Query-Independent Ranking for Large-Scale Persistent Search Systems

Erich R. Schmidt

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

Adviser: Jaswinder Pal Singh

November 2008

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 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 (DPS), 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.

In order to fully utilize the distributed architecture of DPS and eliminate the single point of failure that is the rank server, we also propose Distributed SiteRank, a fully distributed citation-based rank computation which scales well with the number of documents and can be used in both traditional and persistent search systems.

Acknowledgments

I would like to express my thanks to my thesis advisor, Dr. Jaswinder Pal Singh, for his guidance and support during my research years at Princeton. His advice and encouragement were essential for the completion of this thesis.

I am grateful to the members of my thesis committee, Dr. Moses Charikar, Dr. Andrea LaPaugh, Dr. Kai Li and Dr. Jennifer Rexford, for taking time off their busy schedule to review my work.

I would like to thank my fellow graduate students from the Department of Computer Science, for discussions, support and camaraderie during my years in Princeton.

I would like to acknowledge Steven Kleinstein for his guidance on my initial research on Computational Immunology, and Fengyun Cao for her collaboration on Persistent Search. Working with them was a great experience.

The research presented in this thesis was supported by the National Science Foundation via grant number DGE-9972930.

To Cristina

Contents

List of Figures

List of Tables

Chapter 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 persistent query feature [21, 66]. There are also similar services being deployed directly by information providers [14, 39, 70], aggregators [5, 60] or RSS readers [23]. While dedicated services work well for specialized content (e.g. news, blogs), 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, and more obscure publications; researchers can discover new publications, grants, announcements on topics of interest; marketing departments can check if a new product is mentioned etc.

1.1 General-purpose persistent search

Persistent search services are usually built on top of publish-subscribe (pub-sub) architectures. The publish-subscribe paradigm is defined as the asynchronous communication between publishers (producers of data) and subscribers (consumers of data). Subscribers' interests are stored in the system and matched against the data produced by the publishers; the subscribers are then notified of any data that matches their interests.

Publish-subscribe systems can be classified in three categories based on the subscriptions they support:

- *Channel-based* pub-sub systems [24] classify all publications into a predefined set of channels; subscribers can elect to receive all publications in one or more channels.
- *Subject- or topic-based* pub-sub systems [38] allow subscriptions expressed as narrow conditions on a single dedicated field of each publication (subject or topic). For example, in a hypothetical academic notification service, a researcher can subscribe to all publications in the "Distributed Systems" category.
- *Content-based* pub-sub systems [11, 45] have the most flexible matching model, allowing complex conditions over multiple dimensions of the publications' content. If the researcher in the earlier example feels overwhelmed by the large amount of notifications received through the topic-based system, moving to a content-based notification service could allow a more fine-grained filter, e.g. publications that contain the word "distributed" in the title and the researcher's name appears in the bibliography.

Content-based pub-sub systems can be further separated into structured and unstructured systems. Structured systems operate on well-defined types of publications, each publication consisting of a sequence of fields (e.g. an academic publication consists of title, author, abstract, main body, references). Unstructured systems do not make any assumptions about the structure of the publications they handle and instead treat them as bags of words; this can happen if the publications lack a well-defined structure, or the system handles many different types of publications whose structures are incompatible with each other.

Due to the large structural variety of online publications (and the lack of structure in many of them) a persistent search service over the World Wide Web [21, 66] is best implemented on top of an unstructured content-based publish-subscribe system. Figure 1.1 shows a high-level view of a general persistent search service. The main elements are publishers, subscribers and the persistent search system.

- *Publishers* provide publications (text-based files in our case: txt, html, rss, xml):
  - by *actively* pushing/advertising them (via RSS or other protocols),
  - or by *passively* allowing the search system to crawl them periodically.

  Throughout this thesis, we interchangeably use the terms document, page and publication to refer to these text-based files that are published into the persistent search system.

- *Subscribers* submit queries and expect notifications of matching publications. In the context of persistent search, the search queries constitute the subscribers' subscriptions – they are stored in the system and any notification to a subscriber must match at least of of their stored queries/subscriptions.
- *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; to reduce traffic inside the persistent search system, the actual document is not sent to the user, but can be accessed at the publisher.

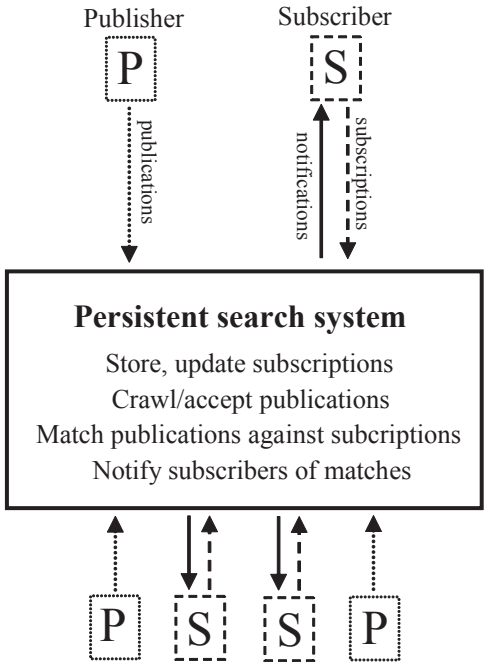

Figure 1.1: Persistent search system.

Existing persistent search systems are limited in scale, both in single-location and distributed architectures. Data throughput is always limited in a single-location architecture [21], either single-server setups or parallel/cluster solutions consisting of multiple machines on the same local network. While keyword-based subscriptions are small and relatively static, the size and growth rate of the web present a significant challenge [37] (publications come in various sizes and are very dynamic). Existing distributed solutions [10, 11, 45] do not scale well due to prohibitive communication costs when handling general keyword-based queries and unstructured text documents.

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 - while users might easily ignore low-quality results in a traditional search engine, a persistent search system must carefully limit the number of results so its users do not become indifferent to frequent, low-quality notifications.

To assess a potential result's relevance to a given query, two values are computed and combined: a query-independent measure of the page's popularity and the strength of the match between the query and the content of the page. Along with content-based relevance, traditional search engines have successfully relied on query-independent citation-based authority to assess the quality of results. While other signals are also considered when computing the relevance of a potential result to a given query (e.g. the previously mentioned match between the query keywords and the document's content), query-independent measures of the pages' authority have been proven good predictors of the pages' popularity and are used to differentiate between multiple pages whose content matches the same query. PageRank [7, 41] is a good example of using the web's link structure to assign importance to well-cited pages. This link 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 links pointing to them - while a key focus of persistent search systems is to provide new matching content quickly. PageRank does not simply rely on the number of links pointing to a page – links are not all equivalent, the rank contribution of a link is directly proportional to the authority of the originating page. However, an old page with many low-weight links can still overpower a new page, even if the new page is published on a high-quality site and it has a small number of high-weight inlinks. Previous studies [2, 12] 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.

In a distributed persistent search system, 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. Distributing the link structure data and the rank computation is a difficult problem since remote lookup of rank contributions is very costly [48, 58, 61].

1.2 Goals

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.

Fresh results are only useful if they are also of high quality, so we first focus our research on improving citation-based authority. Our main goal is reducing the bias against new pages, which need more quality signals to rank well in the absence of many links pointing to them. The proposed solution is 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 to PageRank. While we introduce this new algorithm in the context of persistent search, SiteRank is a viable alternative to PageRank in traditional search systems too.

Having reduced the bias against new pages in ranking and improved the predictability of link-based relevance computation, we concentrate our efforts on building a scalable persistent search system. To increase publication throughput, we distribute the underlying publish-subscribe 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.

SiteRank, our proposed authority computation, still requires that all link structure is known at a single location, the rank server, even if the document processing (matching against subscriptions, duplicate detection) is distributed. This comes at the small communication cost but more importantly introduces a single point of failure in an otherwise fully distributed persistent search system. Distributing the rank computation would eliminate this bottleneck and increase overall system performance. If naively distributed, both PageRank and SiteRank add prohibitive communication costs. We therefore propose a new partitioning scheme for the link graph that significantly reduces the network load during distributed rank computation, while accurately computing the same ranks as the much slower single-server methods. Similar to SiteRank, Distributed SiteRank is a viable alternative to PageRank and more naive distributed PageRank implementations in existing distributed traditional (non-persistent) search systems [25].

1.3 Thesis organization

Immediately following this introduction, Chapter 2 summarizes our work on improving the link-based relevance computation used in traditional and persistent search systems. Based on current popularity among traditional search engines, we decided to improve on PageRank. In its current form, it has proven to be a good indicator of page popularity. However, it has one major flaw that is especially noticeable in persistent search: new pages have few links pointing to them and are therefore considered low-quality. We describe other attempts to improve on PageRank, give a high-level overview of SiteRank – our proposed approach that removes the bias against new pages – and conclude with experimental evaluation that illustrates the accuracy and performance of our solution.

In addition to improving our predictions about page popularity, SiteRank also significantly speeds up the rank computation. This is a very important aspect in a real persistent search system, so to properly assess its impact we also implemented a general-purpose keyword-based persistent search system. In Chapter 3, we first argue that a truly scalable persistent search system must be distributed, then propose and evaluate Distributed Persistent Search (DPS).

Using multiple publish-subscribe servers should lower publication latency. SiteRank speeds up link relevance computation and therefore should allow more frequent re-computation, keeping scores up-to-date. Together, the distributed publish-subscribe architecture and the site-rank-based relevance computation form a scalable persistent search system serving fresh and relevant results over a large publication base.

In a distributed system like DPS, the single-server rank computation constitutes a single point of failure and limits the system's scalability since it requires that all link information is collected in a single location. While naive distributions of the rank computation have prohibitively high communication costs, better partitioning schemes could be more practical. Chapter 4 summarizes our attempt to distribute SiteRank using dense-component based graph partitioning.

From an algorithm design point of view, Distributed SiteRank should have immediately followed SiteRank. However, a major aspect of Distributed SiteRank is the actual distribution of the documents across the servers, which influences other components in the Distributed Persistent Search system (e.g. can we publish documents efficiently to the servers assigned to each partition?). We are therefore evaluating the performance of Distributed SiteRank as part of the Distributed Persistent Search system and have ordered the chapters accordingly.

Finally, Chapter 5 summarizes all our results and lists possible improvements. To better illustrate our evaluation, we also include tables of our processed experimental results in Appendix A and some of the raw data used to derive these results in Appendix B.

Chapter 2

SiteRank

Recent years have seen an accelerated reliance on using online data and services in all aspects of life and work. Search engines are important tools that help us discover relevant data from a variety of sources. While accessing a large collection of sources increases our chances of finding documents that are meaningful to our query, it does not necessarily increase the quality of the results. Along with content-based relevance, traditional search engines have successfully relied on citation-based authority to measure the quality of their results; PageRank [7, 41] is the most popular and arguably the most successful example of using the web's link structure to assign importance to well-cited pages.

This thesis covers research in persistent search systems; since it actively notifies users of new content, a persistent search system has an even higher pressure to provide only high quality and highly relevant results than a traditional search engine. The main problem with PageRank-style algorithms is that they assign very low authority to new pages - since they by definition have few inlinks - while a key focus of persistent search is to provide new matching content quickly. Previous studies [2, 12] 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.

We start this chapter with a short look at previous work in this area, including a summary of the PageRank algorithm; our proposed solution, SiteRank, is described in detail; we continue with a description of our experimental setup and evaluation results, and conclude with a short summary.

2.1 Related work

The overall relevance score of a web page for a given query is computed as the combination of two numbers: a query-independent rank that summarizes the relative importance of the page and a content-based IR relevance value.

In the context of web search, ranking pages based on content-based relevance alone would often return skewed results. The commercial aspect of the web is an important incentive to artificially manipulate a page's content (spamming) to increase the likelihood that the page is returned by a query search. Ideally, the IR score should by itself determine the page's relevance to the query terms, but supplemental signals are needed to determine the page's overall quality.

Most query-independent page scores rely on the link structure between documents to assess a page's quality. In our work, we focused on the most popular - Pagerank [41]; other major query-independent page quality systems include HITS [34], which computes a hub score and an authority score, and SALSA [36], a hybrid approach between PageRank and HITS.

2.1.1 PageRank

As Google's [20] popularity has steadily increased in the past years, web designers and search optimizers have become increasingly aware of PageRank and its effect on search results: one important measure of the quality of a webpage comes from the number and quality of its inlinks. We have been applying the same criterion in academia and research even before the ubiquity of online search - papers with more citations are more popular. Today, an important signal in every major search engine is the citation-based quality of webpages, making use of the auxiliary information provided by the link structure of the web. The underlying assumption is that web users are more likely to visit more important pages, and they also preferentially link to pages they consider important.

The PageRank [41] of a page is based on the link structure of the web - an inlink from a page is interpreted as a citation or recommendation from that page's author. The number of links alone is open to spamming, so an important part of the recommendation is the weight carried by the link. In general, more important pages will have more inlinks, and links from more important pages carry more weight. PageRank is defined in a recursive manner, since the rank for any particular page is built based on the ranks of the pages linking to it.

PageRank is usually described using the random human surfer model - a hypothetical user navigating through a series of webpages. When viewing a webpage, the user may follow any of the page's outlinks to other pages on the web. Webpages can be modeled as nodes in a directed graph, where edges between nodes represent links between pages - we will be referring to this as the *link graph*.

For webpage i, we define F_i as the set of pages i points to (its outlinks, or forward links), and B_i as the set of pages pointing to i (its inlinks, or backward links); also, let $N_i = |F_i|$ be the number of links from i (to reduce the impact of link spam, only a single link in each direction between any two pages is counted), and let *c* be a normalization factor that ensures that the total rank of all pages is constant. We can then define a simplified version of PageRank, ranking R for page i:

$$R(i) = c \times \sum_{j \in B_i} \frac{R(j)}{N_j} \tag{2.1}$$

The above equation summarizes some basic concepts in PageRank:

- If a human surfer is on a random page j, she has an equal probability of following any of the links out of this page, therefore we evenly divide the rank of this page by its links to determine an individual link's weight: $R(j)/N_j$.
- The human surfer can end up on page i by following any of its incoming links B_i, so the rank of a page is computed by summing up the individual weights (rank contributions) of its inlinks.
- The weight of any link is directly proportional to the probability that the surfer will traverse that link. A link can have a higher weight if:
  - the originating page has a higher rank, and therefore a higher probability to be visited by the human surfer; or
  - the originating page has fewer outgoing links, and therefore the probability of following any of them is higher than in the case of a page with more outlinks.
- If the correspondence between page ranks and link weights seems circular, it really is - link weights are just rank transfers between pages. When computing PageRank, there is no need to compute the link weights separately, since they follow directly from the page rank values.

Equation 2.1 is recursive, but it can be computed sequentially by starting with any seed values in R and iteratively refining the ranking function until it converges.

The weight of pages with no forward links is lost from the system, hence $c > 1$. Also note that we're dividing each page's rank among its outlinks - another way to encode this is using a simple square matrix A (rows and columns correspond to web pages) where $A_{i,j}$ = $1/N_i$ if there is a link from page i to page j and 0 if not; if there are multiple links from page i to page j, only a single one is counted since each page should get a single vote towards the rank of another page. This is not just a simple adjacency graph, since each non-zero entry encodes the weight carried by the link between two pages and not just the existence of the link. Similar to Markov chains, we call A a *transition probability matrix*, as it encodes the probability of following any possible link from the present page i.

A small adjustment is necessary to our random surfer model: some pages form closed loops with no outward links, so the surfer cannot advance to a new page, and the rank of these pages is not distributed further in the graph. To make up for these cycles, a vector E over all webpages is used as a source of rank, injecting a small weight into the rank of each page and offering the surfer a way to randomly jump to other pages. A refined version of equation 2.1 is:

$$R(i) = c \times \sum_{j \in B_i} \frac{R(j)}{N_j} + c \times E(i) \tag{2.2}$$

The PageRank values are normalized, so $||R||_1 = 1$ (the L_1 norm of R). In our work, E was selected to distribute weight uniformly over all web pages, with $||E||_1 = 0.15$. This assumes that the random surfer can periodically jump to any random page from the current page and ignore all outlinks.

Rewriting equation 2.2 in matrix notation, we have $R = c(AR + E)$. Since $||R||_1 = 1$, this leads to:

$$R = c \times (A + E \times \mathbf{1}) \times R \tag{2.3}$$

where $\mathbf{1}$ is the vector consisting all of ones. Mathematically, we could compute R as the principal eigenvector of $(A + E \times \mathbf{1})$. In practice, the link graph is prohibitively large for this computation, so we resort to an iterative solution: R_{i+1} is computed from R_i based on equation 2.3, until $||R_{i+1} - R_i||_1$ is smaller than some threshold.

Given the vast size of the web, even the iterative solution is extremely time-consuming on distributed computing clusters. PageRank's major weakness consists in the time it needs to recompute page quality coupled with the highly dynamic nature of the web. New pages and links to existing pages appear continuously, but their contribution will not be reflected until the next PageRank computation finishes.

2.1.2 Improving PageRank

PageRank is the most popular link-based authority computation, so naturally there have been many projects trying to improve or expand its results.

Far from being random, the web exhibits a nested block structure, where pages on the same site (domain) preferentially link to one another; Kamvar et al. [31] take advantage of this structure and propose BlockRank, a 3-stage algorithm:

- for each domain, local ranks are computed for all pages using only the links internal to that domain; this step can be easily parallelized since each domain's computation is completely self-contained;
- the local page ranks are weighed by the rank of their domains (which were computed as regular ranks over the reduced graph of inter-domain links);
- the adjusted local scores are used as the starting vector in standard PageRank, reducing the number of iterations required to converge.

For a large data set (290 million pages), this approach reduces the number of full-graph PageRank iterations from 28 to 18. After adding the cost of the intra- and inter-domain rank computation and adjustment, the overall performance improvement is a constant factor of 2 for a full PageRank computation, and 3 for incremental updates (when only parts of the full computation need to be performed). This approach, while measurably reducing the computation time, does not address PageRank's bias against new pages.

The same authors [32] also introduce a quadratic extrapolation method to accelerate the convergence in computing PageRank. Periodically subtracting off estimates of the non-principal eigenvectors from the current iterate can speed up the PageRank computation by 25-300%. This approach therefore improves rank computation performance by a small constant factor (up to 3), but again does not address PageRank's bias against new pages.

Haveliwala [27] uses a block-based strategy to compute PageRank efficiently and proposes topic-sensitive PageRank vectors [28]. All these approaches, and many others, while addressing different issues

and improving on the original PageRank algorithm in various ways, do not handle new pages well since they directly follow the original PageRank model. Each page's rank is directly determined by its links alone. Recently, ranking of sites has also been taken into consideration. Google announced that it plans to rank pages by the quality and reliability of the publishing site (source), for its Google News service [18].

In parallel with our work, two other teams have identified the importance of the host-graph and proposed improvements to PageRank based on this observation. Wu and Aberer [62] introduced rank computation in the host graph (named SiteRank, same as our proposed algorithm) and observed that page ranks (as given by traditional PageRank [41]) and site ranks follow very similar power-law distributions. The authors computed local page ranks based on links within each site, then combined the site ranks and local page ranks to get the global page ranks. If Kendall's Tau distance is computed between their ordering and the traditional PageRank ordering (see section 2.3), we see that the proposed approach is very close to traditional PageRank (a distance of about 10%, although this was measured on small data sets smaller than 25 thousand pages). The authors did not evaluate how well their local page ranks can predict the popularity of pages on large graphs, nor did they evaluate the speed of their proposed algorithm. Their main motivation for the use of site ranks is to distributed the rank computation to multiple servers, and we will discuss this aspect in section 4.1.

More recently, Xue et al. [64] aggregated web pages at directory, host and domain level and performed link analysis on this reduced graph with the intention of overcoming PageRank's bias against new pages. In their HostRank approach, host ranks are distributed to pages using a dissipative heat conductance model, based on the pages' position on the host and the percentage of the host's inlinks pointing to each particular page. Pages closer to the root of the host receive more of the host's rank than pages further away. Also, pages with more incoming links from other hosts have a higher weight as well, but links internal to the host are not used at all.

However, computing page ranks on flat-structured hosts yields poor results, since the shallow directory structure is not a good quality signal in this case, and HostRank does not differentiate between various page links but treats all of them identically. It should be noted here that many high-authority hosts (e.g. library catalogues, shopping and review sites) contain dynamically generated content which is presented in a very flat internal structure.

HostRank completely ignores the weight of the incoming links, adjusting the page ranks based on the percentage of the host's inlinks that point to each page. This approach fails to differentiate between new

pages with few inlinks coming from other hosts when those links carry vastly different weights; highly-relevant pages cited by high-authority external sources lose a significant part of their rank to less-relevant pages that happen to have a similar number of inlinks. Also, not using host-internal links in the individual page rank adjustment can hurt the rank of pages that are deemed important by the host owners, e.g. index pages that many other pages from the same host link to.

It must be noted that all these proposed alternative to PageRank were general solutions intended to fix various shortcomings of PageRank (speed, bias against new pages), none of them was designed specifically to be used in a persistent search system.

2.2 SiteRank design overview

As mentioned earlier, new pages do not have many incoming links and are therefore assigned low PageRank values. Drawing on Wu and Aberer's [62] observations that the rank of pages and the rank of sites containing these pages follow similar distributions, we propose to derive the ranks of individual pages from their sites' rank. We expect that new pages from authoritative sources (highly-ranked sites) will be the main beneficiaries of this approach, but any new page will gain some rank and will not be always ignored in favor of older pages.

An expected positive side-effect of this approach is speeding up the computation of page ranks. Search engines compute PageRank periodically, after refreshing their index; this algorithm is computationally intensive, due to the extremely large set of data it works on. Computing site ranks is much faster than computing PageRank due to the much reduced link graph. Since the web grows at a high rate and new pages arrive continuously, speeding up the link authority computation is by itself a major improvement, especially in the context of persistent search systems that must quickly notify their subscribers of new high-quality content,

We propose SiteRank - a modification of the classic PageRank algorithm [41] that handles new publications well, even in the absence of many inlinks. The rest of this section describes our solution in detail (with a short summary of all the SiteRank computation steps in section 2.2.3), followed by a presentation of our experimental evaluation in section 2.3.

