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. © 1998 Published by Elsevier Science B.V. All rights reserved.

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 © 1998 Published by Elsevier Science B.V. All rights reserved.
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

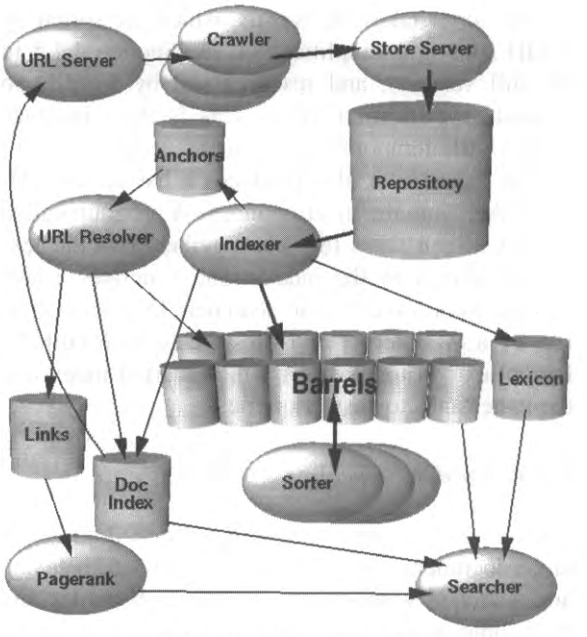

Fig. 1. High level Google architecture.

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.

The sorter takes the barrels, which are sorted by docID (this is a simplification, see Section 4.2.5 in the full version), and resorts them by wordID to generate the inverted index. This is done in place so that little temporary space is needed for this operation. The sorter also produces a list of wordIDs and offsets into the inverted index. A program called DumpLexicon takes this list together with the lexicon produced by the indexer and generates a new lexicon to be used by the searcher. The searcher is run by a Web server and uses the lexicon built by DumpLexicon together with the inverted index and the PageRanks to answer queries.

4.2. Major data structures

Google's data structures are optimized so that a large document collection can be crawled, indexed, and searched with little cost. Although, CPUs and bulk input output rates have improved dramatically over the years, a disk seek still requires about 10 ms to complete. Google is designed to avoid disk seeks whenever possible, and this has had a considerable influence on the design of the data structures. The full version of this paper contains a detailed discussion of all the major data structures. We only give a brief overview here.

Almost all of the data for Google is stored in Bigfiles which are virtual files we developed that can span multiple file systems and support compression. The raw HTML repository uses roughly half of the necessary storage. It consists of the concatenation of the compressed HTML of every page, preceded by a small header. The document index keeps information about each document. It is a fixed width ISAM (Index sequential access mode) index, ordered by docID. The information stored in each entry includes the current document status, a pointer into the repository, a document checksum, and various statistics. Variable width information such as URL and title is kept in a separate file. There is also an auxiliary index to convert URLs into docIDs. The lexicon has several different forms for different operations. They all are memory-based hash tables with varying values attached to each word.

A hit list corresponds to a list of occurrences of a particular word in a particular document including position, font, and capitalization information. Hit lists account for most of the space used in both the forward and the inverted indices. Because of this, it is important to represent them as efficiently as possible. We considered several alternatives for encoding position, font, and capitalization — simple encoding (a triple of integers), a compact encoding (a hand optimized allocation of bits), and Huffman coding. In the end we chose a hand optimized compact encoding since it required far less space than the simple encoding and far less bit manipulation than Huffman coding.ding. Our compact coding uses two bytes for every hit. The details of this coding are in the full version of this paper. The length of a hit list is stored before the hits themselves. To save space, the length of the hit list is combined with the wordID in the forward index and the docID in the inverted index.

The forward index is actually already partially sorted. It is stored in a number of barrels (we used 64). Each barrel holds a range of wordIDs. If a document contains words that fall into a particular barrel, the docID is recorded into the barrel, followed by a list of wordIDs with hitlists which correspond to those words. This scheme requires slightly more storage because of duplicated docIDs but the difference is very small for a reasonable number of buckets and saves considerable time and coding complexity in the final indexing phase done by the sorter. The inverted index consists of the same barrels as the forward index, except that they have been processed by the sorter. For every valid wordID, the lexicon contains a pointer into the barrel that wordID falls into. It points to a list of docIDs together with their corresponding hit lists. This list is called a doclist and represents all the occurrences of that word in all documents.

