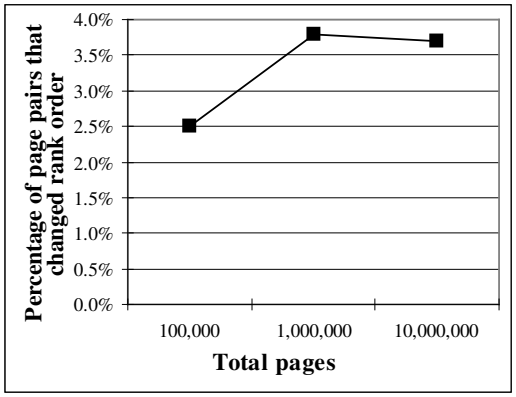
Varying the total number of pages from 100,000 to 1 million and 10 million, we randomly selected 1 million page pairs and checked their relative rank order as given by PageRank and SiteRank; Figure 4 (b) summarizes the results. The accuracy of SiteRank is even more visible now - less than 4% of the pairs switched order, without any increase when going from 1 million to 10 million pages.

To test the individual parts of SiteRank (site rank computation and page rank computation), we looked separately at:

- pairs of pages from the same site - their relative rank order is determined by page position and possible inlinks
- pairs of index pages (e.g. *index.html* at a site's root level) - their relative rank order is mostly determined by their sites' rank values.



(a)



(b)

Figure 4: SiteRank vs. PageRank accuracy: (a) average order change; (b) page pairs that flipped rank order.

A detailed analysis of these results is summarized in Table 1. When only same-site pairs of pages are considered, 8-10% changed rank order. Among pairs of index pages, only about 3.5% flipped their rank order from PageRank to SiteRank. This suggests that SiteRank preserves much of the classic PageRank algorithm's ordering of the sites, while our heuristics for individual page rank computation starting from the site's rank value introduce most of the rank ordering changes.

Of particular interest are the pages situated at the top of the ordered list. We want to make sure that the most popular pages stay popular when ranks are computed using SiteRank, so we looked at the order change for the top k pages out of a set of 10 million. The results are summarized in Table 2. Since most top pages are also index pages for sites with a very large number of incoming links, the top pages are more likely than an average page to maintain their order relativ to each other and the rest of the pages. The individual rank order changes for top pages are much smaller than for an average

Pages	Fraction of page pairs that changed rank order		
	random	on the same site	index pages
100,000	2.5%	8.0%	3.4%
1,000,000	3.8%	8.9%	3.2%
10,000,000	3.7%	9.7%	3.6%

Table 1: Changes in rank order of random page pairs.

k	Average page rank change	Fraction of page pairs that changed rank order
10	1	1.0%
100	3	1.3%
1,000	7	1.5%
10,000	10	1.6%

Table 2: Changes in rank order for top-*k* pages when SiteRank is computed over 10 million pages.

page, and the page pairs are much less likely to switch order (1.6% for top-100 pages compared to 3.7% for all pages).

The tests presented so far (and the speedup measurements at the end of this section) were performed using all the available links for all pages. However, the original motivation for SiteRank was to correct the bias of PageRank against fresh pages with small number of inlinks. In order to simulate fresh pages gaining citations over time, we removed a part of the links then restored them incrementally, using SiteRank to recompute page ranks. On a set of 10 million pages, we first computed PageRank using all links; we then randomly removed 75% of the links and computed SiteRank on the publication set using the remaining 25% of the links; we recomputed SiteRank using 50%, 75% and 100% of the links. When using partial link information (less than 100% of the links), multiple runs of the experiment were averaged for the final results; different runs showed very similar results, with little variation.

The results of this experiment are summarized in Table 3. While rank order changes quite frequently when only 25% of the links are used (almost 13% of the page pairs switched order), gradually adding links significantly decreases this number. However, even in the absence of many links a vast majority of page pairs preserved their relative rank order, validating this approach as a useful mechanism to rank fresh pages.

To determine speedup against PageRank, we ran both algorithms on the same publication sets. When randomly choosing pages from the webcrawl, the resulting subsets have different average site sizes (number of pages). We notice that the computational speedup of SiteRank relative to PageRank is mostly determined by the average site size (Figure 5), and ranges between 800 and 2100 for these cases.