HostRank by Xue et al. [64] is the most-promising current alternative to PageRank that tries to solve the same problem – PageRank's bias against new pages. Therefore, our proposed algorithm SiteRank is

evaluated against both traditional PageRank and HostRank. Similar to HostRank, SiteRank computes the ranks of sites and uses these to compute the ranks of individual pages. Unlike HostRank, SiteRank uses more signals that can better differentiate between pages of different rank on the same site:

- Similar to HostRank, SiteRank assigns progressively smaller weight to pages further away from the site's root.
- To differentiate between pages on the same level, especially in sites with a shallow directory structure, the weights of the individual links pointing to each page are then used to adjust page ranks in SiteRank. Unlike HostRank, which only uses the percentage of a site's incoming links to determine the relative importance of an individual page, SiteRank can differentiate between pages based on the weight of individual links pointing to different pages on the same site. This is especially important for new pages, since they only have a very small number of incoming links, and the weight of an incoming link from an authoritative source can be the most significant contributor to a new page's rank.

2.2.1 Sites

Throughout the following sections, we will use the following notations:

- We expand the previous definitions (section 2.1.1) of B_i and F_i to sites, i.e. they will stand for the back and forward links of page p_i or site s_i.
  - B_{p_i} is the set of all unique back links of page p_i; we will refer to either the links, or the originating page or site as members of this set.
  - B_{s_i} is the set of all unique back links of site s_i; we will refer to either the links, or the originating page or site as members of this set.
  - F_{p_i} is the set of all unique forward links of page p_i; we will refer to either the links, or the destination page or site as members of this set.
  - F_{s_i} is the set of all unique forward links of site s_i; we will refer to either the links, or the destination page or site as members of this set.
  - $NB_{p_i} = |B_{p_i}|$ is the number of unique back links of page p_i.

  - $NB_{s_i} = |B_{s_i}|$ is the number of unique back links of site s_i.
  - $NF_{p_i} = |F_{p_i}|$ is the number of unique forward links of page p_i.
  - $NF_{s_i} = |F_{s_i}|$ is the number of unique forward links of site s_i.

- R_{p_i} is the rank of page p_i.
- R_{s_i} is the rank of site s_i.
- NP_{s_i} is the number of pages on site s_i.
- LC_{p_i} is the link contribution for page p_i, defined as the rank value of the page divided by the number of unique outlinks from that page (remember, we only count a single link in each direction between any two pages). Similar to the transition probability matrix A used in PageRank (see section 2.1.1), it represents the weight carried by each outgoing link: $LC_{p_i} = R_{p_i}/NF_{p_i}$.
- Similarly, LC_{s_i} is the link contribution for site s_i: $LC_{s_i} = R_{s_i}/NF_{s_i}$.

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 [62, 64]. 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. We defined a site as the collection of pages with identical hostname (the first part of their URL, not including directory and page name). For some hosts, this might results in extremely large page counts for the same site (e.g. geocities.com), in which case partitioning the host into multiple sites is necessary. The data we used for our experimental evaluation [30] artificially limited the number of pages from the same site to at most 10,000, and was therefore usable without considering multi-site hosts. It also did not contain sites similar to geocities.com where multiple different logical sites are sharing the same domain. These sites can be problematic as they contain many logical sites within them, and the internal link pattern is different from

regular sites: each logical site has links between its pages and to/from other domains, but there are very few links between pages on different logical sites on this same domain. In practice, this characteristic can be used to identify and partition these sites into the individual logical sites they consist of.

We must adjust the transition probability matrix A from section 2.1.1 so that the probability of moving from a site s_i to a site s_j is directly proportional to the number of pages from s_i linking to pages on s_j: $A_{s_i,s_j} = N_{s_i,s_j}/N_{s_i}$, where N_{s_i,s_j} is the number of pages on site s_i that link to any page on site s_j, and $N_{s_i} = \sum_{p_k \in s_i} NF_{p_k}$ is the sum of the number of individual forward links NF_{p_k} from all pages p_k on site s_i. To reduce the impact of link spam, only a single link in each direction between any two pages is counted; this is true throughout all SiteRank computations. By counting only unique links between any page pair, N_{s_i,s_j} is directly proportional to both the number of links and the number of pages linking from s_i to s_j, and therefore each link constitutes one confidence vote, similar to regular PageRank.

Note that we do not need to normalize the rows after this initialization, since the sum of all entries for any particular row is 1. For example, if a site with a single page containing links to two pages on distinct sites; the transition probability for each of these site-to-site destinations is 1/2, since the only page on the originating site has links to both destination sites.

Similar to PageRank, SiteRank is the principal eigenvector of $(A_s + E \times \mathbf{1})$ (see section 2.1.1, A_s is the site-level transition probability matrix as defined above), and it is computed through the iterative solution of the equation:

$$R_s = c \times (A_s + E \times \mathbf{1}) \times R_s \tag{2.4}$$

The 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 the smaller site-graph compared to the original page-graph, and the overall computation converges faster. For example, on the 10 million page data set, the number of iterations was reduced to 20 in SiteRank, while PageRank required 30. This allows rank scores to be recomputed more often, a welcome option in a system processing a high volume of new publications.

2.2.2 Pages

The second part of SiteRank consists of computing the actual page rank values. To compute the final rank for any given page, we take into account multiple factors: the rank of the site containing the page; the

position of the page on the site, based on the site's directory structure; finally and most importantly, the individual links pointing to the page - from other pages on the same site, and from pages on other sites.

- Based on the observed correlation between site ranks and page ranks, the starting rank value for each page is based on *the rank of the site containing the page*. 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 $R_{p_j} = R_{s_i}/NP_{s_i}$ for every page p_j on each site s_i.

- *The page's position on the site*, given by the directory structure, also contains some useful information about the relative importance of the page - in general, pages further away from the root have lower rank values compared to pages closer to the root. While not necessarily always true, this is a good indicator when only a small number of external links are available [64]. For each site, the total rank (the rank of the site, equal to the sum of the ranks of all the pages on the site) is redistributed based on the site's directory structure. A constant reduction factor f is applied between each consecutive levels, i.e. a page at level l is assigned a rank value f times larger than a page at level $l + 1$; the root of the site (index page, other top-level pages) are at level 0. Our experiments show that a value of $f = 1.2$ 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 sites consisting mainly of dynamic pages. More signals are needed to differentiate between individual pages on such sites, or pages at the same level on deeper sites.

- 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 based on the directory structure, we can take advantage of the fact that page rank values are different from site rank values and adjust link contributions between pages accordingly.
  This is not a full page-rank computation, it is a sequence of operations performed once for all pages to compute their rank. It is even possible to only compute the ranks of some pages, together with any intermediate values (site and page rank contributions) needed for those pages only. Since there is a very small chance that any search system needs the ranks for all pages after each repository refresh operation, we can save a significant amount of time by only computing the ranks of the pages we consider as results for a given query.
  The link contributions for each page are computed based on their current rank, which was determined

in the previous two steps using the site ranks and the directory structure of the sites. In order for the final page rank computation to be consistent throughout all pages, we do not update the link weights for any page after computing the page's final rank, so that the rank of all pages is computed based on the same input.

The initial citation authority in our algorithm was transferred through inter-site links, and so far we only propagated this value into the page authority through the first step of the page rank computation, which treats all the pages on a site the same way, disregarding whatever individual authority they build up through links. According to this method, the main page of the New York Times website [39] *www.nytimes.com/index.html* is just as authoritative as its privacy policy page *www.nytimes.com/privacy.html*. To compute a more accurate link authority value for a given page, we look at each link pointing to that page and adjust its current site-based contribution using the rank of the originating page. Given page p_1 on site s_1, with a back-link from page p_2 on site s_2, we compute this link's contribution to the rank of p_1 (R_{p_1}) as follows:

– To determine the fraction of the current score of page p_1 coming from page p_2, we first compute the rank contribution of site s_2 towards the rank of site s_1 as the sum of the link weights going from s_2 to s_1:

$$RC_{s_2 \to s_1} = LC_{s_2} \times |F_{s_2} \cap B_{s_1}| \tag{2.5}$$

The relative rank contribution of site s_2 to site s_1 divides the absolute rank contribution $RC_{s_2 \to s_1}$ by the sum of all rank contributions towards s_1:

$$RRC_{s_2 \to s_1} = \frac{RC_{s_2 \to s_1}}{\sum_i RC_{s_i \to s_1}} \tag{2.6}$$

where i iterates through all the sites s_i with forward links to site s_1.

– The current page rank contribution from p_2 to p_1 is based only on inter-site links, so we can compute it based on the above relative site rank contribution. Since not all sites linking to s_1 are actually linking to p_1, we re-normalize the rank contribution based on the relative rank contributions coming from sites linking to p_1:

$$RC_{s_2 \to p_1} = R_{p_1} \times \frac{RRC_{s_2 \to s_1}}{\sum_i RRC_{s_i \to s_1}} \tag{2.7}$$

where i iterates through all the sites s_i with forward links to page p_1.

– The final rank contribution from p_2 to p_1 is computed by taking into account the actual link contribution of p_2 instead of the link contribution from site s_2:

$$RC_{p_2 \to p_1} = RC_{s_2 \to p_1} \times \frac{LC_{p_2}}{LC_{s_2}} \tag{2.8}$$

The final rank of page p_1 is computed as the sum of the scaled inter-page contributions over all its inlinks:

$$R_{p_1} = \sum_i RC_{p_i \to p_1} \tag{2.9}$$

where i iterates through all the pages with forward links to page p_1.

While it would be possible to use PageRank inside each site to compute individual pages' ranks [62], 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; in contrast, SiteRank performs quick link authority computation for all pages.

Fully refreshing the whole link graph can be very expensive, but the proposed SiteRank algorithm can incrementally compute the page ranks on request when new pages or links are added to the graph, using the stale site rank values. These page rank values would of course be different from the result of a full-scale re-computation of all ranks, but it's a good approximation for new pages' ranks, especially considering that the alternative is to use a rank of 0 until the next full rank computation. Also, even when the full link graph is refreshed and site ranks are recomputed, we can lazily compute only the page ranks for the pages that we consider as part of a reply to a query, further reducing the computation cost of SiteRank.

2.2.3 SiteRank summarized

To summarize the previous subsections, we present here the sequence of operations performed to compute all page ranks for the given link graph.

To compute site ranks:

$$
\begin{aligned}
N_{s_i} &= \sum_{p_j \in s_i} N_{p_j} && \text{for all sites } s_i \\
N_{s_i,s_j} &= |F_{s_i} \cap B_{s_j}| && \text{for all sites } s_i \text{ and } s_j \\
A_{s_i,s_j} &= N_{s_i,s_j}/N_{s_i} && \text{for all sites } s_i \text{ and } s_j \\
R_s &= c \times (A_s + E \times \mathbf{1}) \times R_s
\end{aligned}
$$

To compute the page rank of page p_m on site s_i:

$$
\begin{aligned}
R_{p_n} &= \frac{R_{s_k}}{NP_{s_k} \times f^l} && \text{for all sites } s_k, \text{ for every page } p_n \text{ at level } l \text{ on site } s_k \\
RC_{s_j \to s_i} &= LC_{s_j} \times |F_{s_j} \cap B_{s_i}| && \text{for all sites } s_j \in B_{s_i} \\
RRC_{s_j \to s_i} &= \frac{RC_{s_j \to s_i}}{\sum_{s_k \in B_{s_i}} RC_{s_k \to s_i}} && \text{for all sites } s_j \in B_{s_i} \\
RC_{s_j \to p_m} &= R_{p_m} \times \frac{RRC_{s_j \to s_i}}{\sum_{s_k \in B_{p_m}} RRC_{s_k \to s_i}} && \text{for all sites } s_j \in B_{p_m} \\
RC_{p_n \to p_m} &= RC_{s_j \to p_m} \times \frac{LC_{p_n}}{LC_{s_j}} && \text{for all pages } p_n \in B_{p_m} \text{ on site } s_j \\
R_{p_m} &= \sum_{p_n \in B_{p_m}} RC_{p_n \to p_m}
\end{aligned}
$$

2.2.4 Predictive SiteRank

The proposed SiteRank algorithm takes into consideration site ranks, directory structure and individual page links to help new pages gain authority while their set of back links is small. As our results in section 2.3 show, we can successfully predict the popularity of most new pages, as SiteRank maintains a page ordering very similar to traditional PageRank. A small accuracy loss is present when comparing pairs of pages on the same site, as their relative order is not maintained by SiteRank when a new page is compared to an older page with a non-trivial set of back-links. Their relative ordering is mainly due to the heuristics we use to compute page ranks from the rank of the site.

New pages on the same site, at the same level, with similar, very small number of inlinks end up with very similar page rank values, independent of their individual contribution to the site. To better predict the distribution of future links, we analyzed consecutive web crawls for the same set of sites and observed that many sites have a tendency to add links to new pages on other sites they are already linking to, if the new

page is topical to content that already exists on the links' originating site. Measuring the fraction of new links in our experimental data (consecutive web crawls from the Stanford WebBase project [30], containing 100 million pages each), we found that up to 70% of new links between pairs of sites show up if the new page's title matches some content on the links' originating site and there are already links between the two sites. An every-day example of this behavior can be observed as a local newspaper usually links to the same major news agency when reporting new stories.

Predictive SiteRank adds this new information on top of the basic SiteRank algorithm described in sections 2.2.1 and 2.2.2. For a new page p_1 on site s_1, we check all other sites s_i that link to site s_1: if there is a match between the current content of any page p_j on site s_i and the title of the new page p_1, we insert a *virtual link* from p_j to p_1, which is only used during the page-rank computation phase of SiteRank. In case that multiple pages from site s_i match the new page's title, the one with the highest rank (based on position within the site) is selected. Since this is a speculative adjustment trying to predict future links, the new link is not counted among the regular outlinks of p_j, F_{p_j}, and its weight is just half of the weight of a regular link.

A second piece of information neglected so far is the age of pages. If we know a page is new, we can artificially increase its final rank to compensate for the limited inlinks. For regular search, this can reduce the bias against new pages, but it does so without using any authority signal and is therefore easily exploitable by low-quality pages. For persistent search, this additional adjustment is not especially useful since most incoming pages are new. Therefore, we did not pursue this direction in our work.

Our experiments, described in section 2.3, show that Predictive Siterank improves marginally on SiteRank. However, we must pay the additional price of storing all the pages' content, which is not regularly used in rank computation, and checking for content-based matches during the rank computation. The additional costs make this approach much less practical, especially taking into consideration the small incremental improvement over regular SiteRank. One alternative would be matching the title of the new page only against the more significant content of the existing pages (titles, anchors), but this reduces the match rate from 70% to about 5-10%, with a similar reduction in cost. However, since the impact of this change is vastly reduced, the extra effort does not justify the added cost in storage and computation costs.

2.3 Evaluation

We measured the accuracy and performance of the SiteRank-based page rank computation, compared with the classic PageRank algorithm. The accuracy of SiteRank was measured by comparing the rank order as defined by SiteRank and PageRank. Since the main motivation of our search is improving the ranks of popular new pages in a persistent search system, further experiments evaluating SiteRank as a component of a distributed persistent search system are also presented in section 3.3.

Following the example of other projects working to improve PageRank, we implemented a basic version of PageRank [41] also used when evaluating other PageRank improvements [31, 62, 64].

The publication data set used initially for these experiments is a July 2004 text webcrawl from the Stanford WebBase project [30], 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 (variation between various runs was up to 4-5% for the small subsets of 100,000 pages, and as low as 0.5-1% for the large 10 million page subsets). Later, consecutive crawls of 90-100 million documents also became available (September and October 2005, March and April 2007) and were used to confirm our earlier results, and to test how well SiteRank predicts the popularity of new pages.

Based on the current widespread use of PageRank and similar link-authority computations, we are considering the PageRank value of old pages to be a good indicator of their popularity, and use the PageRank-induced ordering of web pages as reference when evaluating our proposed improvements.

In order to test the accuracy of our new approach, we compared the rank order of page authority values as given by the SiteRank-based computation versus the classic PageRank computation. For each document set used in the evaluation, both PageRank and SiteRank was computed independently; after each computation, the pages were ordered in decreasing page rank order, separately for PageRank and SiteRank. While the actual page rank values were quite similar between the two computations, comparing the two approaches comes down to the relative order of pages in the two ordered lists. For page p_i, we use $p_{p_i}^{PR}$ and $p_{p_i}^{SR}$ to denote its position in the PageRank- and SiteRank-based ordered page lists.

To compute disagreements between two ordered lists, Kendall's tau distance [17] (*KDist*) counts the page pairs that flip their rank order between the two orderings:

$$|\{(p_1, p_2) : ((p_{p_1}^{PR} < p_{p_2}^{PR}) \wedge (p_{p_1}^{SR} > p_{p_2}^{SR})) \vee ((p_{p_1}^{PR} > p_{p_2}^{PR}) \wedge (p_{p_1}^{SR} < p_{p_2}^{SR}))\}| \qquad (2.10)$$

Given n pages, a KDist of 0 means the two ordered lists are identical; a KDist of $n \times (n-1)/2$ means one list is the reverse of the other. We normalize this value to the interval $[0, 100]$, dividing it by $n \times (n-1)/2$, and converting it to a percentage.

The actual page rank values are very similar between the two lists. Also, the page rank is one of many signals used to compute the relevance of a document to a given query. Given two documents with similar query-dependent support, their query-independent rank order (PageRank or SiteRank-based) will correctly identify the better document. We consider therefore that testing the relative rank ordering is a good way of evaluating how well our proposed rank computation identifies more popular pages.

2.3.1 Rank prediction for new pages

The main motivation for SiteRank is to correct the bias of PageRank against fresh pages with a small number of inlinks. HostRank, proposed by Xue et al. [64] (see section 2.1.2), also aims to solve this problem, and we include it in our evaluation to illustrate the importance of individual page links as used in SiteRank.

Due to space restrictions and data availability, our initial data set consisted of a single crawl containing 11 million pages; later in this section, further experiments are presented that were performed with consecutive page crawls containing 90-100 million documents each. In order to simulate fresh pages gaining links over time, we used the same pages with incomplete links as an older version of the same data set. Removing some of the links gave us an *older* version of the data set; we computed PageRank, HostRank and SiteRank over this reduced graph to check how well each of these methods can predict the popularity of new pages. A reference ordered set was generated computing PageRank over the full data set with all the links, illustrating the *future* popularity of all pages.

We randomly selected 5% of the pages as new pages, with uniformly distributed initial rank values. These pages' incoming links were reduced by 1%, 25% and 50%, and predictability was measured by computing KDist between each of the methods' ordered list on this partial graph and the future reference list. Multiple trials of the experiment were averaged at each link reduction level, selecting a new random subset of links in each trial. There was little variation visible between different trials at the same level (up to 4-5%), as long as none of the sites changed more than 80% of its pages; the results presented here exclude these degenerate cases.

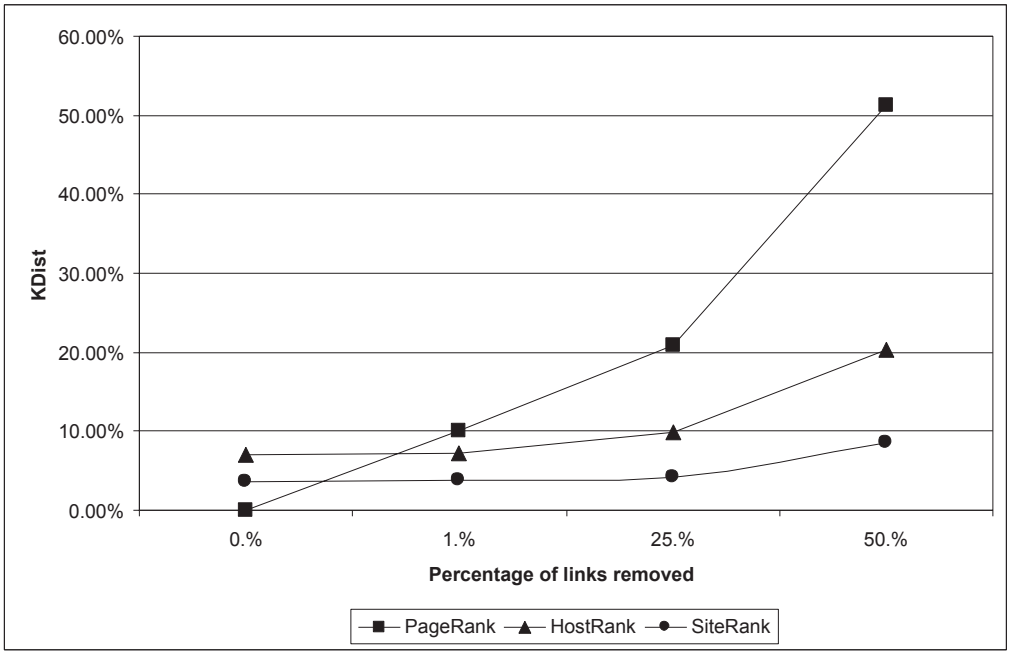

Figure 2.1: KDist over all page pairs, complete graph vs. reduced links.
Data in Table A.1.

Figure 2.1 summarizes the results over all page pairs, comparing how well PageRank, HostRank and SiteRank can predict the rank of new pages in this synthetic graph aging setup. As expected, PageRank is by far the weakest predictor of the three, since changes in the link graph are immediately reflected in the pages' relative ordering. HostRank uses the sites' rank and the pages' position within the sites; summarizing ranks at site level results in more robust page ordering, since the omission of few links has smaller overall effect on ranks. This is of course based on the assumption that page ranks and site ranks are highly correlated, which has been reported in multiple publications; naturally, sites containing highly-ranked pages will be more likely to collect new links (to their existing and new pages) than sites with fewer initial links.

Not surprisingly, SiteRank outperforms HostRank, proving that even for new pages, individual links matter (the major design difference between HostRank and SiteRank) - a link from a highly-ranked source, carrying higher weight, is a good signal that its destination is a high-authority page. Even in the absence of many links, we see that SiteRank is very good at maintaining the overall page ordering. This is more visible if we restrict our comparison to page pairs where both pages are new, i.e. the pages whose incoming link sets were manipulated.