An important issue is in what order the docIDs should appear in the doclist. One simple solution is to store them sorted by docID. This allows for quick merging of different doclists for multiple word queries. Another option is to store them sorted by a ranking of the occurrence of the word in each document. This makes answering one word queries trivial and makes it likely that the answers to multiple word queries are near the start. However, merging is much more difficult. Also, this makes development much more difficult in that a change to the ranking function requires a rebuild of the index. We chose a compromise between these options, keeping two sets of inverted barrels — one set for hit lists which

include title or anchor hits and another set for all hit lists. This way, we check the first set of barrels first and if there are not enough matches within those barrels we check the larger ones.

4.3. Crawling the Web

Running a Web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of Web servers and various name servers which are all beyond the control of the system.

In order to scale to hundreds of millions of Web pages, Google has a fast distributed crawling system. A single URLserver serves lists of URLs to a number of crawlers (we typically ran about 3). Both the URLserver and the crawlers are implemented in Python. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve Web pages at a fast enough pace. At peak speeds, the system can crawl over 100 Web pages per second using four crawlers. A major performance stress is DNS lookup so each crawler maintains a DNS cache. Each of the hundreds of connections can be in a number of different states: looking up DNS, connecting to host, sending request, and receiving response. These factors make the crawler a complex component of the system. It uses asynchronous IO to manage events, and a number of queues to move page fetches from state to state.

The more than half million servers that we crawl are run by tens of thousands of Webmasters. As a result crawling the Web involves interacting with a fair number of people. Almost daily we receive emails like "Wow, you looked at a lot of pages from my Web site. How did you like it?" Other interactions involve copyright issues and obscure bugs which may only arise on one page out of ten million. Since large complex systems such as crawlers will invariably cause problems, there needs to be significant resources devoted to reading the email and solving these problems as they come up.

4.4. Searching

The goal of searching is to provide quality search results efficiently. Many of the large commercial search engines seemed to have made great progress in terms of efficiency. Therefore, we have focused more on quality of search in our research, although we believe our solutions are scalable to commercial volumes with a bit more effort.

Google maintains much more information about Web documents than typical search engines. Every hitlist includes position, font, and capitalization information. Additionally, we factor in hits from anchor text and the PageRank of the document. Combining all of this information into a rank is difficult. We designed our ranking function so that no one factor can have too much influence. For every matching document we compute counts of hits of different types at different proximity levels. These counts are then run through a series of lookup tables and eventually are transformed into a rank. This process involves many tunable parameters. We have not spent much time tuning the system; instead we have developed a feedback system which will help us tune these parameters in the future.

5. Results and performance

The most important measure of a search engine is the quality of its search results. While a complete user evaluation is beyond the scope of this paper, our own experience with Google has shown it to produce better results than the major commercial search engines for most searches. As an example which illustrates the use of PageRank, anchor text, and proximity, Fig. 2 shows Google's results for a search on "bill clinton". These results demonstrates some of Google's features. The results are clustered by server. This helps considerably when sifting through result sets. A number of results are from the whitehouse.gov domain which is what one may reasonably expect from such a search. Currently, most major commercial search engines do not return any results from whitehouse.gov, much less the right ones. Notice that there is no title for the first result. Instead, Google relied on anchor text to determine this was a good answer to the query. Similarly, the fifth result is an email address which, of course, is not crawlable. It is also a result of anchor text.

All of the results are reasonably high quality pages and, at last check, none were broken links.

Query: bill clinton

http://www.whitehouse.gov/
100.00% (no date) (0K)
http://www.whitehouse.gov/
Office of the President
99.67% (Dec 23 1996) (2K)

http://www.whitehouse.gov/WH/EOP/OP/html/OP_Home.html
Welcome To The White House
99.98% (Nov 09 1997) (5K)
http://www.whitehouse.gov/WH/Welcome.html
Send Electronic Mail to the President
99.86% (Jul 14 1997) (5K)

http://www.whitehouse.gov/WH/Mail/html/Mail_President.html

mailto:president@whitehouse.gov
99.98%
mailto:President@whitehouse.gov
99.27%
The "Unofficial" Bill Clinton
94.06% (Nov 11 1997) (14K)
http://zpub.com/un/un-bc.html
Bill Clinton Meets The Shrinks
86.27% (Jun 29 1997) (63K)
http://zpub.com/un/un-bc9.html
President Bill Clinton - The Dark Side
97.27% (Nov 10 1997) (15K)
http://www.realchange.org/clinton.htm
$3 Bill Clinton
94.73% (no date) (4K)
http://www.gatewy.net/~tjohnson/clinton1.html

Fig. 2. Sample results from Google.