This is seen even better when only the average site size is changed while keeping

Percentage of links used	Fraction of random page pairs that changed rank order
25	12.7%
50	8.5%
75	4.2%
100	3.7%

Table 3: Rank order changes using partial link information to compute SiteRank over 10 million pages.

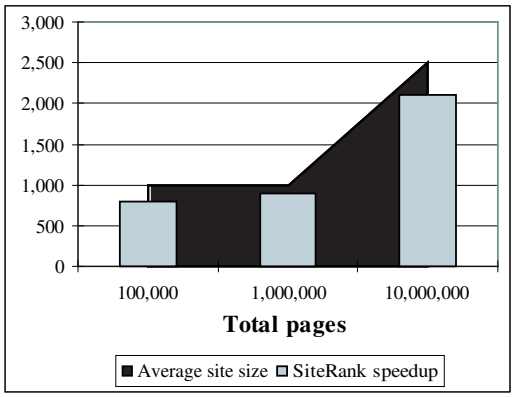


Figure 5: SiteRank vs. PageRank speedup (foreground columns) follows average site size (background shape).

the publication set size constant at 1 million pages. We achieve linear speedup when the average site size varies from 1,000 to 2,000 and 2,500 pages/site.

While the main motivation for SiteRank is the management of fresh pages in persistent search, the results show that this algorithm might be a much lower cost and yet accurate replacement for PageRank in general for regular search.

4 Distributed Persistent Search

Citation-based authority computation is only one component on a persistent search system. Building a large-scale distributed persistent search system raises a number of additional challenges that need to be addressed. Some of them are listed below:

- *Distribution:* As mentioned above, we aim to minimize publication traffic. A key challenge in this case is the positioning of servers close to high-volume publishers, and the non-overlapping partitioning of the publication space among servers. We have not yet addressed this issue in our system and we are using simulation for testing.

- *Matching:* Depending on subscription and publication distribution, matching can be done once or many times, against all of the subscriptions or only part of them. We are currently using a classic algorithm matching each incoming publication against all subscriptions, and will enhance this component if required by more expressive subscriptions.

- *Notifications:* This component depends heavily on the distribution scheme used, the main goal being to reduce internal traffic necessary to transmit notifications for each matched publication. Possible approaches include publication routing via successive matching, building static or dynamic multicast trees. We are currently using MEDYM [9], which offers efficient notifications with low cost maintenance.

- *Citation-based authority:* This component was discussed in Section 3.

- *Content-based quality assessment:* The lack of a global, centralized document repository presents a challenge for publication content analysis. We currently use local information on each pubsub server, but previous work on latent semantic indexing over peer-to-peer architectures can be applied here and we will address this issue in the future.

- *Duplicate detection:* Since distribution divides the publication management load between multiple nodes, storage requirements per node are relatively small; this allows us to detect duplicate documents managed by the same server. However, the absence of a single global repository makes it difficult to find duplicates managed by different servers. We propose a simple mechanism that uses publication metadata to detect exact duplicates before user notification, but further work is required to detect near duplicates across the system.

In our current system, we addressed these challenges in various degrees. The following section presents some details of our system design.

4.1 DPS overview

4.1.1 Server distribution

We designed and built the Distributed Persistent Search (DPS) system. A high-level view is illustrated in Figure 6.

The main goal of the distributed architecture is to minimize publication latency by eliminating intra-system publication traffic and shortening publisher-to-server paths. We therefore propose to distribute servers geographically based on publisher density, and assign publishers to servers based on publication latency, i.e. each publisher/site is assigned to the closest server. Servers can acquire documents from publishers in two ways: active publication, where publishers actively push documents to the servers; and passive publication, where servers crawl nearby sites for publications. Since we have not yet deployed the system but use simulation to evaluate it, no details of this component are discussed in this paper.