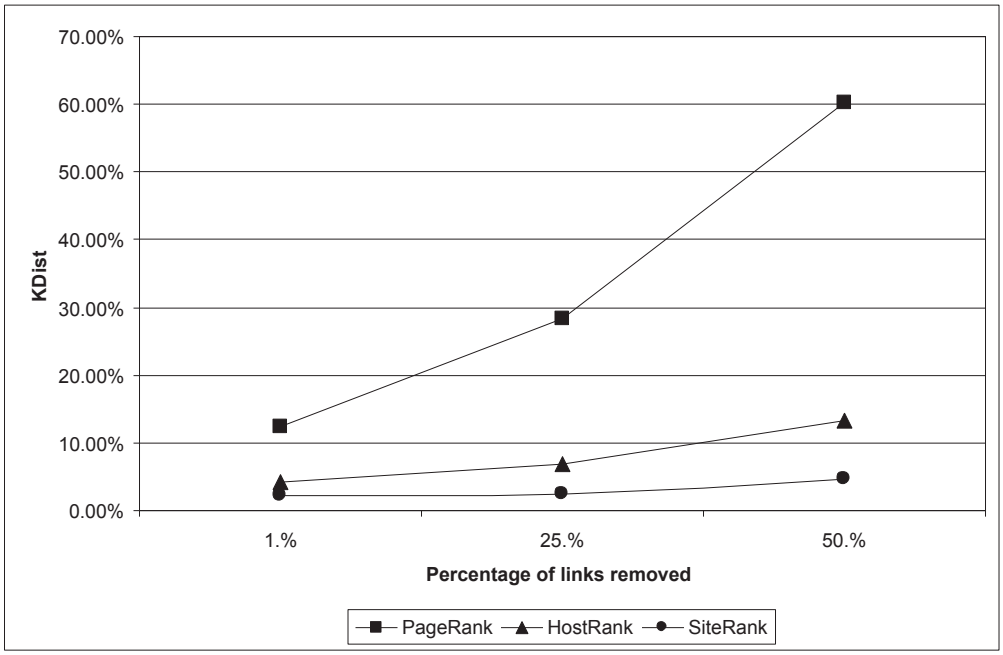

Figure 2.2: KDist over pairs of new pages, complete graph vs. reduced links.
Data in Table A.2.

Figure 2.2 restricts our comparison to pairs of new pages. The lack of links eliminates the only signal for page ranks in PageRank, so pages with fewest links are affected the most. On the other hand, site ranks are more reliable than page ranks when some links are missing, and most new pages' importance is mainly determined by the site they get added to, since they haven't accumulated many inlinks yet, so HostRank and SiteRank predict new pages' rank better than PageRank. In fact, both HostRank and SiteRank are weakest at maintaining the relative order of established (non-new) pages: these pages have collected enough inlinks that their rank is well-defined by the links pointing to them; no approximation can perfectly guess the same relative order among all old pages.

The best way to check how well we eliminate the bias against new pages is by looking at the relative order of page pairs where one page is new, and the other old. The results are summarized in figure 2.3, and they show HostRank and SiteRank strongly outperforming PageRank. While not the best solution for ordering just the old pages, siterank-based page ranks are a good approach to level the field - it gives new pages some authority from their host/site, and it removes some of the exceedingly strong authority accumulated by old pages.

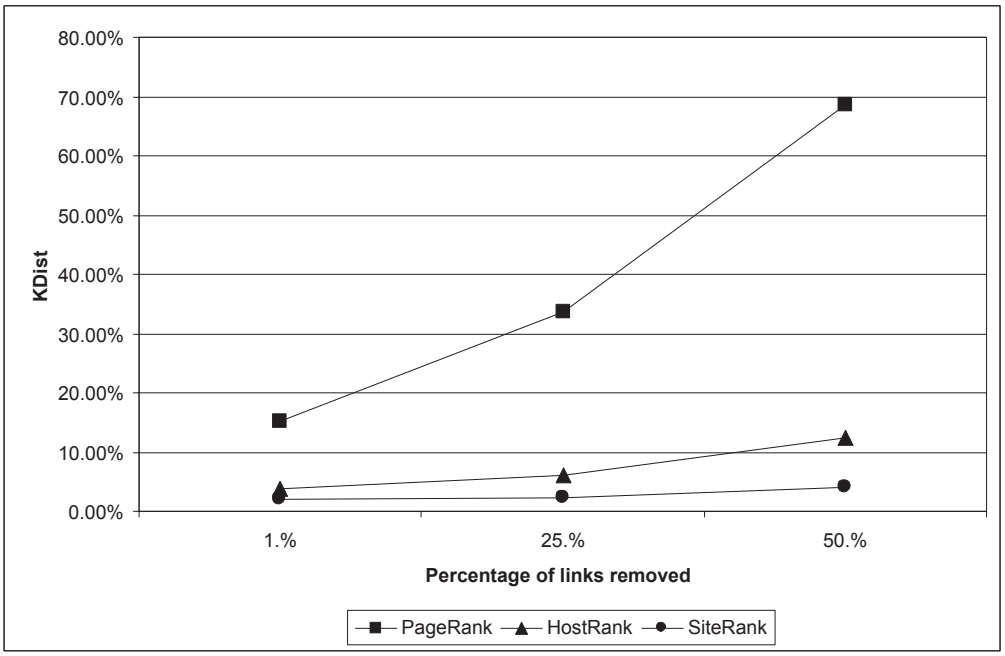

Figure 2.3: KDist over pairs of (new,old) pages, complete graph vs. reduced links.
Data in Table A.3.

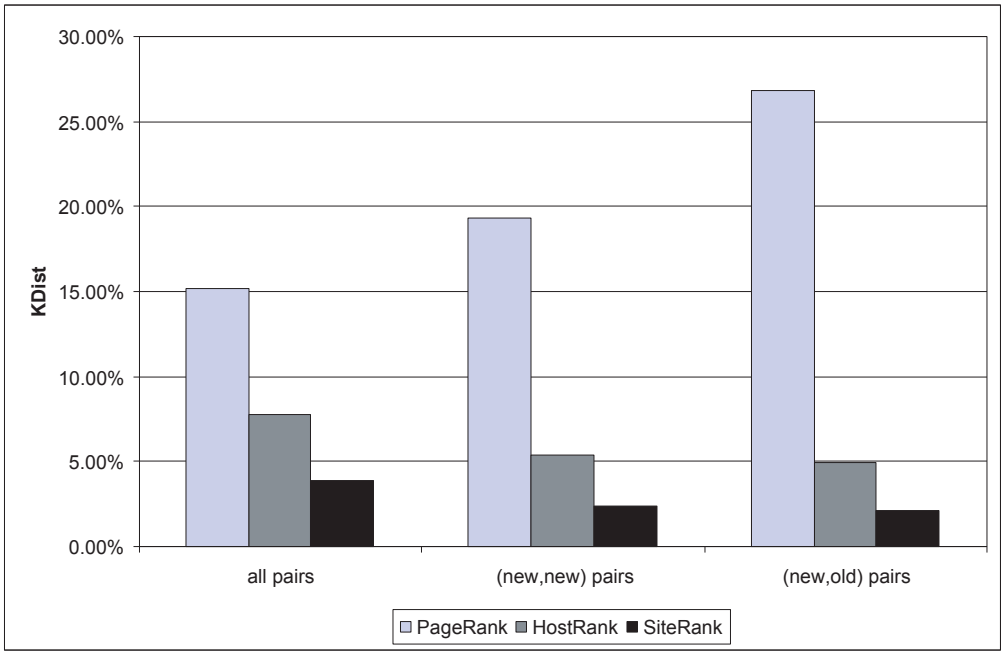

Figure 2.4: KDist over all page pairs, two consecutive crawls.
Data in Table A.4.

All these initial experiments were based on a single web crawl of 11 million documents. We built synthetic data sets that mimic how new pages gain links over time; however close statistically to real link graph evolution, this approach is still a crude approximation. Fortunately we were able to repeat these experiments on two consecutive 90-100 million document web crawls from the Stanford WebBase project [30], collected one month apart (September and October 2005; also confirmed with data from March and April 2007). About 10% of the links have changed between the two crawls, approximately 5% because of new links being added and 5% because the page sets in the two crawls don't overlap perfectly. In this setup, while all the pages were used to compute page ranks, only the intersection of the two page sets was considered when comparing the ordered lists. The 5% pages that acquired new links were considered new pages – the vast majority (more than 90%) of the links were added to pages that had very few links in the first of the two crawls. The results are illustrated in figure 2.4, and they confirm the earlier results collected on the synthetically aged single crawl: PageRank is the weakest predictor of future link authority, and both HostRank and SiteRank handle new pages better than old pages.

2.3.2 Page order in SiteRank and PageRank

The previous section proved that SiteRank is significantly better than PageRank in predicting the link authority of new pages. We also tested the relative accuracy of SiteRank compared to PageRank, to measure how close the new order is to the widely-used PageRank-based page ordering. This experiment is meant to gauge how easily can SiteRank be used as a replacement for PageRank in an existing search system without disrupting the current ranking order significantly. While the previous experiments were measuring how well SiteRank matches future values of PageRank, this experiment compares PageRank and SiteRank order on the same link graph.

Since results were repeated and averaged over multiple runs, we only computed KDist over random samples of all the page pairs:

- a different uniformly random selection of 10% of all page pairs in each separate run, where each page is randomly selected from the document set (random);
- a different uniformly random selection of 10% of all page pairs in each separate run, where the first page was randomly selected from the document set and the second page was randomly selected from the same site as the first page (on the same site);

- and all the page pairs where each page is the index page of a site (index pages).

Figure 2.5 summarizes a number of interesting results:

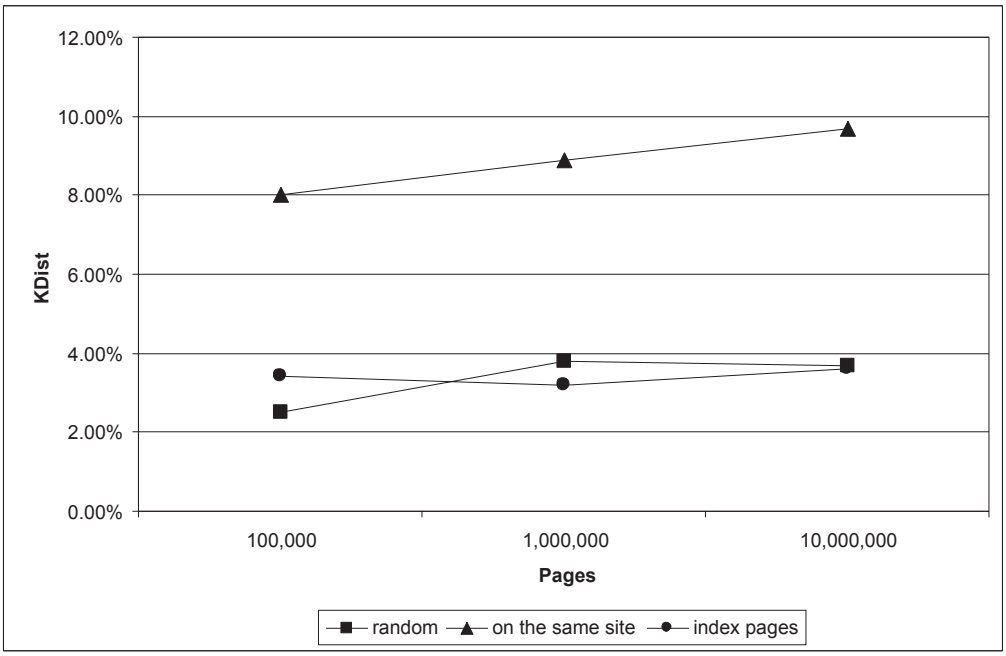

Figure 2.5: KDist of SiteRank vs. PageRank page order change.
Data in Table A.5.

- Overall, less than 4% of the page pairs have changed order from PageRank to SiteRank (random page pairs, i.e. page pairs where each page is randomly selected from the document set). There is no increase when going from 1 million to 10 million pages; not included in this graph, the same experiment over the 100 million page data set shows no significant increase, suggesting that KDist tops off at less than 4%.

- We looked separately of page pairs from the same site - their relative rank order is determined by page position and individual inlinks for each page. KDist over these pages is significantly higher than the number computed over all pages, suggesting that our heuristics for individual page rank computation starting from site ranks introduce most of the rank ordering changes.

- Restricting KDist to index page pairs (e.g. *www.nytimes.com/index.html*) keeps it close to (and lower than) the number computed over all pages. This indicates that SiteRank preserves most of the classic PageRank algorithm's ordering of the sites.

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 2.1. 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 relative to each other and the rest of the pages. The individual rank order changes for top pages are much smaller than for an average page, and the page pairs are much less likely to switch order (1.3% for $top - 100$ pages compared to 3.7% for all pages). In these experiments, the top-k is defined based on the PageRank order; if any of these pages slips out of the top-k in SiteRank order, it is considered a loss and every corresponding pair is considered flipped between PageRank and SiteRank.

k	**KDist**
10	1.0%
100	1.3%
1,000	1.5%
10,000	1.6%
All pages	3.7%

Table 2.1: Changes in rank order for top-k pages.

2.3.3 Performance

As described at the beginning of section 2.3, the basic implementation of traditional PageRank we used as baseline was also used in other projects [31, 32, 62, 64]. Since it has no optimizations for speed or accuracy, it serves well as baseline when comparing one's results against other work in this area. It is easy to improve on the basic version, but not all proposed improvements are addressing the same issues so they can be combined for even better performance. Many of the improvements proposed by other groups in section 2.1.2 (e.g. [27, 31]) can be used in SiteRank too.

To eliminate variations due to exact hardware used (computation power, cache sizes, speed to access memory and secondary storage, etc.) we counted the actual number of operations (rank lookups and arithmetic operations) performed by each algorithm.

While this was not the main original motivation for SiteRank, during our experimentation it became clear that SiteRank is much faster than the classic PageRank algorithm (as it was expected given that it runs the iterative algorithm on a much smaller link graph). Section 2.1.2 listed some attempts at speeding

up PageRank, but the result is usually a small constant speedup factor (up to 4 [32]). Figure 2.6 shows that our approach achieves a much higher speedup factor, and this number is actually determined by the average site size. When randomly choosing pages from the webcrawl, the resulting subsets have different average site sizes (number of pages), and we noticed that the speedup over page sets of various sizes does not change significantly if the average site size is the same.

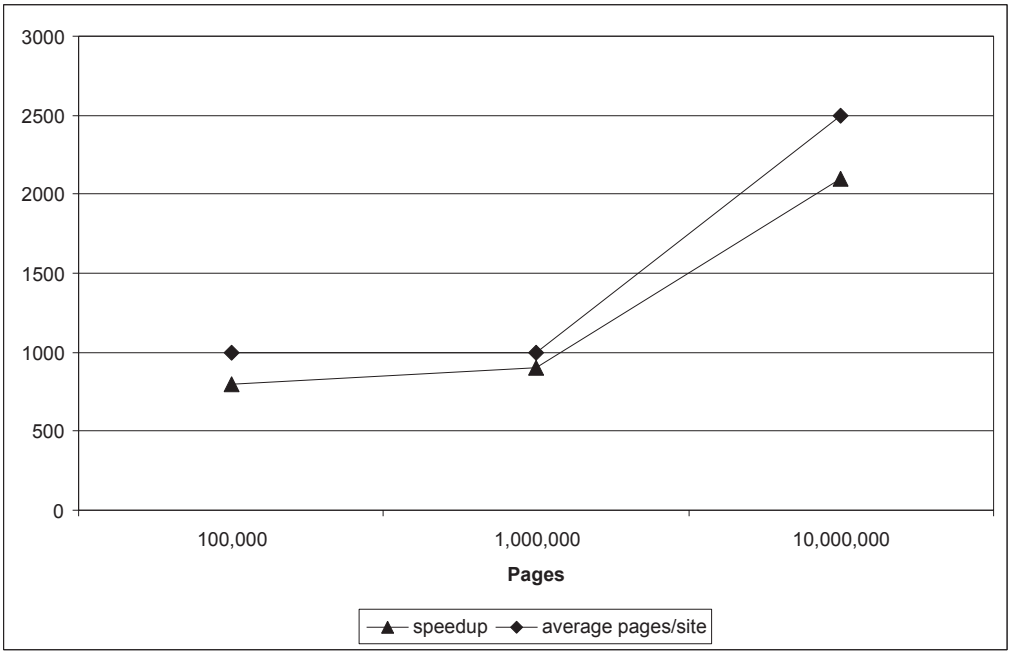

Figure 2.6: SiteRank vs PageRank speedup factor.
Data in Table A.6.

This is seen even better when only the average site size is changed while keeping 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, as summarized in figure 2.7. Higher average site sizes were not available in the data sets we used, and restricting the data sets artificially to contain only large sites would skew the results considerably from the normal document distribution.

Even though the main motivation for SiteRank is improving the rank computation of fresh pages, the results show that this algorithm might be a much lower cost and still accurate replacement for PageRank in general for regular or persistent search. As noted earlier in section 2.2.1, reducing the web graph for the iterative computation reduces both the duration of each iteration, and the number of iterations needed to converge. On the 10 million page data set, the number of iterations was reduced from 30 in PageRank

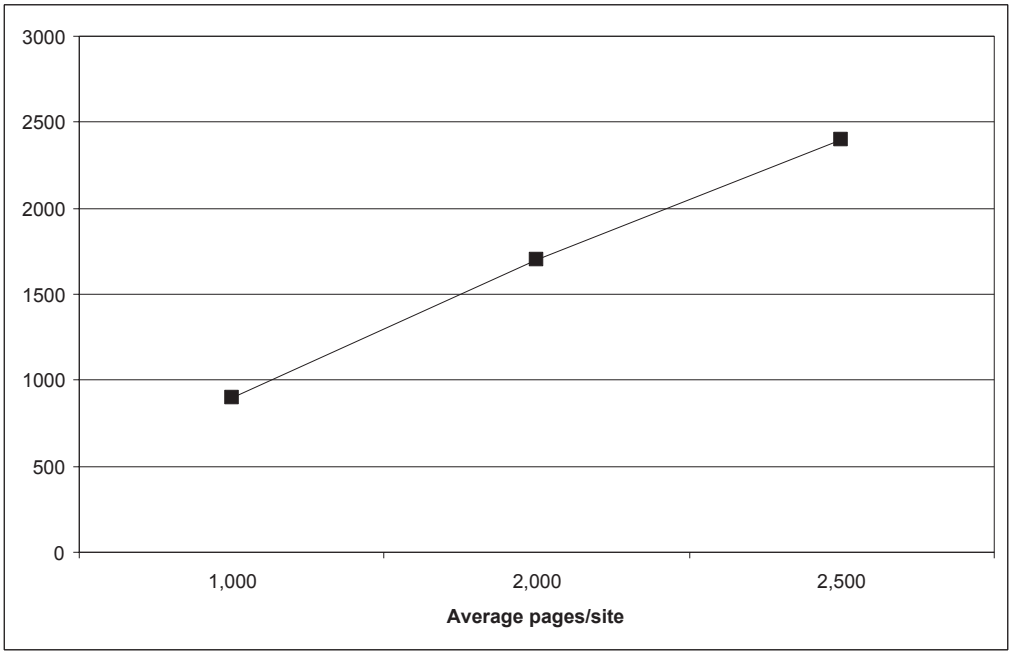

Figure 2.7: SiteRank vs PageRank speedup factor on 1 million-page data sets, varying average site size. Data in Table A.7.

to 20 in SiteRank. This allows rank scores to be recomputed more often, a welcome option in a system processing a high volume of new publications.

2.3.4 Predictive SiteRank

Because of the additional storage and computational cost to match titles of new pages against the content of existing pages (see section 2.2.4), we decided to run Predictive SiteRank experiments separately as an exercise in detecting possible improvements.

Virtual links based on new pages' topic/title can lower KDist to about $50 - 60\%$ of its previous value, but the computational cost is prohibitive - on the 1 million-page set, the running time grows many orders of magnitude higher than its regular value, mainly because document content could not be stored in memory. As noted earlier, limiting the amount of stored content to match against new pages' titles to titles and anchors reduced both the storage and computation costs, and the impact of the virtual links to a point where this additional expense was not justified.

Independently, we tried promoting new pages' ranks at the end of the computation, using an exponentially decaying boost value based on the age of the pages. Surprisingly, this only improved the results by a

relative 7% over the basic version of SiteRank, and was completely ineffective when combined with virtual links. In retrospect, this additional rank adjustment promotes new pages without using any authority signals and cannot be used to usefully differentiate between new pages, or between new and old pages (e.g. new pages on low-quality sites received a boost that was never fulfilled by later links).

2.4 Summary

We proposed and developed SiteRank, a novel link-based relevance algorithm that predicts the popularity of new pages better than existing approaches.

It accomplishes this by collapsing the link graph to site granularity, and using the computed site ranks to determine individual page ranks. Making use of the pages' position on sites and their inlinks allows new pages on high-quality sites, referred by high-quality sources to out-rank less-popular old pages that have collected multiple inlinks over a long time.

Tests on synthetic data sets and consecutive web crawls prove that this approach outperforms existing approaches in predicting the popularity of new pages. However, SiteRank loses a little accuracy in ranking old pages compared to SiteRank, and is therefore not a good replacement if exactly maintaining the existing ordering is important. For all other traditional and persistent search systems, SiteRank is better alternative to PageRank. Additionally, it reduces the rank computation time significantly, allowing for more often re-computation to keep page ranks up to date.

Chapter 3

Distributed Persistent Search

Citation-based authority computation is only one component of a persistent search system (see section 1.1). Building a large-scale distributed persistent search system raises a number of additional challenges that need to be addressed in order to make a system practical. Some of these challenges 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.
- **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; this component can easily be enhanced if required by more expressive subscriptions.
- **Notifications:** This component depends heavily on the distribution scheme used, the main goal being to reduce internal and external 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 maintenance cost.
- **Citation-based authority:** This component was discussed in chapter 2. We also refer to it as link relevance computation.

- **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 publish-subscribe server.

- **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.

Of the challenges listed above, our contributions address the distribution of servers and documents and citation-based authority. For all other challenges, we use fairly basic existing solutions. Our contributions can be summarized in two major directions that are currently poorly addressed in persistent search systems:

- How can we minimize the publication and notification latency? For a persistent search system to be indeed useful to its subscribers, it needs to serve fresh documents fast.

- How can we increase the publication crawling rate? While low publication latency is a requirement for high refresh rate, we also need to speed up some other components (e.g. link relevance computation) in order to make the entire document processing pipeline fast enough for frequent refresh cycles.