This is largely because they all have high PageRank. The PageRanks are the percentages in red along with bar graphs. Finally, there are no results about a Bill other than Clinton or about a Clinton other than Bill. This is because we place heavy importance on the proximity of word occurrences. Of course a true test of the quality of a search engine would involve an extensive user study or results analysis which we do not have room for here. Instead, we invite the reader to try Google for themselves at http://google.stanford.edu.

Aside from search quality, Google is designed to scale cost effectively to the size of the Web as it grows. One aspect of this is to use storage efficiently. Table 1 has a breakdown of some statistics and storage requirements of Google.

It is important for a search engine to crawl and index efficiently. This way information can be kept up to date and major changes to the system can be tested relatively quickly. In total it took roughly 9 days to download the 26 million pages (including errors). However, once the system was running smoothly,

Table 1
Statistics

Storage statistics	
Total size of fetched pages	147.8 GB
Compressed repository	53.5 GB
Short inverted index	4.1 GB
Full inverted index	37.2 GB
Lexicon	293 MB
Temporary anchor data (not in total)	6.6 GB
Document index incl. variable width data	9.7 GB
Links database	3.9 GB
Total without repository	55.2 GB
Total with repository	108.7 GB

Web page statistics	
Number of Web pages fetched	24 million
Number of Urls seen	76.5 million
Number of E-mail addresses	1.7 million
Number of 404's	1.6 million

it ran much faster, downloading the last 11 million pages in just 63 hours, averaging just over 4 million pages per day or 48.5 pages per second. The indexer runs at roughly 54 pages per second. The sorters can be run completely in parallel; using four machines, the whole process of sorting takes about 24 hours.

Improving the performance of search was not the major focus of our research up to this point. The current version of Google answers most queries in between 1 and 10 seconds. This time is mostly dominated by disk IO over NFS (since our disks are spread over a number of machines). Furthermore, Google does not have many of the common optimizations used to speed up information retrieval systems, such as query caching, subindices on common terms, and other common optimizations. We intend to speed up Google considerably in the future. Table 2 has some sample query times from the current version of Google.

6. Conclusions

Google is designed to be a scalable search engine. The primary goal is to provide high quality search results over a rapidly growing World Wide Web. Google employs a number of techniques to improve search quality including page rank, anchor text, and proximity information. Furthermore, Google is a complete architecture for gathering Web pages, indexing them, and performing search queries over them.

6.1. Future work

A large-scale Web search engine is a complex system and much remains to be done. Our immediate goals are to improve search efficiency and to scale to approximately 100 million Web pages. Some simple improvements to efficiency include query caching, smart disk allocation, and subindices. Another area which requires much research is updates. We must have smart algorithms to decide what old Web pages should be recrawled and what new ones should be crawled. Work toward this goal has been done in [2]. One promising area of research is using proxy caches to build search databases, since they are demand driven. We are planning to add simple features supported by commercial search engines like boolean operators, negation, and stemming. However, other features are just starting to be explored such as relevance feedback and clustering (Google currently supports a simple hostname based clustering). We also plan to support user context (like the

Table 2
Search times

Query	Initial query		Same query repeated (IO mostly cached)	
	CPU time (s)	Total time (s)	CPU time (s)	Total time (s)
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

user's location), and result summarization. We are also working to extend the use of link structure and link text. Simple experiments indicate PageRank can be personalized by increasing the weight of a user's home page or bookmarks. As for link text, we are experimenting with using text surrounding links in addition to the link text itself. A Web search engine is a very rich environment for research ideas. We have far too many to list here so we do not expect this Future Work section to become much shorter in the near future.

6.2. High quality search

The biggest problem facing users of Web search engines today is the quality of the results they get back. While the results are often amusing and expand users' horizons, they are often frustrating and consume precious time. For example, the top result for a search for "Bill Clinton" on one of the most popular commercial search engines was the **Bill Clinton Joke of the Day: April 14, 1997**[7]. Google is designed to provide higher quality search so as the Web continues to grow rapidly, information can be found easily. In order to accomplish this Google makes heavy use of hypertextual information consisting of link structure and link (anchor) text. Google also uses proximity and font information. While evaluation of a search engine is difficult, we have subjectively found that Google returns higher quality search results than current commercial search engines. The analysis of link structure via PageRank allows Google to evaluate the quality of Web pages. The use of link text as a description of what the link points to helps the search engine return relevant (and to some degree high quality) results. Finally, the use of proximity information helps increase relevance a great deal for many queries.