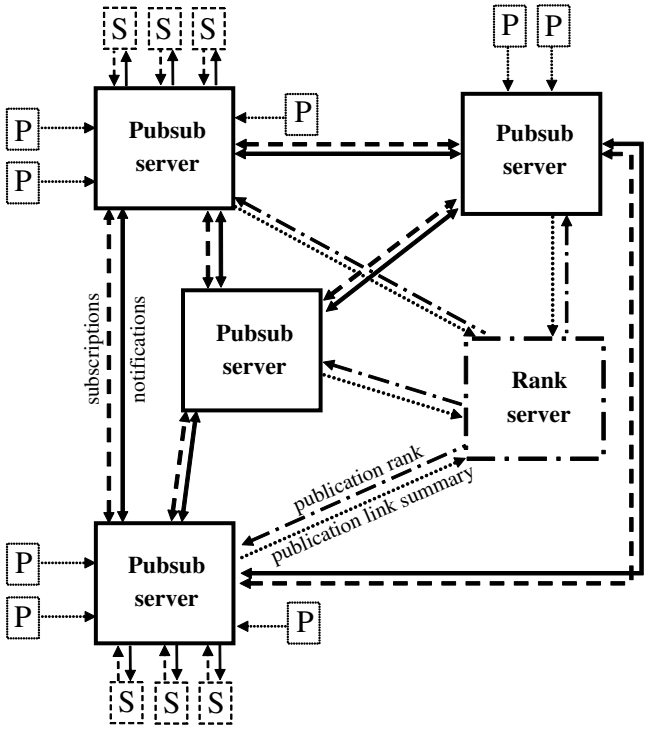


Figure 6: Distributed Persistent Search

4.1.2 Subscription management

A subscription is a vector of keywords, stored at the server in an inverted map from keywords to subscription IDs. Subscribers connect to the closest server and all their subscriptions are first stored at this location; these subscriptions stored at the server closest to their owner are called *local subscriptions*. Since publications are not forwarded through the system but matched at their entry server, each new or updated subscription is also broadcast to all other nodes in the system; these *remote subscriptions* are stored with references to the originating server ID, which has to be notified if these subscriptions are matched. Additional information (user ID, notification preferences) are stored only locally at the originating server.

4.1.3 Publication management

Text-based publications are processed at the incoming server and matched against all subscriptions. Publication link summaries (list of outgoing links), typically much smaller than the actual publications, are sent to the siterank computing server for web graph maintenance; the reply from this server consists of the rank value of the new (or updated) page.

Subscriptions and publications are managed by the same servers; each server han-

dles both subscriptions and publications; we will refer to them as *pubsub servers*.

4.1.4 Publication relevance and quality assessment

Our approach to compute content-based authority was discussed in section 3.1. A dedicated server (*rank server*) is used to collect the link information. Link summaries are sent to this dedicated server from the document processing nodes whenever a new page is acquired, or an existing page is updated and the page's links are modified. The rank server uses the current siterank and other information (see 3.1.2) to compute an authority value for the page; this value is sent back to the pubsub server handling this publication. When a large enough portion of the pages are modified, siterank values are recomputed asynchronously in the background by the rank server. We have also explored a distributed implementation of SiteRank and will publish our results.

The overall relevance score of a publication also takes into consideration content-based relevance, currently the TF-IDF score [28] between the publication and each matched subscription (inverse document frequency is computed from the local information at each pubsub server). Judging content-based relevance is an interesting topic that is beyond the scope of this paper.

4.1.5 Matching and notification

Matching of each publication against all subscriptions is done at the publication's entry point. A successful matching generates a list of local matched subscriptions, and a list of remote subscriptions whose originating servers must be notified. For each local matched subscription, a notification (currently email) is sent directly to the owner of that subscription. All matched remote subscriptions' servers are notified using a single dynamic multicast message [9] containing the publication's metadata and relevance score. Each owner of a matched subscription is then notified by the local server.

In MEDYM [9], individual user subscriptions are aggregated before propagation and retained individually at the original servers for local matching before sending notifications to individual subscriber. This approach requires two matching operations: first at the document entry server, where a list of servers with potentially matched subscriptions is determined; at each of these servers, the document is matched against local subscriptions. While subscription replication is less expensive due to aggregates, the same mechanism generates possible false duplicates that require unnecessary inter-server notification messages. DPS uses MEDYM's dynamic notification, but subscriptions are propagated individually to all servers, so only one match operation is required at the entry server. Aggregates and other optimizations can be added to this mechanism, and we will study their benefits and costs in future work.