When addressing large-scale general-purpose persistent search, the publication base consists of publicly accessible pages on the web (html, text, processed PDF and PostScript files). Similarly, user subscriptions in such a persistent search system consist of keyword-based queries. The only Boolean operation we implemented is AND of keywords, i.e. all keywords in a query must be matched for a document to match the subscription. Other operations (OR, NOT, optional keywords) are not the focus of our work; since the matching operation can be completely isolated from the other components, this does not affect the overall design.

This chapter compares single-server and distributed architectures. In each of these, any single location (node or server) can consist of one of many machines on the same network; whenever we refer to a server or location in a single-server or distributed system, we are not restricting that server to a single machine, but to potentially an entire local cluster. Some problems can be solved by simply using more machines,

but as we will show our particular problem of building a large-scale search system is best solved by a distributed system which distributes the content across multiple locations and therefore increases bandwidth and shortens communication paths.

After a short overview of existing systems, we give a high-level description of our design and use experimental results to prove that this design outperforms existing systems when utilized for large-scale general persistent search over web documents. We conclude the chapter with a short summary.

3.1 Related work

3.1.1 Persistent search

Highly specialized services (such as financial data or news [14, 40]) use active and passive publishers (see section 1.1) to cover a relatively small and highly specific publication base. Real-time coverage is therefore possible, and clients are quickly informed of new or updated data. 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 3.1.

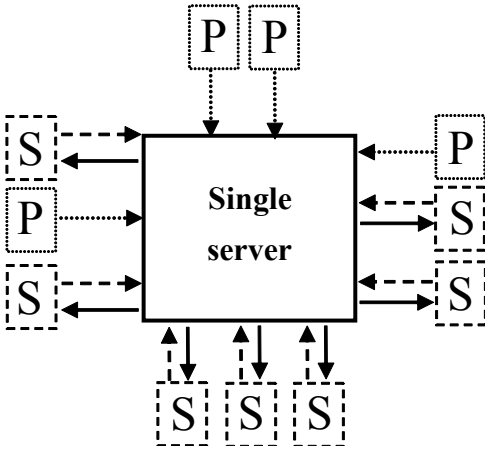

Figure 3.1: Single-server architecture (1S).

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 subscribers, or the size and/or the update frequency of the publication base grows significantly. Earlier estimates on the size of the web [49] and search engine

coverage [53] 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. The surface web is what is traditionally discovered, indexed and served by search engines – static web pages with links from other pages. The search engines can't easily see the deep web, since it consists of pages that are created dynamically by a specific lookup or search; this information is stored in databases that are not directly accessible from outside their sites, but the information is available to the user on request (e.g. the price of a book on an online store's site). This data would expand both the size and the growth rate of the web by an estimated factor of 500 [3], considerably widening the gap between the available publication base and the search engines' coverage.

Existing search engines do not fare well when it comes to data refresh rate, either. On average, the bulk of most search engine databases is about one month old [53]. 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 2006 estimates can show us the magnitude of the problem. Recent estimates put the size of the indexable web over 11.5 billion pages [26], 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 [49]. 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 [3], 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 bandwidth of 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.

The numbers in the preceding two paragraphs are relatively old, but they serve well to illustrate the problems faced by large-scale search engines. While there is no significant research going on to estimate the size of the web, its growth rate and major search engine refresh rates, some notable recent announcements give strong support to the earlier estimates:

- In July 2008, Google [20] announced that it is indexing 1 trillion URLs (not pages) [4].

- Soon after this, new search engine Cuil [15] was launched, claiming to index and serve 120 billion web pages [44, 57].

Both coverage and refresh rate in these systems can be improved through parallelization or distribution of crawling. Even during the early days of Google, the web crawling was done by multiple distributed crawlers [7] since parallel crawling is limited by bandwidth available at a single location. Search engines of various sizes are currently running as distributed crawlers coordinated from a central control location [20, 25, 65]. In this architecture, a central server/location assigns crawling to other servers, which then send the content back to the central control. This architecture (*DCC*) is illustrated in Figure 3.2:

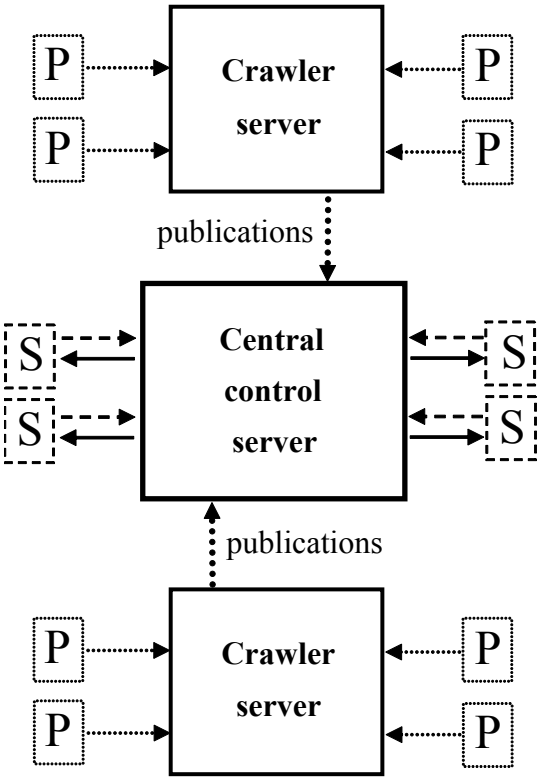

Figure 3.2: Distributed architecture with central control server (DCC).

This approach effectively distributes the crawling load and increases the crawling-based refresh rate, but it fails to reduce the publication traffic to a central server. Also, all subscription handling, matching and notification is done at the central server. This architecture is currently used by persistent search services deployed by traditional search engines [21, 66]. However, infrequent updates of all but the most popular pages on the web make the user-percieved latency between the publication of a new page and the receipt

of a notification to be measured in days, or even weeks for less-popular pages. Dealing with the large volume and freshness requirements of a web-scale persistent search engine requires a more fully distributed architecture.

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.

- Structured peer-to-peer systems based on distributed hash tables (DHT) scale better than unstructured peer-to-peer systems and can offer search guarantees [43, 46, 55, 69]. Workload and performance estimates [37] suggest that very large systems built on these architectures might be used for distributed search, but their estimates of the size and growth rate of the web are a couple of magnitudes lower than the already optimistic numbers presented earlier in this section.

  Also, dynamic peer sets in a structured system could easily make the overlay network too expensive to maintain.

- Unstructured peer-to-peer architectures [59, 33] are not usable due to their lack of guarantees and poor scalability [37].

As an example of distributed search over a structured DHT architecture, Tang and Xu [56] propose pFilter; a global-scale persistent search system, it uses a structured DHT [63] to store queries and efficiently route publications to related queries.

- Both documents and queries are routed through the overlay (a hierarchical version of CAN [43, 63]) based on their semantics. Document semantics are derived using the vector space model (VSM) and latent semantic indexing (LSI). While this leads to an efficient matching algorithm that eliminates some problems of literal matching schemes, it still relies on storage and routing in CAN's cartesian space which has very poor performance if the number of dimensions is large. Trying to index the large and diverse document space served by a web-scale search system cannot be done in a small dimensional space.

- Subscriptions are aggregated for more efficient routing, building a single multicast tree for similar queries. This might lead to many false positives as unnecessary notification messages are sent. Even with this optimization, registering one static application-level multicast tree for each subscription (or subscription cluster) cannot scale well when the number of users is counted in the millions.

- Documents are not processed at their initial server, they are routed based on their CAN coordinates. Due to the relatively large size and publication rate of web pages, the inter-server publication traffic can prove very expensive.

- The lack of any additional document quality assessment further increases non-relevant notification traffic.

These important issues have not been addressed properly by pFilter, and no proper evaluation of the system has been included in this or any related publication. This further underlines the difficulty of operating a general-purpose web search system on peer-to-peer architectures.

To date, the only proven approaches to do persistent search at web-scale are based on the distributed architecture with central control (DCC). We will evaluate both our proposed solution and DCC against the single server (1S) architecture to objectively determine how well each of these scales with the number of servers and the number of documents in the system.

3.1.2 Distributed publish-subscribe systems

A high-level overview of publish-subscribe architectures has been given in section 1.1. Distributed publish-subscribe architectures have been proposed for structured and unstructured persistent search. The three major approaches are content-based forwarding, best illustrated by SIENA [1, 10, 11, 16]; channelization [19, 45]; and dynamic multicast [9].

Content-based forwarding

In content-based forwarding (CBF) [1, 10, 11, 16], the network of servers is organized in an acyclic overlay. Figure 3.3 presents a simple server architecture in SIENA.

Users can connect to any server on the tree, and their subscriptions are broadcast from the originating server to all its neighbors. Each server builds filters for each neighbor that summarize all subscriptions coming from that direction. These filters are propagated in all directions, and progressively assembled at each server. In Figure 3.4, filter X published at node 1 and filter Y published at node 2 are assembled into filter $X - Y$ which is propagated to other nodes, e.g. node 5 which will store filter/subscription $X - Y$ from node 4.

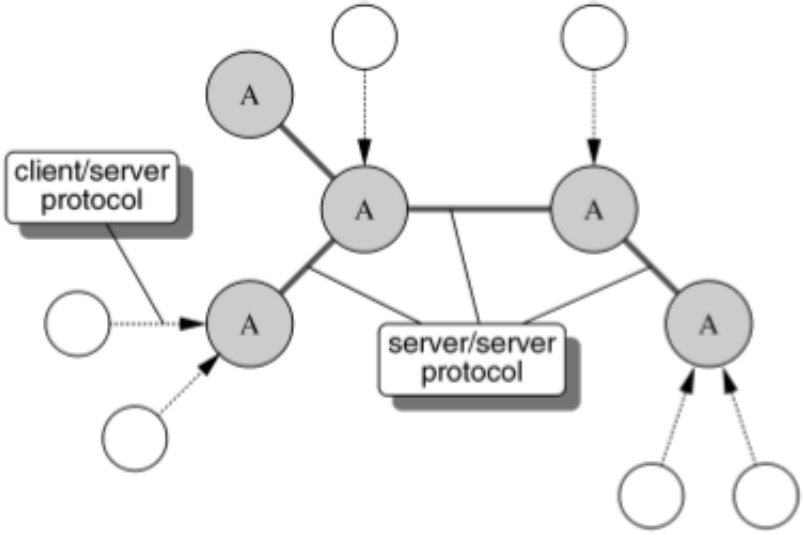

Figure 3.3: Content-based forwarding: Acyclic server architecture.
Source [10].

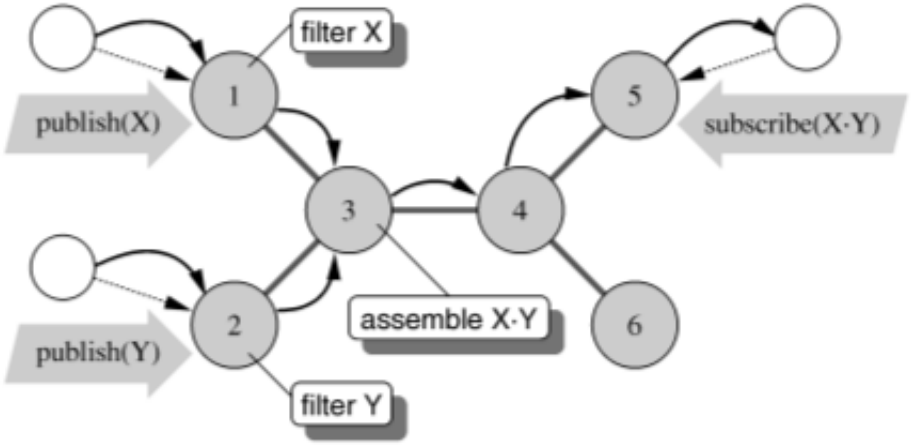

Figure 3.4: Content-based forwarding: Subscription propagation.
Source [10].

Documents are matched at each server against all the forwarding tables (filters), and forwarded to neighboring servers that have matching subscriptions. In Figure 3.5, a document published at node 1 that matches subscriptions from nodes 5 and 6 is routed through filtering tables in nodes 3 and 4. Node 4 replicates the notification and the corresponding document and sends them to nodes 5 and 6.

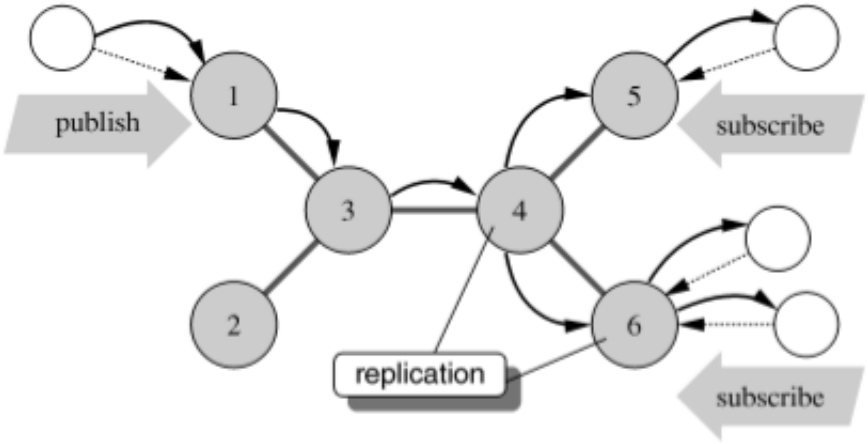

Figure 3.5: Content-based forwarding: Notification routing.
Source [10].

The progressive filter-based matching in SIENA forwards each document only towards nodes that contain subscriptions matched by the document, but the path goes through nodes that might not be interested in this particular publication. Additionally, each matching operation is expensive - while the complexity of matching is determined by the specific application running on this architecture, it might include text search, range queries, complex Boolean expressions against a potentially large set of subscriptions, at each node in the system.

Channelization

Channelization [19, 45] partitions the publication space into a fixed, limited number of channels, with one multicast tree built for each channel (see Figure 3.6). Each information provider can publish documents in any channel, and each user can subscribe to any channel.

The multicast tree corresponding to a channel connects all the servers containing subscriptions that might be matched by publications from that channel. The mapping of document subsets to channels is replicated at all nodes in the system and each server uses this information to determine which channel

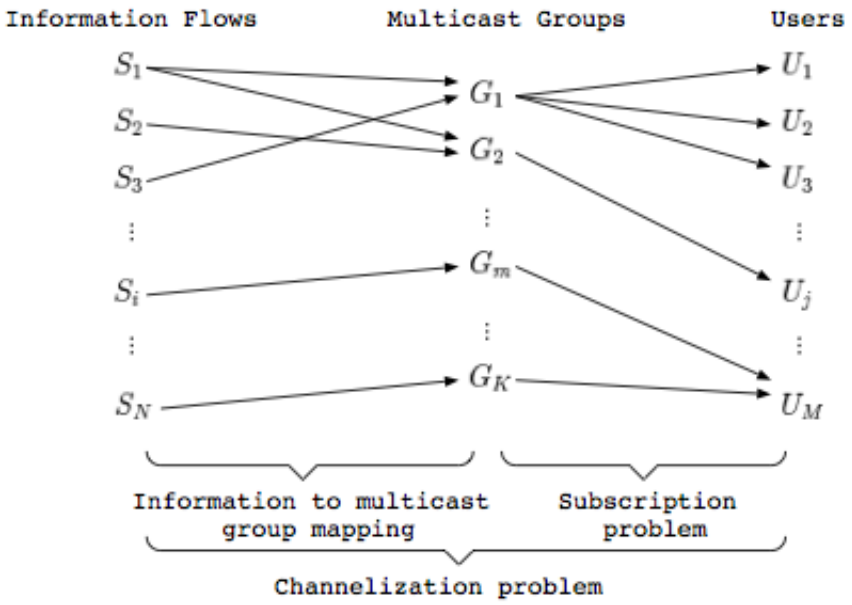

Figure 3.6: Channelization: Multicast groups mapping publishers (information flows) to subscribers (users).

Source [19].

a newly published document belongs to. Upon receiving a new document, it looks up its corresponding channel and sends a notification to the closest server in the corresponding tree, which then distributes it to all other servers in the matched multicast tree. Only a single lookup in the partition map is required, since the document carries its channel ID after the initial match.

While more efficient that content-based forwarding, channelization is severely limited by the relatively small number of available multicast channels. This approach works well in topic-based persistent search systems; however, since the possible number of keyword-based subscriptions is much larger than any practically possible number of channels, each channel can cover a very diverse subset of subscriptions and publications. This requires high communication costs for each channel, as many unrelated subscriptions share the same channel and each node can receive many publications that don't match any of its subscriptions.

Dynamic multicast

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.

Similar to content-based forwarding, MEDYM (Match Early with DYnamic Multicast) [9] determines the exact destination servers for each publication by matching its content against subscriptions. Similar to channelization, MEDYM only matches each publication once, at the first server that receives it from the publisher. The content-based matching is removed from the routing component, and the publication is sent only to nodes containing matching subscriptions. The disadvantage is that subscriptions must be broadcast to all the servers. However, most applications have a much larger and more dynamic publication base than their subscription base; this is especially true in the case of web-scale persistent search.

After determining the full destination list by matching the new document against subscriptions, MEDYM forwards the notification based on the addresses of the destination servers, without the need for further content-based matching. At each step, dynamic multicast routing uses current network conditions to determine where to forward the event in the remaining list of destinations. Figure 3.7 illustrates an example where event 2 published at node A needs to be delivered to nodes C, E and H.

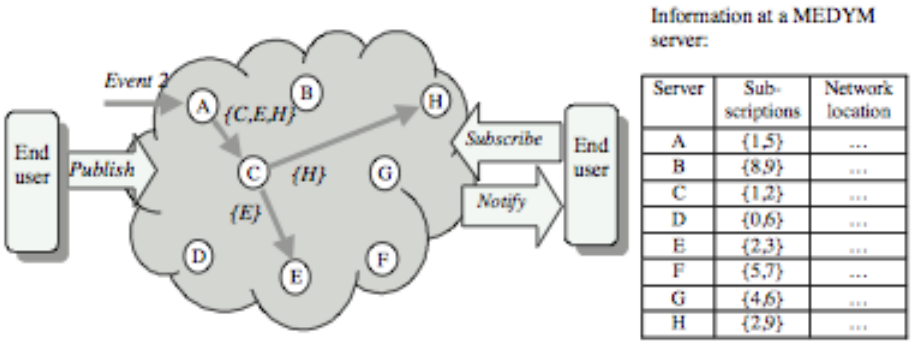

Figure 3.7: Dynamic multicast: MEDYM network and destination lists.
Source [9].

Dynamic multicast achieves high network efficiency with low overhead. The system's reliability is improved by distributed decision-making when routing each notification to its destination list, as each node uses its information about local network connections to determine the best next-hop. For large-scale content-based publish-subscribe, MEDYM has proven to be a very good event routing solution, as it eliminates the need for multiple matching operations without introducing false positives. Even at the cost of replicating subscriptions at all nodes, the MEDYM architecture is very well suited to our problem because publications are much larger and more dynamic than subscriptions.

3.2 DPS system design overview

The main focus of our work has been computing accurate link-based relevance for web documents in a persistent search system. While we can evaluate ranking accuracy in stand-alone tests (section 2.3 summarizes these experiments), the performance improvement in SiteRank can best be evaluated in a complete persistent search system since it should also take into consideration the communication between ranking and the other components. To address the challenges listed at the beginning of this chapter, we designed and built the Distributed Persistent Search (DPS) system.

3.2.1 Server distribution

A high-level view of DPS is illustrated in Figure 3.8. The defining characteristic of an internet-scale, keyword-based, general persistent search system over web documents is that publications are much larger and more dynamic than subscriptions. The performance of the system is then mainly determined by the publication traffic.

The main goal of the distributed architecture is to minimize publication latency by eliminating publication traffic (inside and outside the system) 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. This non-overlapping partitioning of the publication set achieves the shortest average publication path.

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. Both methods incur the same network cost to route the published documents to the processing servers, but crawling also adds the cost of periodically checking if documents have been updated, or new documents have been added. Determining when and what to crawl is a significant problem by itself and it is not addressed in our work, so in our evaluation we model publication simply by routing new documents to the processing servers over the shortest available path.

3.2.2 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

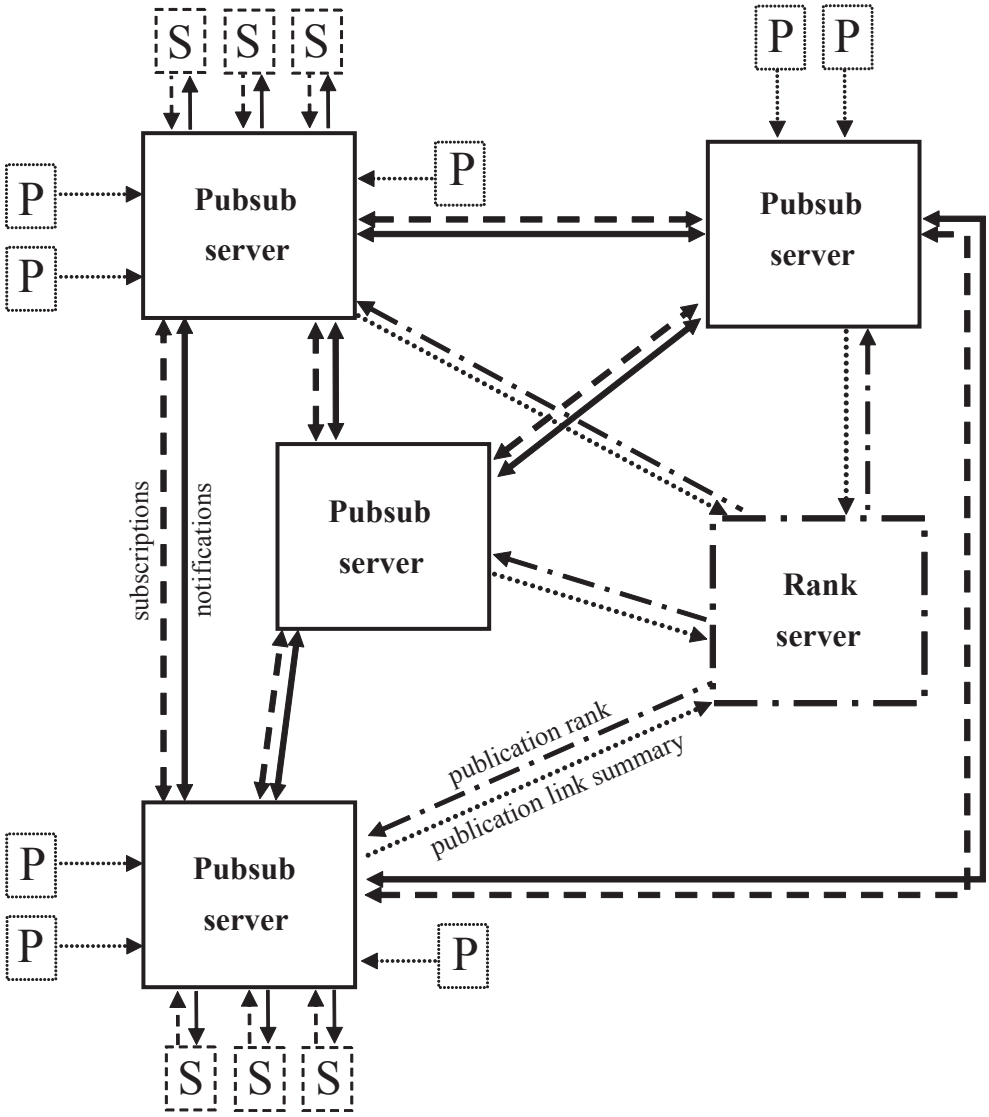

Figure 3.8: Distributed Persistent Search.

