WELCOME TO THE GATHERING PLACE

Computer Networks and ISDN Systems 30 (1998) 107–117

COMPUTER NETWORKS and ISDN SYSTEMS

The anatomy of a large-scale hypertextual Web search engine [1]

Sergey Brin [2], Lawrence Page [*,2]

Computer Science Department, Stanford University, Stanford, CA 94305, USA

Abstract

In this paper, we present Google, a prototype of a large-scale search engine which makes heavy use of the structure present in hypertext. Google is designed to crawl and index the Web efficiently and produce much more satisfying search results than existing systems. The prototype with a full text and hyperlink database of at least 24 million pages is available at http://google.stanford.edu/

To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of Web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the Web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and Web proliferation, creating a Web search engine today is very different from three years ago. This paper provides an in-depth description of our large-scale Web search engine — the first such detailed public description we know of to date.

Apart from the problems of scaling traditional search techniques to data of this magnitude, there are new technical challenges involved with using the additional information present in hypertext to produce better search results. This paper addresses this question of how to build a practical large-scale system which can exploit the additional information present in hypertext. Also we look at the problem of how to effectively deal with uncontrolled hypertext collections where anyone can publish anything they want.

Keywords: World Wide Web; Search engines; Information retrieval; PageRank; Google

1. Introduction

The Web creates new challenges for information retrieval. The amount of information on the Web is growing rapidly, as well as the number of new users inexperienced in the art of Web research. People are likely to surf the Web using its link graph, often starting with high quality human maintained indices such as **Yahoo!** [3] or with search engines. Human maintained lists cover popular topics effectively but are subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topics. Automated search engines that rely on keyword matching usually return too many low quality matches. To make matters worse, some advertisers attempt to gain people's attention by taking measures meant to mislead

* Corresponding author.

[1] There are two versions of this paper — a longer full version and a shorter printed version. The full version is available on the Web and the conference CD-ROM.

[2] E-mail: {sergey, page}@cs.stanford.edu

[3] http://www.yahoo.com/

0169-7552/98/$19.00
PII S0169-7552(98)00110-X

automated search engines. We have built a large-scale search engine which addresses many of the problems of existing systems. It makes especially heavy use of the additional structure present in hypertext to provide much higher quality search results. We chose our system name, Google, because it is a common spelling of googol, or 10100 and fits well with our goal of building very large-scale search engines.

1.1. Web search engines — scaling up: 1994–2000

Search engine technology has had to scale dramatically to keep up with the growth of the Web. In 1994, one of the first Web search engines, the World Wide Web Worm (WWWW) [6] had an index of 110,000 Web pages and Web accessible documents. As of November, 1997, the top search engines claim to index from 2 million (WebCrawler) to 100 million Web documents (from **Search Engine Watch** [4]). It is foreseeable that by the year 2000, a comprehensive index of the Web will contain over a billion documents. At the same time, the number of queries search engines handle has grown incredibly too. In March and April 1994, the World Wide Web Worm received an average of about 1500 queries per day. In November 1997, Altavista claimed it handled roughly 20 million queries per day. With the increasing number of users on the Web, and automated systems which query search engines, it is likely that top search engines will handle hundreds of millions of queries per day by the year 2000. The goal of our system is to address many of the problems, both in quality and scalability, introduced by scaling search engine technology to such extraordinary numbers.

1.2. Google: scaling with the Web

Creating a search engine which scales even to today's Web presents many challenges. Fast crawling technology is needed to gather the Web documents and keep them up to date. Storage space must be used efficiently to store indices and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently. Queries must be handled quickly, at a rate of hundreds to thousands per second.

These tasks are becoming increasingly difficult as the Web grows. However, hardware performance and cost have improved dramatically to partially offset the difficulty. There are, however, several notable exceptions to this progress such as disk seek time and operating system robustness. In designing Google, we have considered both the rate of growth of the Web and technological changes. Google is designed to scale well to extremely large data sets. It makes efficient use of storage space to store the index. Its data structures are optimized for fast and efficient access (see Section 4.2). Further, we expect that the cost to index and store text or HTML will eventually decline relative to the amount that will be available (see Appendix B in the full version). This will result in favorable scaling properties for centralized systems like Google.