4.1.6 Duplicate detection

DPS detects exact duplicates at document entry points and at the notification servers.

Publications must be stored for some time locally at the entry server for local duplicate detection - this is not a huge burden, since the publication space is partitioned

among the servers. This solution detects duplicate publications that are handled by the same pubsub server.

A matched publication's metadata is used both in the user notification messages and to detect distributed duplicates (often due to site mirroring). Storing publication metadata at servers that hold recently matched subscriptions might catch some of these duplicates: each notification is cached at the server storing the matched subscription, together with the matching publication's metadata; subsequent notifications are checked against this cache for possible duplicates.

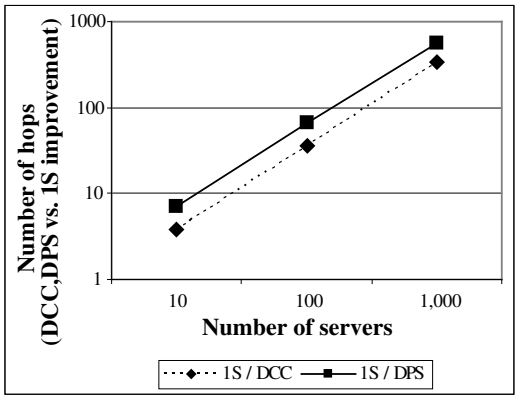
4.2 DPS evaluation

To evaluate this design, we implemented an event-driven, message-level simulator, on a random IP network topology with 100,000 nodes; pubsub and ranking servers were assigned to a subset of these nodes. This simulator contains fully operational components (subscription storage and replication; matching; relevance computation; notification routing), with the exception of live content publication and user interface. The 1S (single server) architecture uses a single server chosen randomly from the top 1% nodes with most links. For each of DCC (distributed architecture with central control) and DPS (distributed persistent search), server nodes were randomly chosen from the nodes in the simulated network; edges were then added so that the servers' subnetwork was fully connected. DCC uses k crawler servers and an additional central control server; DPS uses k pubsub servers and an additional rank server. Limited by memory and computation costs, the system sizes (k) used in these experiments were 10, 100 and 1,000 nodes. Multiple runs of the experiments were averaged, each run using a different assignment of servers to nodes; there were no major variations between different runs. We present the performance/cost values for DPS and DCC normalized by the values for 1S.

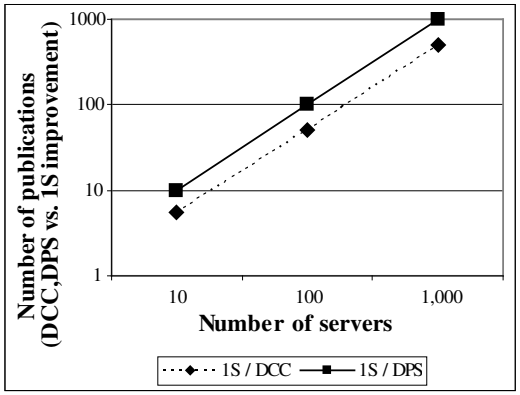
To test the overall system performance, we compared our DPS system with 1S and DCC, measuring publication, notification and subscription management costs. Publication crawling path length is a good indicator of crawling latency. We also checked the load distribution among the servers in the distributed architectures by counting the number of pages handled by each server. For notifications, we counted the total number of messages and their distribution among the servers; to check if the distribution does not introduce expensive internal communication, we measured the total number of message hops for a random notification, including both inter-server and server-to-user messages. Finally, we measured the new costs introduced by our architecture for rank computation (network utilization) and subscription management (sum of path lengths over all the messages sent for a subscription update).

For the publication base, 10 million publications from the WebBase webcrawl were used. Publication sites were assigned to random non-server nodes in the network, then documents were routed to the pubsub/crawler server closest to the publishing node (based on number of hops in the underlying network). Half of the publications were used offline to seed the siterank computation, including at least one page from each site.