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 servers in the system; these *remote subscriptions* (subscriptions replicated from other servers) maintain a reference 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. While including this information with each copy of the subscription would allow the original matching server to directly identify the owners of matched subscriptions, this approach presents a number of drawbacks:

- Anonymizing subscriptions allows us to replicate a single copy of a query that is shared between multiple subscribers connected at the same server. Publish-subscribe servers maintain a map from each individual query to the subscriptions containing that query.

- If a publish-subscribe server notifies its local and remote matched subscribers, it is not taking advantage of the dedicated network resources between the distributed system's servers. A more efficient approach is to send a single notification to the originating server, which then notifies all the corresponding local matched subscribers individually. Since their subscriptions are managed by this server, all subscribers are geographically close and the individual notifications are sent through the shortest path outside the network. While a search system might have great control of the connections between its servers (and can therefore use high-speed connections), it has no control over the users' connection to the system and is therefore wise to shorten any notification path that lies outside the system.

3.2.3 Publication management

Subscriptions (sequences of keywords) and publications (text documents) are managed by the same servers – each server handles both subscriptions and publications; we will refer to them as *publish-subscribe servers*.

Publications are processed at the incoming server and matched against all subscriptions. A publication that matches all the keywords in a subscription is said to match the subscription. As mentioned at the beginning of this chapter, we do not implement additional Boolean operators or advanced matching logic (e.g. OR, NOT, optional keywords).

A dedicated rank server collects all information necessary to compute and update link-based relevance (see section 3.2.4 below). Each publish-subscribe server sends update messages to the rank server, containing the outlink summary of every new or updated page it processes. The rank server replies with the rank value of the new/updated page.

3.2.4 Publication relevance and quality assessment

The overall relevance score of a publication for a given query is computed as the product of two numbers: the query-independent link authority of the publication (PageRank, SiteRank) and a content-base topicality IR score.

Building on a distributed publish-subscribe framework introduces a major problem by removing the single publication repository that can be used for quick link authority computations. Chapter 2 summarizes SiteRank, our proposal for a more accurate and faster link relevance computation; however, even this method requires that all link information is collected at a single location.

We address this limitation first by reducing the amount of information we collect at the dedicated *rank server* - to build and maintain the link graph we only need link information from every page. Link summaries are sent to this dedicated server from the document processing servers 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 site-rank and other information (see section 2.2.2) to compute an authority value for the page; this value is sent back to the publish-subscribe server handling the new or updated document.

To reduce the time needed to compute an individual page's rank, site ranks are not computed at each link update. Instead, an initial site-rank computation is performed at system startup, using all available pages from an initial crawl. When a publish-subscribe server inquires for the rank of a new or modified page, the link graph is update with the page's individual links, but its page rank is computed based on the existing site rank values. When a large enough portion of the links are modified, site-rank values are recomputed asynchronously in the background by the rank server. Choosing the right frequency of this site rank computation is a trade-off between the computation power of the rank server and the desired

freshness of the rank values. In our experiments, we recomputed the site rank values whenever at least 5% of the global page links have been added or updated, or when 10% of any single site's links were added or updated.

This single-rank-server approach presents a single point of failure in an otherwise fully distributed system, and also scales relatively poorly when the publication base is very large. We have therefore also explored a distributed version of SiteRank, summarized in chapter 4.

The overall relevance score of a publication also takes into consideration content-based topicality, which we compute as the TF-IDF score [47] between the publication and the matched subscription (inverse document frequency is computed from the local information at each publish-subscribe server). Judging content-based relevance is an interesting topic that is beyond the scope of our work.

3.2.5 Matching and notification

Matching of each publication against all subscriptions is done at the publication's entry server. 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 containing similar queries are aggregated before propagation to other servers – multiple users' subscriptions might contain similar popular queries and we replicate a single aggregated copy in this case. This is the narrowest possible query that covers all the similar user queries. E.g. "persistent search" and "distributed persistent search" would be aggregated as "persistent search". It is not required that one of the users' queries is the aggregated version, e.g. "distributed search" and "persistent search" can be aggregated as "search", although a publication matching this aggregated query might not match any of the original queries, since it might lack all the additional terms.

All subscriptions are retained individually at the original server for local matching before sending notifications to individual subscribers. After the subscription is matched at any remote server, we need to find all local subscriptions that are matched by the new publication. This approach therefore requires that the matched publication is sent together with the inter-server notification, and a second matching operation is performed at the aggregate subscription's originating server. While subscription replication is less expen-

sive due to aggregates, the same mechanism can generate possible false duplicates that require unnecessary inter-server notification messages and publication traffic, if subscription aggregation is to aggressive.

In DPS we use MEDYM's dynamic notification, but only aggregate subscriptions that contain the same underlying query, as mentioned in Section 3.2.2. To eliminate the need for a second matching operation, all aggregated subscriptions are mapped to the common query and are therefore easily retrieved if the notification from the remote server contains the matched query. This approach eliminates the need for the inter-server notification to contain the matched publication's full content.

More aggressive aggregates and other optimizations can be added to this mechanism, the study of their benefits and costs is not part of our work.

3.2.6 Duplicate detection

DPS detects exact duplicates locally at document entry points and globally 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 storage cost, since the publication space is partitioned among the servers. This solution detects duplicate publications that are handled by the same publish-subscribe server.

- We would also like to detect distributed duplicates (usually due to site mirroring) and not notify a user multiple times if the same publication is received by two or more different publish-subscribe servers and remotely matches one of the user's subscriptions. For this, additional publication metadata is needed – e.g. the matched publication's URL is already sent so the user can look up the document that matched the subscription; other small pieces of information we used are separate checksums computed over the publication's content and its outlink list, and the publication's link-based rank value.

  Each publish-subscribe server caches the metadata associated with recent notifications (last day) and cancels any duplicate notifications for the same user – each notification is checked against the user's cached data for possible duplicates.

In our data sets, only approximately 0.72% of the publications were mirrored and triggered the remote duplicate detection safeguards. Since our experiments were not long-lived (all documents are published into the system sequentially, without break), all of these duplicates were detected since none of the meta-data expired. Overall, this constitutes a minuscule impact, but it is always significant for the affected subscriptions.

3.3 Evaluation

To evaluate the performance of the Distributed Persistent Search system, we implemented an event-driven, message-level simulator on top of a random network topology. We used the GT-ITM random generator with the transit-stub model [68]: 25 transit domains contain an average of 8 routers each; an average of 5 stub domains are attached to each transit router, and each stub domain contains an average of 10 routers. This added up to more than $10,000$ routers and approximately $45,000$ links. Publish-subscribe and ranking servers were randomly 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 in the underlying network.
- For each of DCC – distributed architecture with central control (see Section 3.1.1) and DPS – distributed persistent search (see Section 3.2), server nodes were randomly chosen from the most-connected nodes in the simulated network, uniformly distributing them across transit domains to simulate geographic separation.

  It is safe to assume that such systems would be deployed with very high connectivity, so additional links were added from the servers to their corresponding transit routers.

  - DCC uses k crawler servers and an additional central control server.
  - DPS uses k publish-subscribe 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. Realistically, it would not be practical to deploy and maintain a distributed system with more than $1,000$ nodes (locations).

- Multiple runs (100) of the experiments were averaged, each run using a different assignment of servers to nodes. Additionally, a few runs were executed on different random topologies (generated using the same model). Due to the uniform distribution across transit domains and the high connectivity, all runs were very similar – the largest observed standard deviation across all kinds of experiments was 4% of the corresponding mean value.

To test the overall system performance, we compared our DPS system with 1S and DCC, measuring publication, notification and subscription management costs. The cost values for DPS and DCC were normalized by the values for 1S, so our summary presents the relative improvement of DPS and DCC over 1S.

- For a persistent search system to serve fresh results, it is important that it can re-crawl the publisher sites often. It can do so reliably if page accesses and transfers are fast. Publication crawling path length is a good indicator of crawling latency - the more routers we need to go through, the longer each request takes. For each document, we measured the crawling path length from the publisher's node to the server processing that document.

  We also checked the load distribution among the servers in the distributed architectures by counting the number of pages handled by each server.

- Similarly, an important feature of a persistent search system is the ability to quickly deliver notifications if a publication matches existing subscriptions. It can do so if it uses network resources efficiently and the notification paths are short. 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 average total number of message hops for a notification, including both internal (inter-server) and external (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, 100 million publications from a June 2005 WebBase webcrawl [30] were used. Publication sites were assigned to random non-server nodes in the network, then documents were routed to the publish-subscribe/crawler server closest to the publishing node (based on number of hops in

the underlying network). In the distributed systems, the destination publish-subscribe server was always searched hierarchically – first within the same stub domain as the publisher, then in the same transit domain, then globally. Since a large number of sites of various sizes were uniformly assigned to all transit domains in the network and publish-subscribe servers were also uniformly assigned throughout the network, the load should be distributed evenly among all the servers.

Half of the publications were used offline to seed the siterank computation, including at least 25% of the pages from each site.

3.3.1 Publication handling

We first counted the number of hops in the underlying network needed to crawl/push all documents to the publish-subscribe nodes, from the original random nodes to which their sites were assigned. In the case of the 1S architecture, this meant routing all publications to the single server, following the shortest path from each publisher; 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 publish-subscribe server closest to their publisher node.

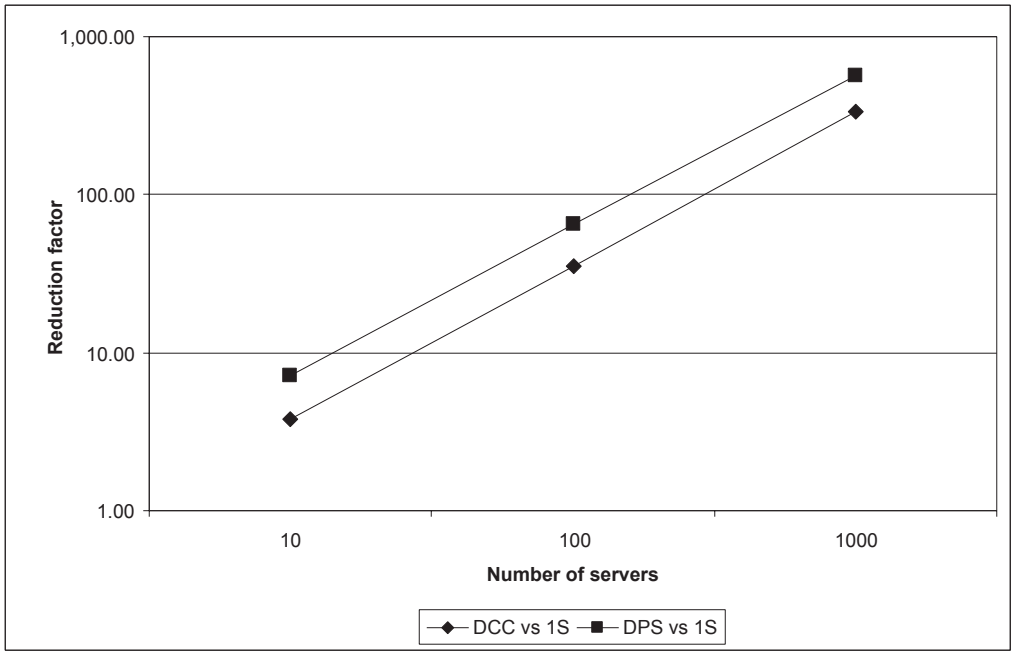

Figure 3.9: Reduction factor in the number of hops to push all documents to the matching server(s). Data in Table A.8.

Figure 3.9 shows that the improvement in the total 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.

The raw data (Table B.1 in Appendix B) shows that the highest-degree DPS (1,000 servers) setup achieves a very low publication path length with small standard deviation, since each publish-subscribe server collects documents in its local stub domain in the underlying network. A smaller set of servers needs to collect documents across stub domains, and the extreme case of the 1S single-server setup shows a very high standard deviation due to the high variance in publication path lengths, crossing stub and transit domain boundaries. Similarly, the added path between crawler servers and the central control server introduces higher variation in the DCC results compared to the similar-size DPS data.

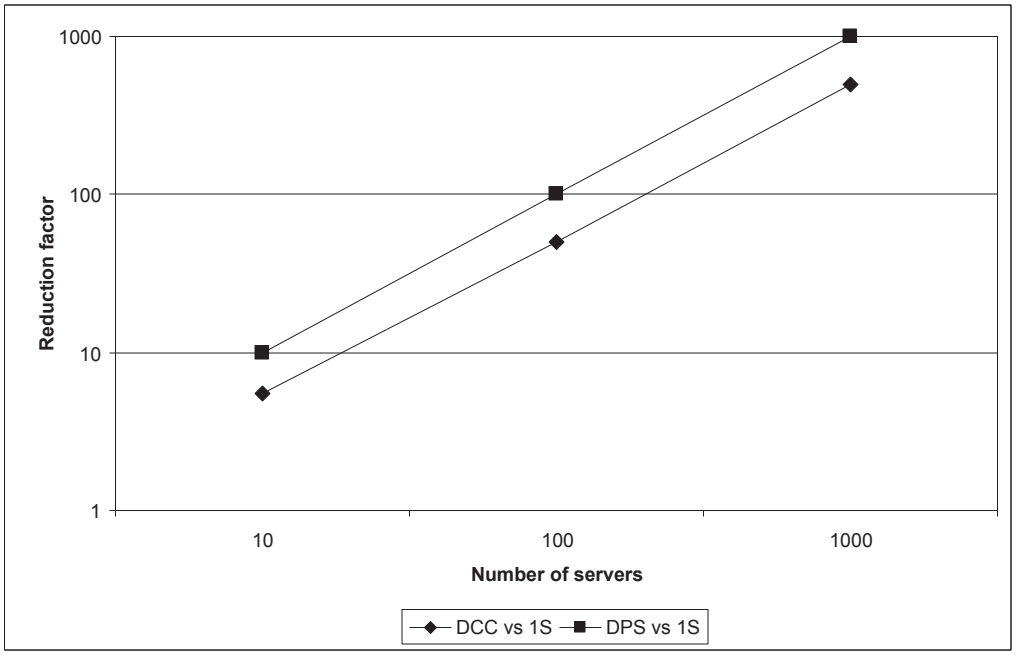

Figure 3.10: Reduction factor in the number of incoming publications per server.
Data in Table A.9.

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 3.10.

- DPS performs as expected and distributes the publications evenly (on average) among all servers, due to the uniform allocation of publishers to servers in our experiments. The standard deviation (Table B.2 in Appendix B) is small, and it decreases slower than the average value – since each site

is managed by a single server, reducing the total number of documents per server without reducing the site sizes increases the variation slightly.

- In DCC this is only true for the edge servers, since the central server still receives all the publications; as a result, the average publication load when taking into consideration the central server is approximately double compared to DPS. This highly uneven load is the cause for the large standard deviation in the raw data for DCC (Table B.2 in Appendix B). The standard deviation measured only across the crawling servers is very similar to DPS.

  Since the central server in DCC must collect all publications, its load is the same as the single server's load in 1S. While parallelized crawling benefits from partitioning the document set among the many servers, all other operations (matching, relevance computation, notifications) are performed from the central server so this improvement in the average load over 1S is misleading.

- Both distributed methods are far better than the single-location approach, where a single server has to collect and process all documents.

3.3.2 Notification management

We also measured the performance of the notification component. 100 million subscriptions with 1-6 keywords each were randomly generated from the publication's vocabulary, using the same overall keyword frequency as the publications. One of 10 million user IDs (subscribers) was randomly assigned to each subscription, to average 10 subscriptions per user. Subscriber IDs were randomly assigned to nodes in the underlying network and connected to the closest system server (based on number of hops in the underlying network); similar to publications, this assignment followed the underlying network hierarchy, preferring same stub domain, then same transit domain. 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 publish-subscribe 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 publish-subscribe servers that handle the matched remote subscrip-

tions. Each of these inter-server notification messages summarizes all the matched remote subscriptions on the destination publish-subscribe servers, 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. Publication path length is measured from the matching publish-subscribe server to the subscriber's assigned node in the underlying network. In DPS, this also includes the inter-server notification initiated by the matched remote subscription; however, when adding up all path lengths, this inter-server message only participates a single time in the sum, since the same message delivers the inter-server notification for all remotely matched subscriptions originating from the same server.

We averaged results over all notifications, with an average matching set size of 100 subscriptions per publication. Overall, the notifications approximately followed a Zipf distribution - few publications are very popular, and most publications match just a very small number of subscriptions.

To evaluate the system performance when handling different types of publications, we collected separate statistics for:

- less popular documents – *10-pubs*, publications matching approximately 10 subscriptions (between 5-15), approximately 5% of all the notifications
- very popular documents – *1000-pubs*, publications matching approximately 1000 subscriptions (between 900-1100), less than 0.5% of all the notifications

in addition to the average statistics over all publications. In general, we can observe that the relative improvement of DPS over 1S is smaller for 10-pubs than 1000-pubs:

- A very popular document that matches many subscriptions will generate many notifications, but they will be efficiently shared between many subscribers on each server; for all the matched remote subscriptions originating from a server, a single message is sent to that server to notify its subscribers, who are then individually notified locally through shorter paths. The same document in the single-server system will need higher notification traffic to deliver the notifications to all subscribers.
- Less popular documents match fewer subscriptions which can originate from separate servers, so an internal notification sent to another server might be intended for a single subscriber; while not many messages are needed to deliver these notifications, the single-server system also needs a small number of messages so the relative improvement in the distributed approach is not so significant.

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, as seen in Figure 3.11. This is because in addition to the same number of user notification messages, the distributed setting also requires inter-server notification messages.

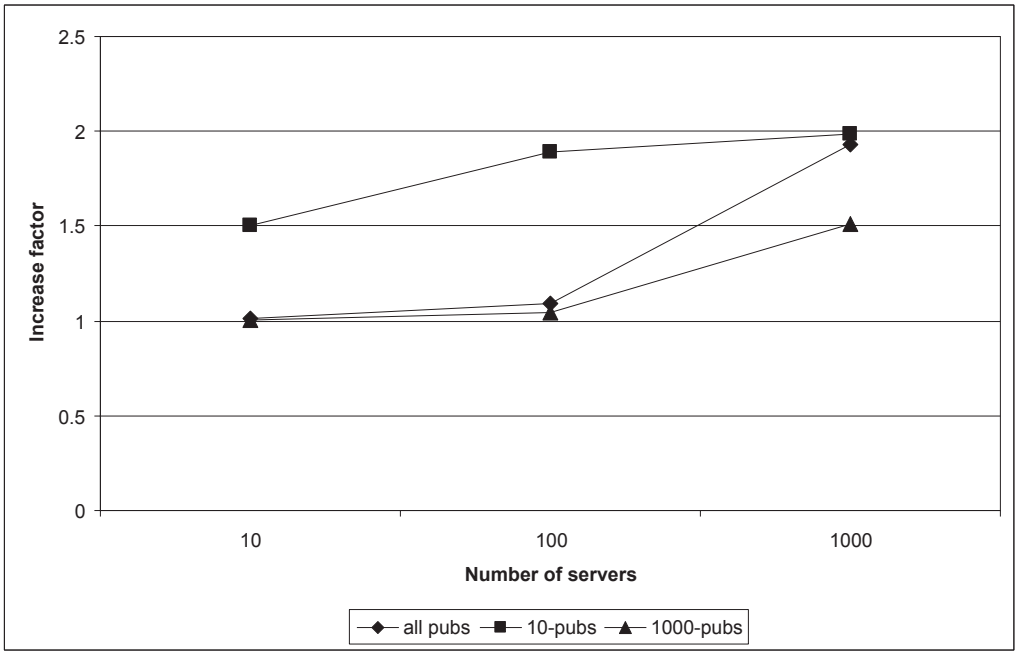

Figure 3.11: Increase factor in total number of messages sent by all servers for one publication - DPS sends more messages than 1S for each match.
Data in Table A.10.

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 3.12). As expected due to the uniform distribution of subscriptions, in DPS with k servers approximately $1/k$ of the subscriptions were local to the matching server.

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 individual user notification paths are much shorter from a nearby publish-subscribe server as opposed to a single server for all users; the measurements summarized in Figure 3.13 show that overall network usage per match scales very well in DPS.