1.3. Design goals

1.3.1. Improved search quality

Our main goal is to improve the quality of Web search engines. In 1994, some people believed that a complete search index would make it possible to find anything easily. According to **Best of the Web 1994 — Navigators** [5], "The best navigation service should make it easy to find almost anything on the Web (once all the data is entered)." However, the Web of 1997 is quite different. Anyone who has used a search engine recently, can readily testify that the completeness of the index is not the only factor in the quality of search results. "Junk results" often wash out any results that a user is interested in. In fact, as of November 1997, only one of the top four commercial search engines finds itself (returns its own search page in response to its name in the top ten results). One of the main causes of this problem is that the number of documents in the indices has been increasing by many orders of magnitude, but the user's ability to look at documents has not. People are still only willing to look at the first few tens of results. Because of this, as the collection size grows, we need tools that have very high precision (number of relevant documents returned, say in the top tens of results). Indeed, we want our notion of "relevant" to only include the

[4] http://www.searchenginewatch.com/

[5] http://botw.org/1994/awards/navigators.html

very best documents since there may be tens of thousands of slightly relevant documents. This very high precision is important even at the expense of recall (the total number of relevant documents the system is able to return). There is quite a bit of recent optimism that the use of more hypertextual information can help improve search and other applications [4,9,12,3]. In particular, link structure [7] and link text provide a lot of information for making relevance judgments and quality filtering. Google makes use of both link structure and anchor text (see Sections 2.1 and 2.2).

1.3.2. Academic search engine research

Aside from tremendous growth, the Web has also become increasingly commercial over time. In 1993, 1.5% of Web servers were on .com domains. This number grew to over 60% in 1997. At the same time, search engines have migrated from the academic domain to the commercial. Up until now most search engine development has gone on at companies with little publication of technical details. This causes search engine technology to remain largely a black art and to be advertising oriented (see Appendix A in the full version). With Google, we have a strong goal to push more development and understanding into the academic realm.

Another important design goal was to build systems that reasonable numbers of people can actually use. Usage was important to us because we think some of the most interesting research will involve leveraging the vast amount of usage data that is available from modern Web systems. For example, there are many tens of millions of searches performed every day. However, it is very difficult to get this data, mainly because it is considered commercially valuable.

Our final design goal was to build an architecture that can support novel research activities on large-scale Web data. To support novel research uses, Google stores all of the actual documents it crawls in compressed form. One of our main goals in designing Google was to set up an environment where other researchers can come in quickly, process large chunks of the Web, and produce interesting results that would have been very difficult to produce otherwise. In the short time the system has been up, there have already been several papers using databases generated by Google, and many others are underway. Another goal we have is to set up a Spacelab-like environment where researchers or even students can propose and do interesting experiments on our large-scale Web data.

2. System features

The Google search engine has two important features that help it produce high precision results. First, it makes use of the link structure of the Web to calculate a quality ranking for each Web page. This ranking is called PageRank and is described in detail in [7]. Second, Google utilizes links to improve search results.

2.1. PageRank: bringing order to the Web

The citation (link) graph of the Web is an important resource that has largely gone unused in existing Web search engines. We have created maps containing as many as 518 million of these hyperlinks, a significant sample of the total. These maps allow rapid calculation of a Web page's "PageRank", an objective measure of its citation importance that corresponds well with people's subjective idea of importance. Because of this correspondence, PageRank is an excellent way to prioritize the results of Web keyword searches. For most popular subjects, a simple text matching search that is restricted to Web page titles performs admirably when PageRank prioritizes the results (demo available at google.stanford.edu). For the type of full text searches in the main Google system, PageRank also helps a great deal.

2.1.1. Description of PageRank calculation

Academic citation literature has been applied to the Web, largely by counting citations or backlinks to a given page. This gives some approximation of a page's importance or quality. PageRank extends this idea by not counting links from all pages equally, and by normalizing by the number of links on a page. PageRank is defined as follows:

We assume page A has pages T1...Tn which point to it (i.e., are citations). The parameter d is a damping factor which can be set between 0 and 1. We usually set d to 0.85. There are more details

about d in the next section. Also C(A) is defined as the number of links going out of page A. The PageRank of a page A is given as follows:

$$PR(A) = (1-d) + d\left(\frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)}\right)$$

Note that the PageRanks form a probability distribution over Web pages, so the sum of all Web pages' PageRanks will be one.

PageRank or $PR(A)$ can be calculated using a simple iterative algorithm, and corresponds to the principal eigenvector of the normalized link matrix of the Web. Also, a PageRank for 26 million Web pages can be computed in a few hours on a medium size workstation. There are many other details which are beyond the scope of this paper.

2.1.2. Intuitive justification

PageRank can be thought of as a model of user behavior. We assume there is a "random surfer" who is given a Web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page. The probability that the random surfer visits a page is its PageRank. And, the d damping factor is the probability at each page the "random surfer" will get bored and request another random page. One important variation is to only add the damping factor d to a single page, or a group of pages. This allows for personalization and can make it nearly impossible to deliberately mislead the system in order to get a higher ranking. We have several other extensions to PageRank, again see [7].

Another intuitive justification is that a page can have a high PageRank if there are many pages that point to it, or if there are some pages that point to it and have a high PageRank. Intuitively, pages that are well cited from many places around the Web are worth looking at. Also, pages that have perhaps only one citation from something like the **Yahoo!** [6] homepage are also generally worth looking at. If a page was not high quality, or was a broken link, it is quite likely that Yahoo's homepage would not link to it. PageRank handles both these cases and everything in between by recursively propagating weights through the link structure of the Web.