We first counted the number of hops in the underlying network needed to crawl/push all documents to the pubsub nodes, from the original random nodes to which their sites



(a)



(b)

Figure 7: Publication handling: (a) DCC and DPS are quicker in getting publications to the node(s) that can match them against subscriptions. (b) DCC and DPS partition the load among servers.

were assigned. In the case of the 1S architecture, this meant getting all publications to the single server; for the DCC architecture, first the crawling servers closer to publishers acquire the publications, then push them to the central control server; for our DPS architecture, publications are collected at the pubsub server closest to their publisher node.

Figure 7 (a) shows that the improvement in this number of hops scales linearly with the number of servers for both DPS and DCC, when compared against 1S. By eliminating the need to ultimately push all pages to a single location, DPS shows between 68-88% improvement over DCC.

Next we measured how the publication management load is divided among the systems' servers, by counting the number of publications handled by each server - the results are summarized in Figure 7 (b). DPS performs as expected and evenly dis-

tributes the publications among all servers, due to the uniform allocation of publishers to servers in our experiments. DCC has the extra cost of all publications coming to a single location, which effectively doubles the average load per server.

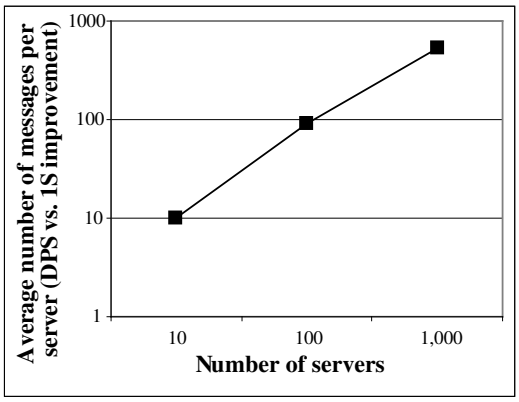
We also measure the performance of the notification component. 10 million subscriptions with 2-6 keywords each were randomly generated from the publication's vocabulary, using the same overall keyword frequency as the publications. Subscribers were randomly assigned to nodes and connected to the closest server (based on number of hops in the underlying network). We assumed in this experiment that all subscriptions are available at the start.

For this experiment, only DPS was compared against 1S, since DCC's subscription management is identical to 1S's. In 1S, a successful match generates only notification messages sent individually from the single server to the owners of the matched subscriptions. In DPS, the pubsub server that acquires the publication and performs the matching will send two types of notification messages: individual notification messages to owners of any matched local subscriptions (subscribers originally connected to this server), and inter-server messages to pubsub servers that handle other matched subscriptions. Each of these inter-server notification messages contain references to all of the matched subscriptions on the destination pubsub server, therefore significantly lowering notification traffic. Local subscriptions are delivered on the shortest path from a server to the node assigned to the matched subscription's owner.

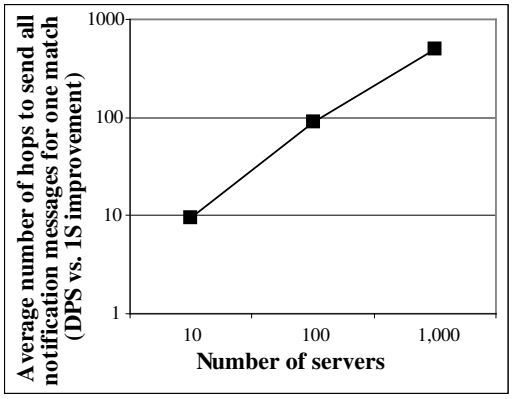
We averaged results over all notifications, with an average matching set size of 500 subscriptions per publication. Similar results were obtained when we used only the less popular publications (each matching on average 10 subscriptions) or only the most popular publications (each matching on average 1000 subscriptions).

Naturally, the total number of notification messages sent for a match in DPS is larger than in 1S, up to double the number of messages for a 1,000 node system. This is because in addition to the same number of user notification messages, the distributed setting also requires inter-server notification messages. However, when dividing the total number of notifications by the number of servers we see that DPS outperforms 1S, scaling well with the number of servers (Figure 8 (a)). The same improvement is evident when counting the total number of underlying network hops for all these messages: while DPS sends more notification messages, the inter-server notifications use the very efficient dynamic multicast, and user notification paths are much shorter from a nearby pubsub server as opposed to a single server for all users; the measurements summarized in Figure 8 (b) show that overall network usage per match scales well in DPS.