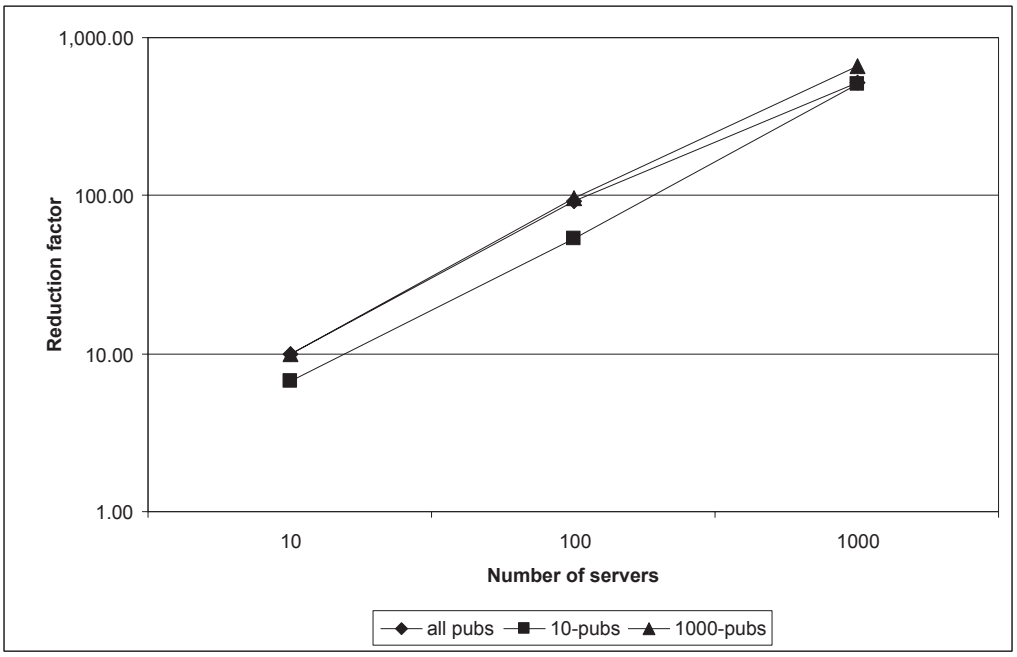

Figure 3.12: Reduction factor in number of messages sent per server for one publication - DPS divides the load among multiple servers.

Data in Table A.11.

Each dynamic multicast in DPS is only sent through the servers that are interested in this notification, i.e. contain subscriptions matched remotely at the source server. Publish-subscribe servers have good connectivity (each publish-subscribe server is connected to a transit router) so each dynamic multicast only has to cover a very small number of hops in the underlying network, on average less than 2 times the number of destination servers. The main advantage however comes from one dynamic multicast delivering the notifications for multiple subscriptions matched on a remote server.

Since the inter-server messages are significantly fewer and travel shorter distances than the average user notification, the network usage is inversely proportional to the system size (number of publish-subscribe servers).

The raw data used to derive these results is summarized in Tables B.3, B.4 and B.5, Appendix B. A couple of notable observations:

- Since the inter-server notification is shared among matched subscriptions with the same underlying query, the average destination list size is smaller than the number of matched subscriptions.

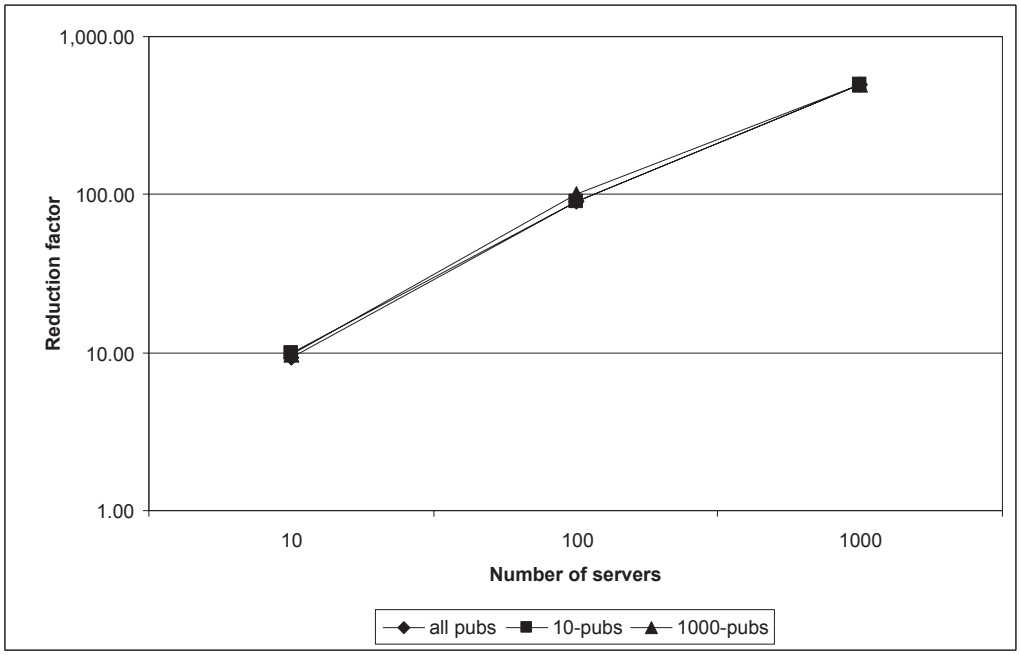

Figure 3.13: Reduction factor in total network hops per matched publication - each message travels much less in DPS.

Data in Table A.12.

- The total network hops in DPS approaches the number of destinations for any matched publication as the size of the network increases. Due to the high connectivity of the DPS servers, the inter-server notification is very cheap, and each individual user notification is very short as the corresponding DPS server is only 1-2 hops away.

3.3.3 Distributed communication costs

Finally, we also measured the new communication costs specifically introduced by our architecture.

As shown in chapter 2, SiteRank is much faster than PageRank; DPS does not need a central server that must match all publications, however we still use a dedicated ranking server to collect all the link information, compute site and page ranks 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 3.14, as a percentage of the publication cost from publishing site to corresponding publish-subscribe server; the measurements were averaged over all

100 million publications. As expected, the smaller size of the link summary 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. The one element that depends on system size is the average ratio between the path length from publish-subscribe server to pagerank server and the path length from publisher to publish-subscribe server; this ratio decreases as the system size increases - even if publishers get closer to the system's servers as the number of servers increases, the rank server gets closer to the publish-subscribe servers faster.

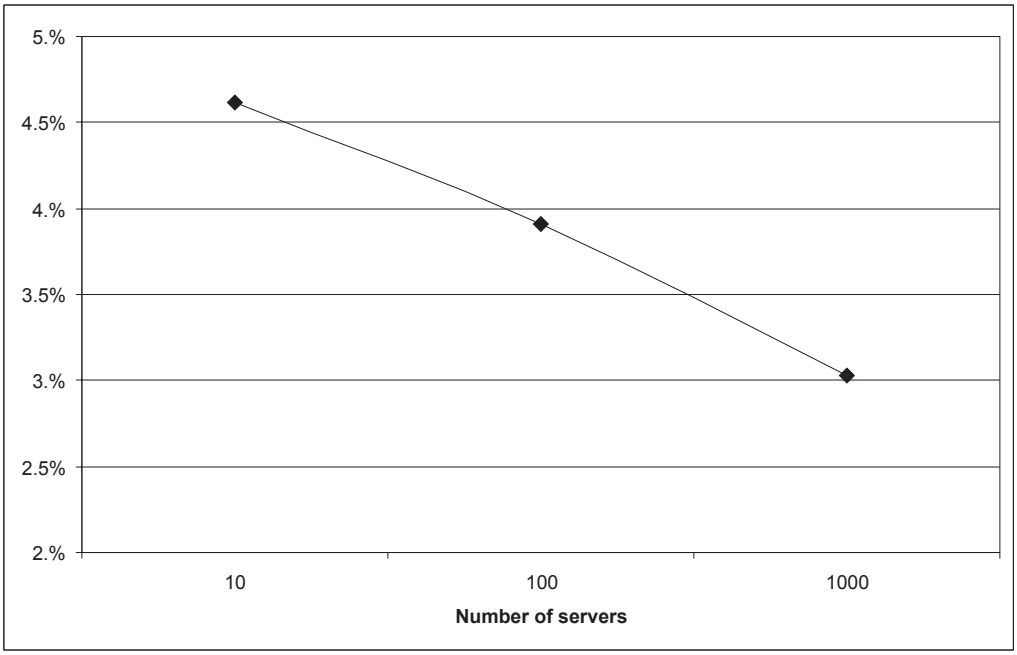

Figure 3.14: Rank server communication is a small percentage of overall publication traffic. Data in Table A.13.

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). An initial base of 100 million subscriptions was partitioned among the publish-subscribe servers, we then measured the network utilization needed to process 50 million new subscriptions and 50 million subscription updates randomly distributed among all servers.

The results are summarized in Figure 3.15. Even with a highly 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. Table B.6 in Appendix B shows the average network utilization for a notification in 1S and DPS of various sizes; the average size of a subscription is about 20 bytes (3 keywords encoded in 4 bytes each, user ID and originating server address). While in the 1,000-server DPS system the overall network utilization for a single subscription crosses over 1 megabyte, the price of routing a publication through the distributed system would be much higher.

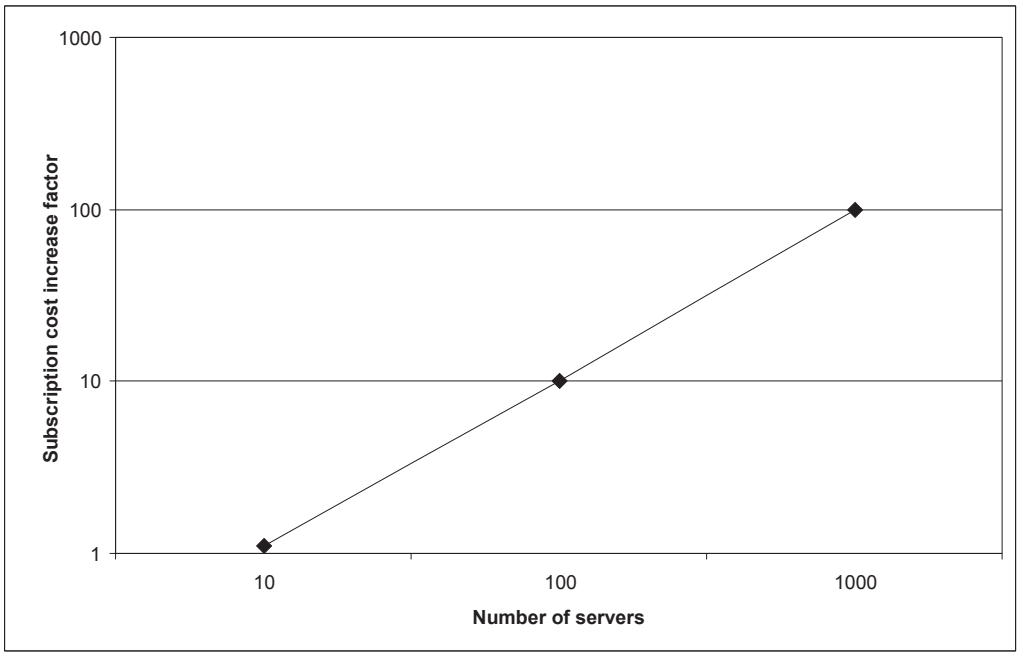

Figure 3.15: Subscription maintenance overhead increases with system size.
Data in Table A.14.

Since subscriptions are much smaller and are added or updated much less frequently than publications, and we achieve a linear improvement of the publication routing mechanism, subscription management is not a 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:

- Subscriptions can be replicated in batches, broadcasting them to other servers only periodically or when a minimum number of new subscriptions have been collected at the same server. This introduces a delay in subscription replication and could have a significant negative quality impact in a real system, since the user could not receive relevant publications that arrived at a remote server after she

had already set up a matching subscription. However, if the subscription volume is high enough the user-perceived delay in propagation would be very short.

- Multiple different subscriptions from the same originating server can be aggregated together before replication (partially overlapping subscriptions, see Section 3.2.5). This can however introduce false positives when the aggregate subscription is matched remotely, but none of the individual subscriptions is matched locally, and requires additional publication traffic for local matching.

3.4 Summary

We designed and developed a highly-scalable, general-purpose persistent search system (Distributed Persistent Search - DPS) based on a distributed publish-subscribe architecture. By partitioning the publication processing load among all servers and eliminating the publication traffic between the system's servers, we reduced crawling latency significantly, allowing the system to scale easily with a large publication base and a large number of servers. This requires that subscriptions are replicated all servers to match against all publications. The use of dynamic multicast significantly lowers the notification load for each server; multiplexing all identical matches in a single message also lowers the network load required to notify the owners of matched subscriptions.

Of course, replicating subscriptions adds a cost that is not present in existing methods, but it more than covered by eliminating the internal publication traffic. Various improvements can be made to reduce this cost, most notably replicating subscriptions in batches and aggregating similar subscriptions.

Similar to existing approaches, DPS collects all page link information in one location at a rank server to compute link-based ranks, a potential single point of failure in this distributed system. Even if the link information is significantly smaller than the full document content, this component can limit the scalability of the distributed system if the number of servers or the publication base increases significantly. This is the focus of the next chapter.

Chapter 4

Distributed SiteRank

The goal of our ranking work is to develop query-independent ranking algorithms that are more appropriate for persistent search systems, and at the same time highly scalable for integration with large-scale publish-subscribe architectures. The first part has been addressed by chapter 2 – SiteRank was designed specifically to reduce the bias against new pages in persistent search systems, and it scales better than traditional PageRank; however, it is still requires that all page link information is known at a single location (single-server, or shared memory parallel architecture). In the introduction, we argued that distributed architectures should scale much better than single-homed approaches, and Chapter 3 summarizes Distributed Persistent Search, our proposed distributed architecture for persistent search.

Distributing subscription management, document processing and notification routing has already been attempted in other persistent search systems (as discussed in Chapter 3), but link-based relevance computation introduces a major challenge - both PageRank and SiteRank require that the link structure of the document set is known at a single location (the rank server), even if document processing is distributed. 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 increase overall performance. Existing distributed PageRank solutions are plagued by high communication costs, or use heuristics that reduce communication but only generate approximate results.

The use of a centralized ranking component in a distributed persistent search system limits its scalability and can become a single point of failure. Our goal is to improve the overall availability and scalability of the distributed search system by distributing the rank computation – this would eliminate the reliance on a

single special server, and speed up the entire system by parallelizing an expensive computation.

To achieve this, we propose a new partitioning scheme for the link graph that significantly reduces communication while accurately computing the same ranks as the single-server SiteRank computation used as reference. DPS already use a geographically-based partitioning scheme where each publication is handled by the publish-subscribe server closest to the network location of the publication's site. Our proposed partitioning scheme for the distributed rank computation uses different criteria to assign publications to publish-subscribe servers; we will evaluate both partitioning schemes for rank computation and also in the overall distributed persistent search system (DPS).

It must be noted that while we re presenting the partitioning algorithm and distributed rank computation in the context of SiteRank, both can be applied without any changes to PageRank. Furthermore, both Distributed SiteRank and Distributed PageRank can be used in regular (non-persistent) search systems when high scalability is required.

This chapter summarizes the design and evaluation of the Distributed SiteRank algorithm. We present a high-level design of the proposed algorithm, with emphasis on the partitioning scheme that reduces communication during computation. The algorithm is evaluated by comparing its performance against single-server and other distributed architectures. Since this algorithm is designed for relevance computation in a persistent search system, this chapter also includes our evaluation of Distributed SiteRank as part of our proposed Distributed Persistent Search system (see chapter 3). We conclude the chapter with a short summary.

4.1 Related work

The basic PageRank algorithm was described in section 2.1.1 - an iterative algorithm that at each step must readjust the rank of each page based on the ranks of pages pointing to it. Distributing this computation means that some of these page-rank contributions need to be requested over the network, so it is no surprising that not many attempts have been made to solve this difficult problem.

A fully distributed implementation of PageRank for peer-to-peer systems is proposed by Sankaralingam et al. [48]. Their algorithm converges rapidly for iterative updates, 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 a very weak overall performance – each iteration requires extensive communication to look up link contributions between the distributed servers, as documents are randomly assigned to servers. Even as the computation is parallelized among all the servers, the running time for a 5 million document set is estimated at close to 19 hours, over high speed connections. As a point of reference, computing single-server PageRank over 75 million documents takes about 5 hours [41].

Clearly, a naive distribution of the pages results in frequent rank contribution lookups over the network, negating any advantage gained by computing ranks in parallel. Additionally, high-quality pages have many inlinks, therefore the communication load on the servers computing their ranks is disproportionately high, which in turn slows down those servers' responses.

A different approach is to lower the communication load by simply ignoring some information: Wang and DeWitt [58] and Wu and Aberer [61] propose two similar distributed solutions - each server computes local PageRank values, adjusting them based on the relative importance of the servers.

The authors argue that this rank computation can be useful in a decentralized rank computation:

- The number of sites is two to three magnitudes lower than the number of documents (based on their limited data set), so site ranks can be computed much faster in a single location.
- Each server in the distributed system computes local page ranks for the pages it is managing, using the links between these pages but none of the links from pages managed by other servers.
- The site ranks and the local page ranks are combined to produce the global page rank values. These values can be quite different from traditional PageRank values, and the authors provide no analysis of how well do the new rank values predict page popularity.

One important omission in this work is the cost analysis for replicating all site rank values to all the servers. This design, while intended to decentralize the computation, still relies heavily on gathering all link information and computing the site ranks in a single location. The cost of collecting this information, computing site ranks and replicating them at all servers in the system can be prohibitive if done too often. At the same time, not computing the site ranks often enough may lead to stale results in a highly dynamic document base.

While these two approaches significantly reduce communication costs, they also generate only approximate page rank values. As we saw when we compared SiteRank to HostRank [64] in Chapter 2, the full

utilization of each page's inlink information is essential to predict the popularity of webpages. The two distributed approaches mentioned above [58, 62] never use the individual links pointing to a page from another page that is assigned to a different server.

More importantly, the relative importance of the servers is a site-rank-like measure that must be constantly refreshed to be meaningful - this computation however requires significant communication between all the servers.

4.2 Distributed SiteRank design overview

During the iterative link-relevance computation, one must repeatedly look up the rank contributions for each inlink. In distributed rank computations, subsets of pages are assigned to different servers, so looking up rank contributions might involve messaging between pairs of servers. Each of these individual messages is quite small, however all the messages required when computing PageRank for billions of pages add up to a very high communication overhead. Even if all the rank lookups between two servers are batched together, the number of messages can still be high as many other servers contain pages or sites linking to the currently considered page/site. Therefore the partitioning of the document space and the assignment of document subsets to servers are very important aspects of our algorithm.

We will describe the separate components of our approach in the next subsections, and summarize the experimental results in section 4.3.

4.2.1 Web graph partitioning

While the distributed rank computation has to run quite often (every time a large enough percentage of the pages or links are updated), the assignment of pages to servers is done much less frequently. We can therefore afford to use a computationally expensive partitioning algorithm, with the ultimate goal of minimizing inter-server communication during the rank computation.

A good approach assigns pages to servers such that the number of inter-server rank lookups is minimal. During the SiteRank computation, rank contribution lookups between different servers are necessary when a site's inlink comes from another site handled by a different server. To minimize our communication load, we must assign sites with many links between them to the same rank computing server.

To achieve this, the site-link graph is recursively divided in dense components, applying a min-cut division at each step. Given the set of pages and the links between them, we construct the site-link graph, where the weight of an inter-site link is given by the number of inter-page links between the two sites. We transform this directed graph into an undirected graph *G* by summing the weights of the directed links for each pair of connected sites - during rank computation, a request from either of the sites to the other can require communication if the sites are managed by different servers.

Given graph *G*, the graph partitioning phase of our Distributed SiteRank algorithm can be naively summarized as follows:

```
S = {G};
while (S is not empty  and
       we need more components) {
    // S is the set of subgraphs to be
    // divided during this step.
    for every G[i] in S {
        // Divide subgraph/component G[i];
        // T is the set of resulting components.
        T = min_cut(G[i]);
        // And update the set of subgraphs/
        // components for the next step.
        S = S - {G[i]};
        S = S + T;
    }
}
```

Since the exact min-cut algorithm used was not the subject of our research, we used one of the more recent algorithms proposed by Stoer and Wagner [54].

At every step of the proposed partitioning algorithm, in the trivial case that the current graph to be divided $G[i]$ is disconnected, its connected components are returned as the set of subgraphs.

The original site-link graph is partitioned through this recursive graph partitioning scheme into a set of dense components. Each of these dense components has many more links inside (between sites in this component) than outside (from sites in its components to sites in other components).

There are two termination conditions for this algorithm. First, we must finish when there are no more subgraphs to be divided; this condition is controlled inside the *min_cut* method, where a component can be considered too small for further division. Our target graph (the World Wide Web link graph) is large enough that this condition is not reached for all components before the second condition is met.

We can also end when the current set of components is large enough. One might stop when the number of components equals the number of rank computing servers. A better approach however is to generate more components than servers, then assign multiple components to each server; this assignment process takes into consideration the sizes of the components to ensure a good load distribution among all servers. Of course, since our main goal is reducing communication load during computation, the assignment of components to servers also takes into consideration the links between components, placing strongly connected components on the same server.

This recursive partitioning operation is applied to the set of known pages/sites, and further incremental updates are straightforward. New pages on an already known site are managed by the same server that already manages the site. New sites (a much less frequent event) are assigned to a server based on their known connectivity, placing them in the component they are most connected to. When a large enough percentage of components are updated with new sites, or a large enough percentage of the links are modified (new or updated links), a complete partitioning is required to balance the load among the servers, and reduce inter-server communication. Both the original partitioning and further updates are performed by a designated server chosen through distributed leader election [35]. Each rank server maintains a copy of the maps assigning sites to components and components to servers, this data is surprisingly small if fingerprints are used instead of actual site addresses. The maps are fully updated after a re-assignment, and updated through broadcast when a new site is added to the document set.

The elected leader server collects all site link information from the other ranking servers and performs the partitioning operation. It then broadcasts the new assignment to all the ranking servers which use the old and new assignment to exchange link information in the background. The leader server checks that all servers update their assigned components, then switches them over simultaneously to the new assignment.

While the re-partitioning operation uses leader election and the partitioning is done on a single server, the communication of all the rank servers with the single elected server is not a full-time requirement in the rank computation. It is only needed during re-partitionings, which are rare events – new pages are always added to the server managing the corresponding site, and new sites are assigned to the server managing the component closest to the given site.

Note that the only information used to partition the document set is the link structure - if available for all sites, geographic locality could potentially improve this algorithm by using geographic location to break ties during partitioning. However, very remote sites can sometimes have strong links between them, e.g. different top-level sites for multi-national companies, all pointing to a single location for product information for their entire catalogue.