[6] http://www.yahoo.com/

2.2. Anchor text

The text of links is treated in a special way in our search engine. Most search engines associate the text of a link with the page that the link is on. In addition, we associate it with the page the link points to. This has several advantages. First, anchors often provide more accurate descriptions of Web pages than the pages themselves. Second, anchors may exist for documents which cannot be indexed by a text-based search engine, such as images, programs, and databases. This makes it possible to return Web pages which have not actually been crawled. Note that pages that have not been crawled can cause problems, since they are never checked for validity before being returned to the user. In this case, the search engine can even return a page that never actually existed, but had hyperlinks pointing to it. However, it is possible to sort the results, so that this particular problem rarely happens.

This idea of propagating anchor text to the page it refers to was implemented in the World Wide Web Worm [6] especially because it helps search non-text information, and expands the search coverage with fewer downloaded documents. We use anchor propagation mostly because anchor text can help provide better quality results. Using anchor text efficiently is technically difficult because of the large amounts of data which must be processed. In our current crawl of 24 million pages, we had over 259 million anchors which we indexed.

3. Related work

Search research on the Web has a short and concise history. The World Wide Web Worm (WWWW) [6] was one of the first Web search engines. It was subsequently followed by several academic search engines, many of which are now public companies. Compared to the growth of the Web and the importance of search engines there are precious few documents about recent search engines [8]. According to Michael Mauldin (chief scientist, Lycos Inc.) [5], "the various services (including Lycos) closely guard the details of these databases". However, there has been a fair amount of work on specific features of search engines. Especially well represented

is work which can get results by post-processing the results of existing commercial search engines, or produce small scale "individualized" search engines. Finally, there has been a lot of research on information retrieval systems, especially on well controlled collections [11].

However, work on information retrieval has mostly been on fairly small, well controlled collections such as the Text Retrieval Conference [10]. Things that work well on TREC often do not produce good results on the Web. For example, the standard vector space model tries to return the document that most closely approximates the query, given that both query and document are vectors defined by their word occurrence. On the Web, this strategy often returns very short documents that are the query plus a few words. For example, we have seen a major search engine return a page containing only "Bill Clinton Sucks" and picture from a "Bill Clinton" query. Given examples like these, we believe that the standard information retrieval work needs to be extended to deal effectively with the Web.

The Web is a vast collection of completely uncontrolled heterogeneous documents. Documents vary significantly in language, format, and style. There can be many orders of magnitude of difference in two documents' size, quality, popularity, and trustworthiness. All of these are significant challenges to effective searching on the Web. They are somewhat mediated by the availability of auxiliary data such as hyperlinks and formatting and Google tries to take advantage of both of these.

4. System anatomy

4.1. Google architecture overview

In this section, we will give a high level overview of how the whole system works as pictured in Fig. 1. Further sections will discuss the applications and data structures not mentioned in this section. Most of Google is implemented in C or C++ for efficiency and can run in either Solaris or Linux.

In Google, the Web crawling (downloading of Web pages) is done by several distributed crawlers. There is a URLserver that sends lists of URLs to be fetched to the crawlers. The Web pages that are

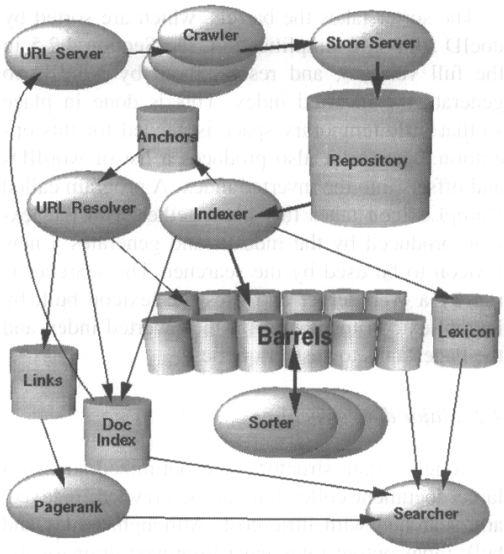

Fig. 1. High level Google architecture.

fetched are then sent to the storeserver. The storeserver then compresses and stores the Web pages into a repository. Every Web page has an associated ID number called a docID which is assigned whenever a new URL is parsed out of a Web page. The indexing function is performed by the indexer and the sorter. The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each document is converted into a set of word occurrences called hits. The hits record the word, position in document, an approximation of font size, and capitalization. The indexer distributes these hits into a set of "barrels", creating a partially sorted forward index. The indexer performs another important function. It parses out all the links in every Web page and stores important information about them in an anchors file. This file contains enough information to determine where each link points from and to, and the text of the link.

The URLresolver reads the anchors file and converts relative URLs into absolute URLs and in turn into docIDs. It puts the anchor text into the forward index, associated with the docID that the anchor points to. It also generates a database of links which are pairs of docIDs. The links database is used to compute PageRanks for all the documents.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.