Finally, we also measured the new communication costs specifically introduced by our architecture. As shown in the previous section, SiteRank is much faster than PageRank and in our system there is no central server that must see all publications; however we still use a dedicated server to collect all the link information and send rank messages back. For a random publication, we measured the network utilization of the link summary and publication rank messages by multiplying each message size by the corresponding number of hops and summing up link summary and rank message costs. This rank server communication cost is illustrated in Figure 9 (a), as a percentage of the publication cost from publishing site to pubsub server; the measurements were averaged over 1 million publications. As expected, the smaller size of the link summary



(a)



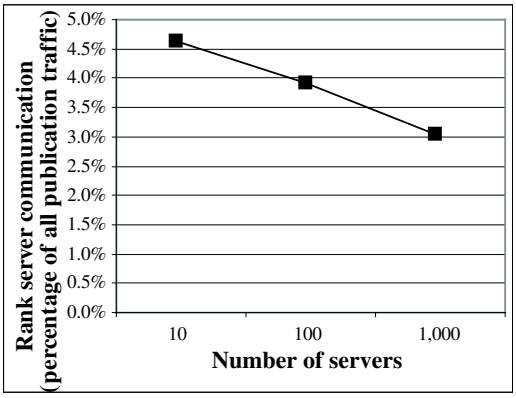
(b)

Figure 8: Notification management: (a) DPS divides the message load among multiple servers; (b) and each message travels much less.

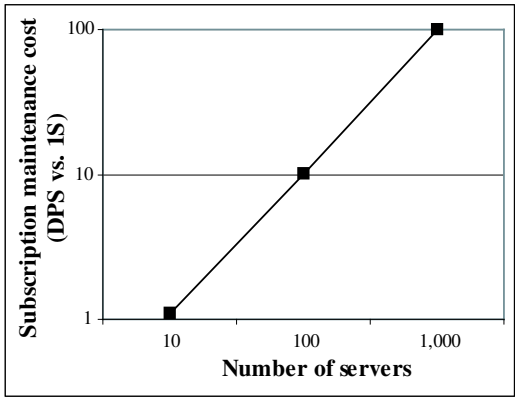
messages and high connectivity of the publish-subscribe network result in a very small relative communication overhead for the rank server which also decreases with system size from 4.6% for 10 servers to 3% for 1,000 servers.

Subscription management requires a more significant cost. While each user is connected to the closest server, each new or modified subscription must be propagated at all servers. We measured the total number of underlying network hops necessary to propagate (broadcast) a subscription from a random user to all the servers through the DPS system, and divided this by the subscription routing cost in 1S (from user to server); the results are summarized in Figure 9 (b). Even with a fully connected server graph, we see a linear increase in this communication overhead with the number of servers; each new server adds a new destination to the broadcast list.

Since subscriptions are much smaller than publications and we achieve a linear improvement of the publication routing mechanism, subscription management is not a



(a)



(b)

Figure 9: DPS communication costs: (a) Rank server communication is a small part of overall publication traffic. (b) Subscription maintenance overhead increases with system size.

big overhead: one subscription's replication cost is never higher than 1% of one publication's crawling and rank cost, even for systems of 1,000 nodes. This cost could be further reduced by replicating subscriptions in batches and/or summarizing subscriptions before replication; we have not implemented these mechanisms yet.

5 Conclusions

We have proposed SiteRank, a citation-based authority computation that is much better suited for computing ranks of fresh pages in a persistent search system than traditional approaches used in search engines. SiteRank reduces computation time considerably compared to existing algorithms and, interestingly, provides ranking results for tradi-

tional search that are not too different from those of full-fledged PageRank. SiteRank might also therefore be worth exploring for regular search systems.

In order to fully test the value of SiteRank in a persistent search system, we have proposed DPS, a distributed architecture for general-purpose persistent search that has substantial advantages over either a single server/location architecture or distributed crawling controlled from a central server. While it introduces relatively small costs for distributed subscription management through replication, we have shown that greatly reducing internal publication traffic and bringing the system closer to publishers and users significantly reduces publication and notification latency.