Since the partitioning is performed off-line, it does not influence the running performance of the persistent search system. However, full link information for each document is maintained only at the server managing the component that contains the corresponding site, so documents must be routed to the appropriate server when they are published. Section 4.2.4 addresses this aspect, merging the two partitioning schemes used for ranking and document management.

4.2.2 Computation

Once the dense components are assigned to the rank computing servers, we can perform the actual rank computation. Each server computes ranks for the sites assigned to it, querying other servers when it follows inlinks from sites in other components. The same designated server chosen through distributed leader election is used to synchronize all the servers so they are all computing the same SiteRank iteration, and to verify the SiteRank termination condition - identical to centralized SiteRank, the algorithm stops when the scores converge, i.e. the average difference in rank values between iterations is less than a very small threshold.

Synchronizing the iterations over all the rank servers is needed for consistency, so all servers work on the same iteration and therefore compute the same scores as the single-server version. Eliminating synchronization could potentially speed up the computation, allowing each server to proceed at its own pace based on its particular load. However, this might also increase the number of iterations needed to converge as the rank values oscillate due to the lack of synchronization.

During each SiteRank iteration, each server batches together the rank lookup messages it needs to send to sites in other components; for each server it needs to contact, it also combines the messages for all the components managed by that server. Each server sends therefore a single rank lookup message to each other server it needs to contact, regardless of the degree of connectivity between sites on these servers; the size of the message is determined by the number of links. The dense component-based partitioning ensures that both the size of these messages and the number of messages sent between pairs of rank servers is small – the sites that are most connected to each server's sites are in the same connected component, on the same server, and don't require remote lookup.

The site ranks can be recomputed off-line whenever a large enough number of links has changed, without requiring a re-partitioning of the site-graph if the load is still balanced among the servers. As presented in the previous section, a full re-partitioning of the site-graph into dense components can also be initiated if a large enough number of sites and/or links has changed, possibly leading to a new assignment of sites to servers.

Similar to single-server SiteRank, site ranks are computed in the iterative computation off-line, and page ranks are computed from site ranks on request - when a page is potentially part of a notification in the persistent search system and we need to look up its rank, we compute it based on the site's rank, the page's position within the site and the individual links pointing to this page. There is no need to pre-compute the ranks of individual pages if they will not be sent to users. This approach takes into consideration updates to the link graph that were applied only after the most recent site rank computation, e.g. a new link pointing to a page will increase the page's rank even if this link was not included in the site-rank computation. The reduced cost of distributed rank computation allows frequent re-computation of the site ranks, so page ranks are very rarely not aligned with their sites' ranks.

Essential during site-rank computation, the min-cut based partitioning scheme ensures that on aggregate we add the smallest possible communication cost even when looking up rank contribution on individual page links during individual page rank computation.

4.2.3 Distributed PageRank

While Distributed SiteRank is built on top of single-server SiteRank, the same distributed approach can be applied to any citation-based ranking method. Given the similarity between SiteRank and PageRank, we also easily developed a Distributed PageRank algorithm. As expected, the performance improvement

of our Distributed PageRank method over single-server PageRank and other distributed PageRank is very similar to Distributed SiteRank's advantage over its competition. The page ranks are computed directly, using the same algorithm that computes the site ranks in Distributed SiteRank; naturally, there is no need for the additional step of computing individual page ranks from site ranks.

Similar to single-server and Distributed SiteRank, Distributed PageRank can be used in regular (non-persistent) search systems too.

4.2.4 Distributed SiteRank in DPS

The Distributed SiteRank algorithm was developed as a component of a large-scale persistent search system (DPS). While it achieves good performance compared to other distributed citation-based authority computations, it does so mainly due to a graph structure-based partitioning scheme. The DPS system uses its own partitioning scheme to assign documents to publish-subscribe servers - it is designed to minimize the publication latency by shortening the paths between publishers and publish-subscribe servers.

Using Distributed SiteRank for authority computation in DPS requires that we use a single partitioning method, since both the publish-subscribe component and the rank computation work on the same document set. By distributing the servers geographically based on publication density and assigning publishers (web-sites) to servers based on proximity (as in the original DPS system), we reduce the publication latency from publishers to the publish-subscribe servers; if we use a graph-partition based assignment of components to servers, the publication latency increases, but the PageRank computation is speeded up by reducing rank lookup time between servers (the components generated by the geographic distribution exhibit some link locality, but at a much smaller level than the components in Distributed SiteRank).

We experimented with both versions and our results are summarized in section 4.3.2.

4.3 Evaluation

While the main focus of this chapter is the proposed Distributed SiteRank algorithm, stand-alone experiments on this algorithm do not give us a full evaluation of its performance. A stand-alone evaluation of Distributed SiteRank compared to the single-server SiteRank computation simply shows that the algorithm is speeded up by partitioning the computation load among multiple servers. Also, the dense component-based partitioning outperforms other known partitioning methods.

However, we initially proposed a different document partitioning scheme in DPS, in which publications are assigned and routed to the geographically closest publish-subscribe server. Also, the distributed lookup for individual page link contributions needs to be performed when the pages are processed by the publish-subscribe servers, since the ranking servers are now co-located with them.

Testing our proposed ranking mechanism (DSR) as part of the entire persistent search system (DPS) shows that even after taking into consideration the additional communication needed between rank servers while computing PageRank, DSR is still a huge improvement over single-server rank computation, reducing both the computation time and the communication costs of collecting all documents at a single location. It also shows that the dense-component based partitioning scheme outperforms other alternatives when the cost of crawling is added to rank computation.

To evaluate the performance of Distributed SiteRank, we reused the event-driven, message-level simulator implemented to evaluate the Distributed Persistent System (see Section 3.3) : transit-stub model with approximately $10,000$ routers and $50,000$ links. 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. k publish-subscribe/ranking servers were randomly chosen from the most-connected nodes in the simulated network, and their connectivity further improved by additional links.

The system sizes (k) used in these experiments were 10, 100 and $1,000$ servers. For the single server ranking method, a dedicated server was randomly selected from the top 1% most-connected nodes and used exclusively for rank computation; for the distributed ranking methods, the same servers performed publish-subscribe duties (publication and subscription management, matching, notification) and ranking.

Multiple runs (100) of the experiments were averaged, each run using a different random assignment of servers to nodes; similar to the DPS experiments in Section 3.3, there were no major variations between different runs – the standard deviation was less than 4% of the mean value.

Since our new tests need to measure operations that involve both computation and communication, we assigned random link latencies to the simulator network above. Links between transit domains have latencies between 40-60ms, transit stub links between 10-20ms, and the intra-stub domain links between 1-5ms.

The publication dataset used for these experiments is a June 2005 webcrawl from the Stanford WebBase project [30], containing 100 million text documents (html pages).

4.3.1 Stand-alone Distributed SiteRank evaluation

We measured the performance of Distributed SiteRank (DSR) and compared it with single-server SiteRank (SR), independent of other components in the persistent search system. To illustrate the advantage of DSR over other partitioning methods, we also compared DSR with two other distributed ranking algorithms - both use a different partitioning scheme, but run the same rank computation.

- The naive case breaks down the alphabetically ordered list of sites into intervals, and assigns these intervals to rank servers ($DSR - List$); to balance the load evenly among servers, the intervals are chosen so that each of them has approximately the same number of pages. The lists can be minimally adjusted so that all the pages from a site are assigned to the same server.
  In this form, this partitioning is used by some parallel crawlers (e.g. [25]) to distribute crawling of large publication sets. This partitioning scheme is not used for distributed rank computation, and we will see that it would not offer good performance.

- The distributed algorithm proposed in various similar forms in many publications assigns sites (pages) to servers based on proximity ($DSR - Prox$) – identical to the DPS partitioning scheme in Chapter 3, each site's pages are managed by the ranking server closest to that site. [58, 62] use this scheme to assign publications to servers and we will first compare it against DSR to determine its performance for distributed rank computation.
  Later, we will compare the performance of DSR and DSR-Prox for publication crawling combined with on-demand page rank computation too.

For this experiment, random subsets of 1 million, 10 million and the full subset of 100 million pages from the webcrawl were used. For the 1 million and 10 million subsets, the results were averaged over 100 runs, using a different random assignment of servers to nodes and a different random subset of pages for each run; for the entire 100 million document set, the results are averaged over 100 runs with a different random server assignment at each run. However, different runs for the same document set size show very similar results, variation between various runs stayed under 5% of the mean for all different experiments.

We measured computation time and total time (computation time + communication time). For the distributed ranking methods, we did not measure the time required to partition the document set, since this operation does not need to be performed at every rank computation and is very infrequent. Single-server SiteRank (SR) was the base setup against which we computed the speedup gained in the three distributed versions.

This experiment was run on a 40-machine Linux cluster, but the number of distributed servers was sometimes larger than the number of available machines (e.g. 100 or 1000 servers). In those cases, at each iteration each physical machine was running multiple simulated servers, and the longest time for any of the servers was chose as the running time for each iteration. Due to the synchronization step at each iteration, this correctly simulates the running time of the entire system.

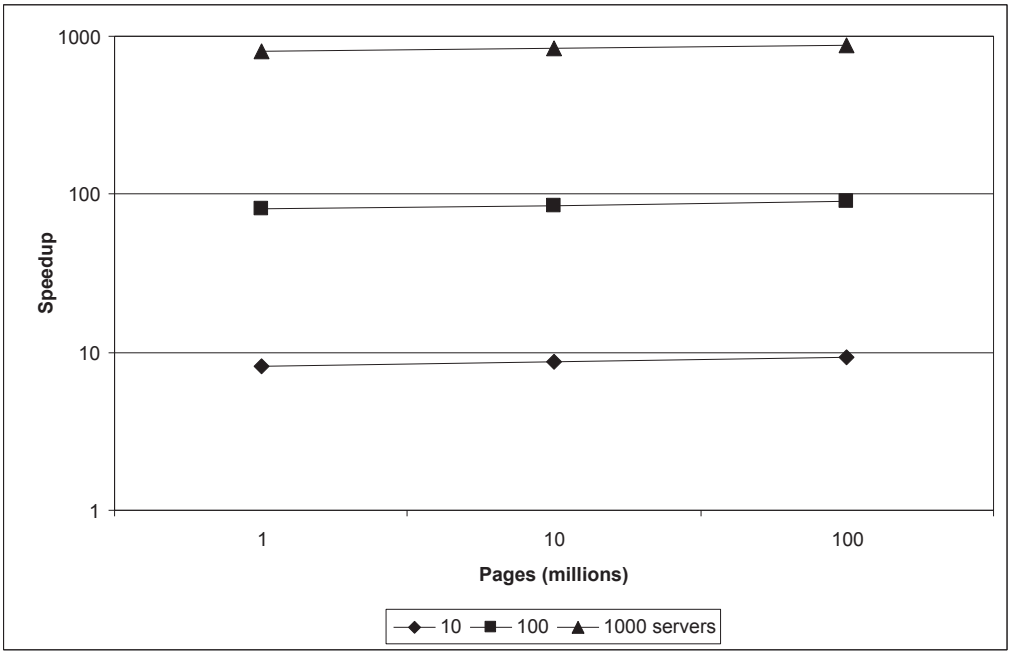

Figure 4.1: DSR vs. SR speedup factor (total time), varying number of rank servers.
Data in Table A.15.

The speedup of DSR compared to SR is illustrated in Figure 4.1, measured as the ratio between total time (computation + communication) to compute all ranks once in single-server and distributed setup. As expected, Distributed SiteRank (DSR) shows near linear reduction in the computation time compared to single-server SiteRank (SR); the speedup is slightly higher for larger document sets, when more dense components ensure a better load distribution among the rank servers. Since the iterative rank computation

on large document sets is very time consuming, including the communication time does not change the results considerably; it is most visible for larger system sizes and smaller data sets, where the speedup is slightly smaller.

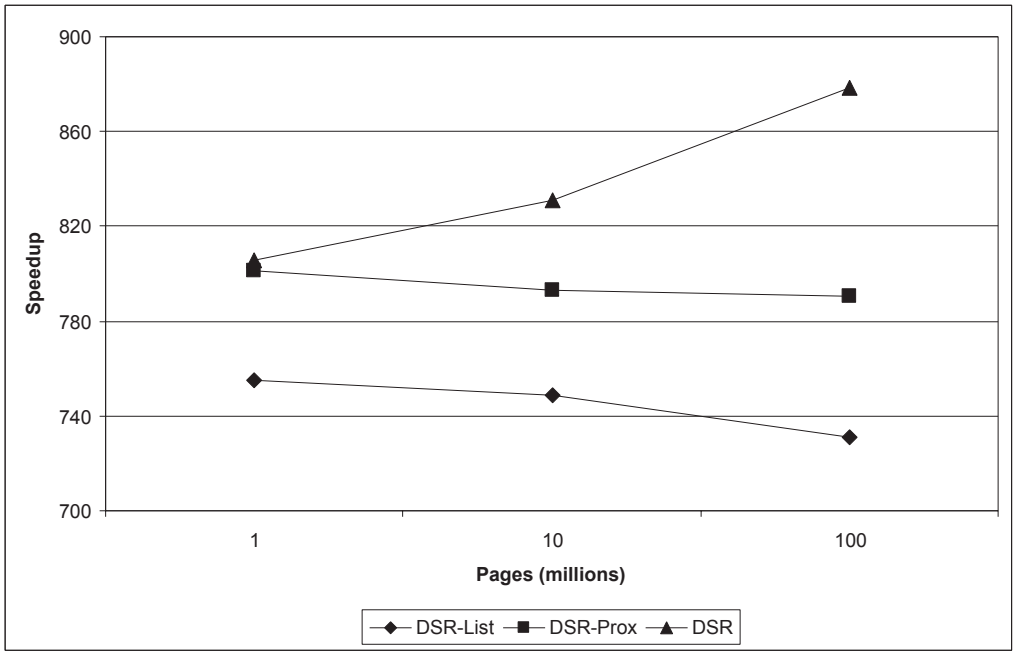

Figure 4.2: Distributed vs. single-server speedup factor (total time), various distributed methods, 1000 servers.

Data in Table A.16.

To compare DSR against DSR-List and DSR-Prox, we also ran these algorithms on the same data sets. Again, we measured the speedup as the ratio between total time for these methods and single-server SiteRank. Figure 4.2 summarizes the results for the 1000-server system size. The computation time is very similar for all distributed methods, since the load is balanced between the servers. The communication time however is very different:

- DSR sends the minimum number of messages between servers;
- DSR-Prox shows some link locality based on site proximity, but it sends about twice as many messages - when computing a new rank iteration on a server, we are batching rank lookup messages to other servers both in DSR and DSR-Prox, but DSR-Prox needs to query about twice as many servers due to the suboptimal partitioning; the larger the data set, the higher the communication overhead compared to DSR, an important factor for real systems;

- DSR-List has no link locality so the computation speedup is severly downgraded by the communication costs.

Even if the computation costs are much higher when computing ranks for a very large data set, a significant increase in the communication costs makes a big difference to overall performance.

4.3.2 Distributed SiteRank in a full Distributed Persistent Search system

Taking into consideration the full DPS system, we compared the performance of the distributed SiteRank methods (DSR, DSR-Prox, DSR-List) against single-server SiteRank (SR) during normal run-time involving document routing to processing servers in addition to page rank lookup. All setups tested below were based on DPS running on 1,000 publish-subscribe servers.

Half of the publications were used offline to seed the siterank computation, at least 25% of the pages from each site. Site ranks were computed in each system based on this subset of pages.

Publication sites were assigned to random non-server nodes (publishers) in the network – note that this is a more pessimistic distribution than in the real world, where some sites that have many links between them might be hosted by the same publisher. While this observation was true in previous experiments too, it only has full effect in this experiment where we combine crawling (which depends on the assignment of publishers to nodes in the underlying network) with rank computation (which performs best if documents are partitioned in dense components).

- For the single-server ranking setup, documents were routed to the closest publish-subscribe server, which in turn queried the rank server for the link relevance value of this page.

- In all the distributed ranking setups, the same nodes were performing both the duties of publish-subscribe servers and rank servers - each node would both manage the document routed to it and compute its rank. Documents were routed from publishers to the DPS node determined by the partitioning method used:

  - the closest server (based on number of hops in the underlying network) for DSR-Prox;
  - the server assigned by the alphabetical list partitioning for DSR-List;
  - the server assigned by dense component-based partitioning for DSR.

All documents in the test set were then published into the system and routed to the corresponding publish-subscribe server. Each server computed the rank of each page it received. For single-server SiteRank, each publish-subscribe server asked the ranking server to compute the rank of all the pages; same as in the original DPS version (section 3.2), the publish-subscribe servers only sent a link summary to the single rank server in SR, and not the entire document.

To focus this evaluation on document ranking and the necessary routing of documents to servers, no subscriptions were stored and naturally no notifications were sent out from the publish-subscribe servers; see section 3.3 for a full evaluation involving all components of the publish-subscribe system, using a single-server ranking component. (Since subscriptions are always replicated at all servers in DPS, these costs do not change when we change the rank computation algorithm.)

For each document, the network consumption was measured by summing up the message size multiplied by the number of hops in the underlying network over all the messages - initial routing of the document, followed by rank lookup (query the rank server for the current document's rank in SR; look up individual inlink rank contributions for the current document in distributed setups).

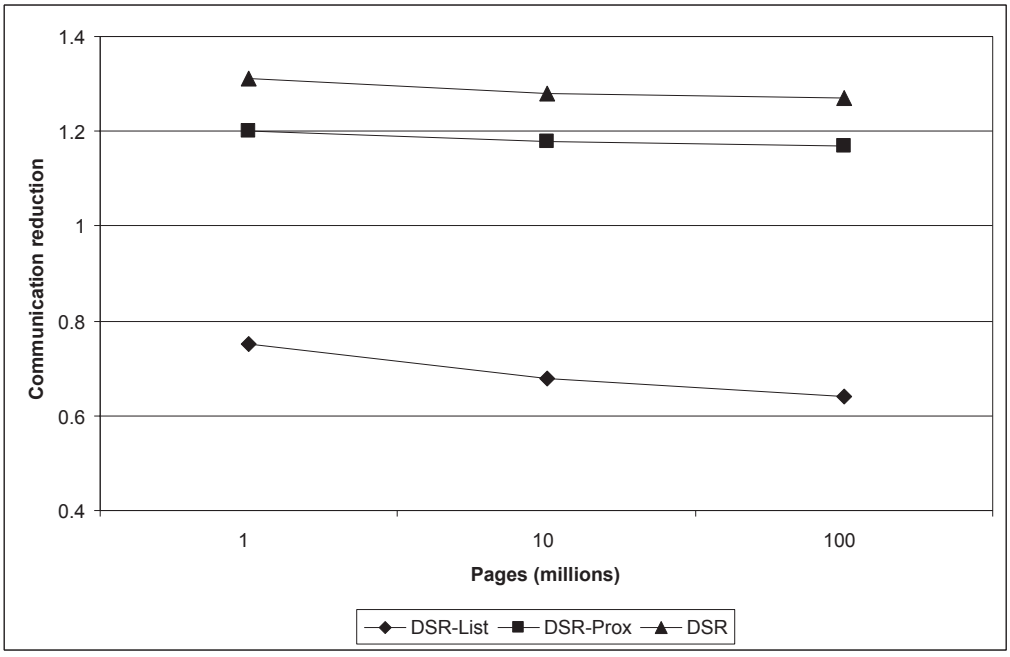

Figure 4.3: Distributed vs. single-server ranking in a full DPS system - reduction factor of network consumption (higher is better).

Data in Table A.17.

The results summarized in Figure 4.3 are the relative improvement of DSR, DSR-Prox and DSR-List over SR, when used in a full 1,000-server DPS system (computed as the inverse ratio of network consumption - higher is better, 1 is neutral).

- The document routing part is identical in SR and DSR-Prox (closest publish-subscribe server), and slightly more expensive in DPS (the dense components are assigned to servers close to the largest publishers). The average text document size in our data set is 25 kilobytes.

- While DSR-Prox is slightly faster than SR, is is outperformed by DSR due to the reduced rank lookup communication costs.

  During the rank lookup and computation, SR sends all links to the dedicated rank server and receives the page's rank value (average of 20 links per page). DSR and DSR-Prox query other DPS servers with a short list of the document IDs for their per-link contribution. The number of servers contacted by DSP for each rank computation is about half of those contacted by DSR-Prox.

- It is not surprising that DSR-List is even much slower than SR, given the high communication load required to both route documents over longer paths and look up individual inlink rank contributions from remote ranking servers during page rank computations.

The raw data in Table B.7 (Appendix B) also shows that there is little difference when the data set increases – the main effect is a higher number of inlinks, which increases the size of the rank lookup message in SR and the number of messages in the various DSR setups.

The network load is well distributed among multiple publish-subscribe servers in all setups (even the single-rank-server SR setup uses multiple publish-subscribe servers). The results in Figure 4.3 show that in addition to this we can further reduce network traffic by distributing the rank computation. Of course, the much bigger numerical impact of distributed rank computation is the significant speedup when computing site ranks. Even if this operation is performed off-line and does not have a big impact on run-time performance, by making it faster we can run it more often and have more up-to-date document rank values, which is very important for the overall quality of the persistent search system.

4.4 Summary

To fully distribute persistent search, we developed a scalable link-based relevance algorithm to replace the initial single-server component proposed in DPS (Chapter 3). Our main contribution is taking advantage of the fact that the number and size of rank lookup messages performed during the iterative rank computation can be significantly lowered by partitioning the document base into dense components – most links are inside, not between components. This approach is superior to interval- or proximity-based partitioning schemes, both for the standalone rank computation and for the combined publication crawling and rank lookup in DPS. While our design is centered around distributing SiteRank, the same approach can be applied to PageRank with similar results.

The current proposal still relies on a single-server approach to perform re-partitioning of the document set when a large enough fraction of the links have changed. While this operation is performed in the background and has no direct effect on the reliability and performance of the rank computation and lookup, we feel that a fully distributed algorithm would further increase the scale of the system as it would no longer require even temporary collection of all the site link information at single location.

To improve performance, a number of other adjustments to Distributed SiteRank are possible – we list a few of them in Section 5.2.1.

Similar to PageRank and SiteRank, both Distributed SiteRank and Distributed PageRank can be used in persistent search as proposed in this chapter, and also in regular (non-persistent) search systems that need a highly-scalable ranking component.

Chapter 5

Summary

Building a high-quality distributed persistent search system is a difficult task, addressing many different components and their complex interactions. The main focus of our work has been developing a scalable persistent search system, with special attention to the quality and freshness of the results. We concentrated our efforts primarily on improving link-based relevance by reducing its bias against new pages and making it scale well over very large sets of documents. To achieve our main goal and to properly put our ranking work in context we also designed, implemented and evaluated a simulation system for highly-scalable distributed publish-subscribe architecture-based persistent search; while this system is lacking user interaction and crawling components, its document and subscription handling, ranking, matching and notification components are fully functional.

5.1 Results

The first problem we addressed was reducing the bias against new pages in page authority scores. Traditionally, the page rank computation is based on a single signal - links between pages. New pages however have very limited support; to better predict the popularity of all pages (new and old), we proposed to also take into consideration the ranks of sites and the pages' position within sites, in addition to the individual page links. Tests on synthetic data sets and consecutive web crawls proved that our proposed algorithm (SiteRank) significantly outperforms PageRank in predicting the rank of new pages, while maintaining the PageRank-induced ordering between old pages, especially among top pages with high number of inlinks.

A positive side-effect of computing ranks over the site link-graph is a significant speedup of the algorithm, allowing frequent refreshes of the page ranks.

We must point out that SiteRank is in no way specific to persistent search systems. It is a general-purpose link-based relevance computation that can be used in any search engine. Both the improved prediction of webpages' popularity and the performance improvement are great assets in any user-oriented search service, providing high-quality, fresh results for any query. As an added bonus, the various proposals to speed up PageRank that were mentioned in section 2.1.2 can be applied to the site rank computation phase of SiteRank for extra performance boost.

Returning to our original goal, we designed and developed a highly-scalable, general-purpose keyword-based persistent search system (DPS) built on a distributed publish-subscribe architecture; this system integrates SiteRank as a single-server relevance computation component. DPS tries to address the scaling problems posed by large publication sets by eliminating internal publication traffic within the distributed service, limiting it to the edges by placing publish-subscribe servers close to publishers. As expected, this reduces overall publication traffic and evenly distributes the publication load among servers, compared to single-server and existing distributed architectures. The total message load in a distributed system is higher than for a single server (since we added inter-server communication to the existing end-user notifications), but this load is divided among multiple servers, and each message travels significantly fewer hops. Of course, reducing publication communication costs requires that we replicate subscriptions at all servers, but they are smaller and less dynamic than publications, so the added cost is much smaller than the performance gains.

In an otherwise fully distributed system, computing SiteRank at a single location constitutes a single point of failure, and limits scalability. To efficiently distribute this computation and keep inter-server traffic low, we partitioned the link graph based on dense components. This method introduces the lowest possible network load during rank computation since most of the rank lookups are local; to use the distributed ranking component in the DPS persistent search system, publish-subscribe and ranking servers are collocated on the same nodes and assigned the same dense component, otherwise we would introduce very high inter-server document traffic between matching and ranking. Each link sub-graph is assigned to the server that minimizes the sum of all publication paths from sites in that subgraph to the selected server; this keeps rank computation fast while only slightly slowing down publication processing, since dense components are usually geographically dense as well.

Similarly to SiteRank, Distributed SiteRank is not specific to DPS - it can be used as a highly scalable ranking component in any search system.

5.2 Future directions

5.2.1 SiteRank

As summarized in the previous section, we developed novel single-server and distributed link-based relevance algorithms that provide high-quality rank values quickly, and we integrated them in a scalable persistent search system.

While Distributed SiteRank is much faster than single-location based alternatives, it still requires the computing servers to be synchronized at each iteration. The dense components should be as close in size as possible, but there are no guarantees to this unless the data set is very large. Even in this case, any difference in the number of links (inter- or intra- dense components) results in load imbalance between the servers. To eliminate waiting periods for the faster / less loaded servers, one can try an asynchronous version of Distributed SiteRank by keeping track of the site ranks for multiple (potentially all) iterations. Eliminating the need for synchronization after each iteration would allow all servers to proceed through the rank computation at their own pace, effectively freeing up faster / less loaded servers for other work by not making them wait for the slower / more loaded servers. Of course, a thorough evaluation of this approach is necessary to determine the right tradeoff between additional storage and potentially improved performance.

A more invasive synchronization requirement in the Distributed SiteRank algorithm is the re- partitioning performed at an elected leader which needs to collect all site link information. To fully distribute the persistent search system and completely eliminate the need to even temporarily collect any link information at a single location, we should perform the partitioning operation in a distributed manner too.

Another promising direction for future work is using SiteRank and Distributed SiteRank in a traditional search system.

- SiteRank can easily be substituted for PageRank in any search system that does not rely on the exact ranks currently produced by PageRank. The performance improvement is significant and would allow more frequent re-computation of the page ranks, and the better handling of new pages can

help to promote new, high quality pages over old pages which accumulated large inlink sets over a long period of time. Both these aspects would be especially useful in search engines operating on highly-dynamic document sets, where new pages from high-quality sites are valued better than old pages with many links (e.g. news or blog search engines [22]).

- Distributed SiteRank can also be used in larger scale traditional search systems. Additionally, the distributed method proposed for SiteRank can be applied with minimal changes to PageRank. Either Distributed SiteRank or Distributed PageRank would allow crawling and rank computation in large traditional search systems to scale better, but for full decentralization a number of other issues should be resolved:
  - Relevance scores need statistics computed over all pages in the repository (e.g. inverse document frequency), which are difficult to gather in a distributed system.
  - While routing small queries to all document processing servers is not too expensive, the result sets returned by all the servers need to be combined before being sent to the user.

    Sending the user all the different untrimmed result sets could overwhelm the user (and the network) and/or promote weak results. Also, a lower limit on the relevance score cannot be imposed when the query is issued, since this might lead to very large variations in the size of the result set:
    - For some queries, many documents score well and we return a huge result set.
    - Other queries might have very weak support in all documents but the user might find some of them useful, however they score under the threshold and never be sent to the user.

    One promising idea would be to dynamically adjust the relevance threshold based on term frequency statistics and results counts for previously issued similar queries.
  - Alternatively, one could non-trivially forward the query from server to server along with the current result set, at each new server the local result set being combined with the incoming global temporary result set. This approach is more likely to return the best result set to the user, but the sequential query processing severely limits the scale of the system by increasing the overall latency significantly.
  - A good alternative to sequential query processing and result set combination is a hierarchical ordering of the servers that would reduce the query processing to logarithmic length from the

linear cost of the sequential version. While this approach introduces the additional cost of maintaining the hierarchical overlay, the significant reduction in latency in large server sets should more than make up for this cost.

Distributed SiteRank/PageRank was designed for a geographically distributed system, however it can be be applied with similar success to a parallel setup. Each dense component in the link graph is assigned to a node in the parallel system, and the computation is performed with very little communication between the nodes, compared to a more naive assignment (e.g. alphabetical).

5.2.2 Distributed Persistent Search

While we extensively tested the Distributed Persistent Search system and Distributed SiteRank in a simulator, the most accurate performance test would require that servers in our distributed system are distributed geographically. Short of deploying a user-facing persistent search service, wide-area distribution could be tested over Planet-Lab [42].

Making such a large scale service available to the public would require many additional improvements (of course, we should make use of existing work as much as possible in each of these areas):

- Users need access to manage subscriptions, and this interface should include some form of authentication.
- Even in a distributed system, acquiring new documents poses many problems; further work would be needed in setting up a crawling infrastructure that would minimize the load it places on the network while accurately predicting new publications so they could be served quickly.
- Simple binary matching between subscriptions and publications is not expressive enough for practical use. Useful extensions include a more complex query grammar, stemming, synonyms.
- Using global document information would improve TF-IDF scores, but requires vocabulary synchronization across all publish-subscribe servers.
- Exact duplicate detection eliminates some unnecessary notifications, but we should filter out near-duplicates too (e.g. the same story from a news agency that appears on multiple sites only differs in boilerplate text and links).

5.3 Publications

Our work was published in:

- E. R. Schmidt and J. P. Singh. Internet-scale distributed persistent search. NY Area DB/IR Day Poster Session, 2005. [50]
- E. R. Schmidt and J. P. Singh. SiteRank: Link-based relevance computation for persistent search. Technical report, Department of Computer Science, Princeton University, 2006. [52]
- E. R. Schmidt and J. P. Singh. Distributed citation-based authority in a large-scale persistent search system. The Ninth International Workshop on the Web and Databases (WebDB) (accepted, not presented due to scheduling conflict), 2006. [51]

Appendix A

Experimental results

Percentage of links removed	Method		
	PageRank	**HostRank**	**SiteRank**
0%	0.0%	7.1%	3.7%
1%	10.1%	7.3%	3.7%
25%	20.9%	9.9%	4.2%
50%	51.3%	20.3%	8,5%

Table A.1: SiteRank: KDist over all page pairs, complete graph vs. reduced links.
Plotted in Figure 2.1.

Percentage of links removed	Method		
	PageRank	**HostRank**	**SiteRank**
1%	12.4%	4.2%	2.2%
25%	28.3%	6.9%	2.5%
50%	60.2%	13.2%	4.7%

Table A.2: SiteRank: KDist over pairs of new pages, complete graph vs. reduced links.
Plotted in Figure 2.2.

Percentage of links removed	Method		
	PageRank	**HostRank**	**SiteRank**
1%	15.1%	3.9%	1.9%
25%	33.7%	6.1%	2.2%
50%	68.5%	12.3%	4.1%

Table A.3: SiteRank: KDist over pairs of (new,old) pages, complete graph vs. reduced links.
Plotted in Figure 2.3.

Method	Page pairs		
	all pairs	**(new,new) pages**	**(new,old) pages**
PageRank	15.2%	19.3%	26.8%
HostRank	7.8%	5.4%	4.9%
SiteRank	3.9%	2.4%	2.1%

Table A.4: SiteRank: KDist over all page pairs, two consecutive crawls.
Plotted in Figure 2.4.

Pages	Page pairs		
	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 A.5: SiteRank: KDist of SiteRank vs. PageRank page order change.
Plotted in Figure 2.5.

Pages	Speedup	Average pages/site	PageRank iterations	SiteRank iterations
100,000	800	1,000	25	16
1,000,000	900	1,000	29	19
10,000,000	2,100	2,500	30	20

Table A.6: SiteRank vs PageRank speedup factor.
Plotted in Figure 2.6.

Pages	Speedup	Average pages/site	PageRank iterations	SiteRank iterations
1,000,000	900	1,000	29	19
1,000,000	1,700	2,000	28	17
10,000,000	2,400	2,500	38	17

Table A.7: SiteRank vs PageRank speedup factor on 1 million-page data sets, varying average site size.
Plotted in Figure 2.7.

Number of servers	DCC vs. 1S	DPS vs. 1S
10	3.79	7.15
100	35.27	65.66
1,000	336.70	568.18

Table A.8: DPS: Reduction factor in the number of hops to push all documents to the matching server(s), DCC and DPS vs. 1S.
Plotted in Figure 3.9, raw data in Table B.1.

Number of servers	DCC vs. 1S	DPS vs. 1S
10	5	10
100	50	100
1,000	500	1,000

Table A.9: DPS: Reduction factor in the number of incoming publications per server, DCC and DPS vs. 1S.
Plotted in Figure 3.10, raw data in Table B.2.

Number of servers	Publication types		
	all pubs	**10-pubs**	**1,000-pubs**
10	1.009	1.502	1.005
100	1.092	1.887	1.048
1,000	1.931	1.982	1.513

Table A.10: DPS: Increase factor in total number of messages sent by all servers for one publication - DPS sends more messages than 1S for each match.
Plotted in Figure 3.11, raw data in Table B.3.

Number of servers	Publication types		
	all pubs	**10-pubs**	**1,000-pubs**
10	9.91	6.66	9.95
100	91.58	52.99	95.42
1,000	517.87	504.54	660.94

Table A.11: DPS: Reduction factor in number of messages sent per server for one publication - DPS divides the load among multiple servers.
Plotted in Figure 3.12, raw data in Table B.4.

Number of servers	Publication types		
	all pubs	**10-pubs**	**1,000-pubs**
10	9.35	9.90	9.71
100	90.91	90.91	100.00
1,000	500.00	500.00	500.00

Table A.12: DPS: Reduction factor in total network hops per matched publication - each message travels much less in DPS.
Plotted in Figure 3.13, raw data in Table B.5.

Number of servers	Rank server traffic as a percentage of total publication traffic
10	4.62%
100	3.91%
1,000	3.03%

Table A.13: DPS: Rank server communication is a small percentage of overall publication traffic.
Plotted in Figure 3.14.

Number of servers	Subscription cost increase factor, DPS vs. 1S
10	1.105
100	10.013
1,000	100.001

Table A.14: DPS: Subscription maintenance overhead increases with system size.
Plotted in Figure 3.15, raw data in Table B.6.

Number of servers	Number of documents (millions)		
	1	**10**	**100**
10	8.2	8.7	9.2
100	81.2	84.7	90.7
1000	805.5	830.8	878.2

Table A.15: Distributed SiteRank: DSR vs. SR speedup factor (total time), varying number of rank servers. Plotted in Figure 4.1.

Distributed method	Number of documents (millions)		
	1	**10**	**100**
DSR	805.5	830.8	878.2
DSR-Prox	801.0	793.3	790.8
DSR-List	755.1	748.7	731.1

Table A.16: Distributed SiteRank: Distributed vs. single-server speedup factor (total time), various distributed methods.

Plotted in Figure 4.2.

Distributed method	Number of documents (millions)		
	1	**10**	**100**
DSR	1.31	1.28	1.27
DSR-Prox	1.20	1.18	1.17
DSR-List	0.75	0.68	0.64

Table A.17: Distributed SiteRank: Distributed vs. single-server ranking in a full 1,000-server DPS system - reduction factor of network consumption (higher is better, 1 is neutral).

Plotted in Figure 4.3, raw data in TableB.7.

Appendix B

Raw data

Number of servers	DCC	DPS
1 (1S)	487.50 ± 412.17	
10	181.40 ± 128.31	96.15 ± 50.75
100	21.30 ± 12.93	10.47 ± 4.42
1,000	2.04 ± 2.28	1.21 ± 0.61

Table B.1: DPS: Average number of hops (and standard deviation) to push a document to its matching server.

Processed data in Table A.8, plotted in Figure 3.9.

Number of servers	DCC	DPS
1 (1S)	100,000,000	
10	20,000,000 ± 28,109,135	10,000,000 ± 1,382
100	2,000,000 ± 9,898,990	1,000,000 ± 853
1,000	200,000 ± 3,159,112	100,000 ± 591

Table B.2: DPS: Number of incoming publications per server (average and standard deviation).
Processed data in Table A.9, plotted in Figure 3.10.

	Publication types		
Number of servers	all pubs	10-pubs	1,000-pubs
1 (1S)	99.41	10.12	998.37
10	100.30	15.20	1,003.36
100	108.56	19.10	1,046.29
1,000	191.96	20.06	1,510.53

Table B.3: DPS: Total number of notification messages sent by all servers for one matched publication.
Processed data in Table A.10, plotted in Figure 3.11.

Number of servers	Publication types		
	all pubs	**10-pubs**	**1,000-pubs**
1 (1S)	99.41	10.12	998.37
10	10.03	1.52	100.34
100	1.09	0.19	10.46
1,000	0.19	0.02	1.51

Table B.4: DPS: Number of messages sent per server for one matched publication.
Processed data in Table A.11, plotted in Figure 3.12.

Number of servers	Publication types		
	all pubs	**10-pubs**	**1,000-pubs**
1 (1S)	68,344.38	6,957.50	686,379.38
10	7,309.56	702.78	70,687.89
100	751.78	76.53	6,863.79
1,000	136.69	13.92	1,372.76

Table B.5: DPS: Total network hops over all notification messages for one matched publication.
Processed data in Table A.12, plotted in Figure 3.13.

Number of servers	Subscription cost
1 (1S)	13.75 ± 8.24
10	15.19 ± 9.11
100	137.68 ± 82.54
1,000	$1{,}375.01 \pm 824.35$

Table B.6: DPS: Subscription maintenance overhead - network utilization in kilobytes.
Processed data in Table A.14, plotted in Figure 3.15.

Ranking method	Number of documents (millions)		
	1	**10**	**100**
SR	136.25 ± 10.31	141.25 ± 12.09	151.25 ± 17.23
DSR	104.01 ± 9.73	110.35 ± 10.65	119.09 ± 14.08
DSR-Prox	113.54 ± 9.97	119.70 ± 11.16	129.27 ± 15.11
DSR-List	181.67 ± 51.14	207.72 ± 70.02	236.33 ± 108.82

Table B.7: Distributed SiteRank: Average and standard deviation of the network consumption (sum of message size over traversed network hops, in kilobytes) required to publish and rank a document in 1,000-server DPS, various ranking setups.
Processed data in Table A.17, plotted in Figure 4.3.

Bibliography

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

[3] M. Bergman. The Deep Web: Surfacing hidden value. *Journal of Electronic Publishing, University of Michigan University Library*, 7(1), 2001.

[4] The Official Google Blog. We knew the web was big... http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

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

[6] K. D. Bollacker, S. Lawrence, and C. L. Giles. A system for automatic personalized tracking of scientific literature on the Web. In *Proceedings of The 4th ACM Conference on Digital Libraries*, 1999.

[7] 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.

[8] L. F. Cabrera, M. B. Jones, and M. Theimer. Herald: Achieving a global event notification service. In *Proceedings of The Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[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] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[11] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM*, pages 163–174, 2003.

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

[13] J. Cho, S. Roy, and R. E. Adams. Page quality: In search of an unbiased Web ranking. In *Proceedings of ACM SIGMOD*, 2005.

[14] CNN. Alerts. http://www.cnn.com/youralerts/.

[15] Cuil. http://www.cuil.com/.

[16] Carzaniga et al. SIENA project homepage. http://www.inf.unisi.ch/carzaniga/siena/.

[17] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[18] B. Fox. Google searches for quality not quantity. *New Scientist*, (2497):24, 2005.

[19] Z. Ge, M. Adler, J. F. Kurose, D. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. Technical report, University of Massachusetts, 2001.

[20] Google. http://www.google.com/.

[21] Google. Alerts. http://www.google.com/alerts.

[22] Google. News. http://news.google.com/.

[23] Google. Reader. http://www.google.com/reader.

[24] J. Greifenberg and D. Kutscher. Efcient publish/subscribe-based multicast for opportunistic networking with self-organized resource utilization. In *Proceedings of The 22nd International Conference on Advanced Information Networking and Applications*, 2008.

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

[26] 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.

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

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

[29] A. Hinze and D. Faensen. Unified model of Internet scale alerting services. In *Proceedings of The 5th International Computer Science Conference (ICSC)*, pages 284–293, 1999.

[30] J. Hirai, S. Raghavan, H. Garcia-Molina, and A. Paepcke. WebBase: A repository of web pages. In *Proceedings of The Ninth International World Wide Web Conference*, 2000.

[31] 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.

[32] 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.

[33] KaZaA. http://www.kazaa.com/.

[34] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, 1998.

[35] E. Korach, S. Kutten, and S. Moran. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems*, 12(1):84–101, 1990.

[36] R. Lempel and S. Moran. The stochastic approach for link-structure analysis (SALSA) and the TKS effect. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 133(1-6):387–401, 2000.

[37] 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.

[38] T. Milo, T. Zur, and E. Verbin. Boosting topic-based publish-subscribe systems with dynamic clustering. In *Proceedings of The 2007 ACM SIGMOD International Conference on Management of Data*, 2007.

[39] New York Times. http://www.nytimes.com/.

[40] New York Times. RSS feeds. http://www.nytimes.com/services/xml/rss/.

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

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

[43] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of The 2001 ACM SIGCOMM Conference*, 2001.

[44] Cuil Press Release. Cuil launches biggest search engine on the web. http://www.cuil.com/info/news_press/.

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

[46] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

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

[48] 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.

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

[50] E. R. Schmidt and J. P. Singh. Internet-scale distributed persistent search. In *NY Area DB/IR Day Poster Session*, 2005.

[51] E. R. Schmidt and J. P. Singh. Distributed citation-based authority in a large-scale persistent search system. In *The Ninth International Workshop on the Web and Databases (WebDB) (accepted, not presented due to scheduling conflict)*, 2006.

[52] E. R. Schmidt and J. P. Singh. SiteRank: Link-based relevance computation for persistent search. Technical Report 745-06, Department of Computer Science, Princeton University, 2006.

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

[54] M. Stoer and F. Wagner. A simple min-cut algorithm. *Journal of the ACM*, 44(4):585–591, 1997.

[55] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnam. Chord: Scalable peer-to-peer lookup service for internet applications. In *Proceedings of The 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.

[56] 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.

[57] TechCrunch. Cuil exits stealth mode with a massive search engine. http://www.techcrunch.com/2008/07/27/cuill-launches-a-massive-search-engine/.

[58] 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.

[59] Wikipedia. Gnutella. http://en.wikipedia.org/wiki/Gnutella.

[60] Wikipedia. PubSub.com. http://en.wikipedia.org/wiki/PubSub_(website).

[61] J. Wu and K. Aberer. Using SiteRank for decentralized computation of Web document ranking. In *Proceedings of The 3rd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 265–274, 2004.

[62] 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.

[63] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *Proceedings of ICDCS*, 2003.

[64] G. Xue, Q. Yang, H. Zeng, Y. Yu, and Z. Chen. Exploiting the hierarchical structure for link analysis. In *Proceedings of SIGIR'05*, pages 186–193, 2005.

[65] Yahoo! http://www.yahoo.com/.

[66] Yahoo! Alerts. http://alerts.yahoo.com/.

[67] T. W. Yan and H. Garcia-Molina. SIFT - a tool for wide-area information dissemination. In *Proceedings of USENIX 1995*, pages 177–186, 1995.

[68] E. W. Zegura, K. Calvert, and M. J. Donahoo. A quantitative comparison of graph-based models for internet topology. *IEEE/ACM Transactions on Networking*, 1997.

[69] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, 2001.

[70] Ziff Davis. Enterprise RSS feeds. http://www.eweek.com/c/a/Past-News/Ziff-Davis-Enterprise-RSS-Feeds/.