There are nevertheless a number of limitations to our system and we would like to address them in future work. Even though SiteRank reduces computation time, it still relies on a centralized repository of all link information (the rank server), similar to other link-relevance algorithms. Distributing this computation (on the same servers used for publication and subscription management, or a different set of dedicated servers) would eliminate this bottleneck and possibly even increase overall performance. We would also like to deploy this system as a public service on PlanetLab [24], generating real end-to-end performance measurements and truly understanding the quality of results in such systems from an end-user perspective. Making the service available to the public means we also have to improve some of its features (e.g. distributed duplicate detection) that would make it valuable to the community.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of The ACM Symposium on Principles of Distributed Computing*, 1999.

[2] C. Alerts. http://www.cnn.com/youralerts/.

[3] G. Alerts. http://www.google.com/alerts.

[4] Y. Alerts. http://alerts.yahoo.com/.

[5] R. Baeza-Yates, F. Saint-Jean, and C. Castillo. Web dynamics, age and page quality. In *Proceedings of SPIRE*, 2002.

[6] M. Bergman. The Deep Web: Surfacing hidden value. http://www.brightplanet.com/deepcontent/tutorials/ DeepWeb/index.asp, 2001.

[7] Bloglines. http://www.bloglines.com/.

[8] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, 1998.

[9] F. Cao and J. P. Singh. MEDYM: An architecture design for content-based publish-subscribe networks. In *Poster Session of ACM SIGCOMM*, 2004.

[10] J. Cho and S. Roy. Impact of search engines on page popularity. In *Proceedings of the 13th International World Wide Web Conference (WWW13)*, 2004.

[11] N. R. Feeds. http://www.nytimes.com/services/xml/ rss/.

[12] Z. R. Feeds. http://www.eweek.com/article2/0,1759,1642191,00.asp.

[13] Feedster. http://www.feedster.com/.

[14] Google Plans to Rank News By Quality. http://www.betanews.com/article/Google_Plans_to_Rank_News_By_Quality/1114819973.

[15] Grub. Distributed Web crawling project. http://www.grub.org/.

[16] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Proceedings of The 14th International World Wide Web Conference (WWW)*, 2005.

[17] T. H. Haveliwala. Efficient computation of PageRank. Technical report, Stanford University, 1999.

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

[19] 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.

[20] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the 12th International World Wide Web Conference (WWW12)*, 2003.

[21] J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer Web indexing and search. In *Proceedings of The 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[22] Monitor110. http://www.monitor110.com/.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Stanford Digital Libraries Working Paper, 1998.

[24] PlanetLab. An open platform for developing, deploying and accessing planetary-scale services. http://www.planet-lab.org/.

[25] T. S. W. Project. http://www-diglib.stanford.edu/~testbed/doc2/WebBase/.

[26] PubSub.com. http://www.pubsub.com/.

[27] A. Riabov, Z. Liu, J. Wolf, P. Yu, and L. Zhang. New algorithms for content-based publication-subscription systems. In *Proceedings of ICDCS*, 2003.

[28] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[29] K. Sankaralingam, S. Sethumadhavan, and J. C. Browne. Distributed PageRank for P2P systems. In *Proceedings of The IEEE International Symposium on High Performance Distributed Computing*, pages 58–69, 2003.

[30] M. Sceats. How big is the Web? Internet facts and figures. http://www.viz.co.nz/internet-facts.htm, 2002.

[31] Search Engine Showdown. Search engine statistics. http://www.searchengineshowdown.com, 2003.

[32] SIENA. http://serl.cs.colorado.edu/~carzanig/siena/.

[33] C. Tang and Z. Xu. pFilter: Global information filtering and dissemination using structured overlay networks. In *Proceedings of The 9th International Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, 2003.

[34] Y. Wang and D. J. DeWitt. Computing PageRank in a distributed Internet search system. In *Proceedings of the 30th VLDB Conference*, pages 420–431, 2004.

[35] J. Wu and K. Aberer. Using SiteRank for P2P web retrieval. Technical Report IC/2004/31, School of Computer and Communication Sciences, Swiss Federal Institute of Technology (EPF), 2004.