6.3. Scalable architecture

Aside from the quality of search, Google is designed to scale. It must be efficient in both space and time, and constant factors are very important when dealing with the entire Web. In implementing Google, we have seen bottlenecks in CPU, memory access, memory capacity, disk seeks, disk throughput, disk capacity, and network IO. Google has evolved to overcome a number of these bottlenecks during various operations. Google's major data structures make efficient use of available storage space. Furthermore, the crawling, indexing, and sorting operations are efficient enough to be able to build an index of a substantial portion of the Web — 24 million pages, in less than one week. We expect to be able to build an index of 100 million pages in less than a month.

6.4. A research tool

In addition to being a high quality search engine, Google is a research tool. The data Google has collected has already resulted in many other papers submitted to conferences and many more on the way. Recent research such as [1] has shown a number of limitations to queries about the Web that may be answered without having the Web available locally. This means that Google (or a similar system) is not only a valuable research tool but a necessary one for a wide range of applications. We hope Google will be a resource for searchers and researchers all around the world and will spark the next generation of search engine technology.

Acknowledgments

Scott Hassan and Alan Steremberg have been critical to the development of Google. Their talented contributions are irreplaceable, and the authors owe them much gratitude. We would also like to thank Hector Garcia-Molina, Rajeev Motwani, Jeff Ullman, and Terry Winograd and the whole WebBase group for their support and insightful discussions. Finally we would like to recognize the generous support of our equipment donors IBM, Intel, and Sun and our funders. The research described here was conducted as part of the Stanford Integrated Digital Library Project, supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA and NASA, and by Interval Research, and the industrial partners of the Stanford Digital Libraries Project.

[7] http://www.io.com/~cjburke/clinton/970414.html

References

[1] S. Abiteboul and V. Vianu, Queries and computation on the Web, in: *Proceedings of the International Conference on Database Theory*, Delphi, Greece, 1997.

[2] J. Cho, H. Garcia-Molina and L. Page, Efficient crawling through URL ordering, in: *Proc. of the 7th International World Wide Web Conference (WWW 98)*, Brisbane, Australia, April 14–18, 1998; also *Comput. Networks ISDN Systems*, 30(1–7): 161–172 (this volume).

[3] J. Kleinberg, Authoritative sources in a hyperlinked environment, in: *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[4] M. Marchiori, The quest for correct information on the Web: hyper search engines, in: *Proc. of the 6th International WWW Conference (WWW 97)*, Santa Clara, USA, April 7–11, 1997.

[5] Mauldin, M.L., Lycos design choices in an Internet search service, IEEE Expert Interview, http://www.computer.org/pubs/expert/1997/trends/x1008/mauldin.htm

[6] O.A. McBryan, GENVL and WWWW: tools for taming the Web, in: *Proc. of the 1st International Conference on the World Wide Web*, CERN, Geneva, Switzerland, May 25–27, 1994, http://www.cs.colorado.edu/home/mcbryan/mypapers/www94.ps

[7] L. Page, S. Brin, R. Motwani and T. Winograd, The PageRank citation ranking: bringing order to the Web, Manuscript in Progress, http://google.stanford.edu/~backrub/pageranksub.ps

[8] B. Pinkerton, Finding what people want: experiences with the WebCrawler, in: *Proc. of the 2nd International WWW Conference*, Chicago, USA, October 17–20, 1994, http://info.webcrawler.com/bp/WWW94.html

[9] E. Spertus, ParaSite: mining structural information on the Web, in: *Proc. of the 6th International WWW Conference (WWW 97)*, Santa Clara, USA, April 7–11, 1997.

[10] D.K. Harman and E.M. Voorhees (Eds.), *Proceedings of the Fifth Text REtrieval Conference (TREC-5)*, Gaithersburg, Maryland, November 20–22, 1996, Department of Commerce, National Institute of Standards and Technology, 1996; full text at http://trec.nist.gov/

[11] I.H Witten, A. Moffat, and T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Van Nostrand Reinhold, New York, NY, 1994.

[12] R. Weiss, B. Velez, M.A. Sheldon, C. Manprempre, P. Szilagyi, A. Duda, and D. K. Gifford, HyPursuit: a hierarchical network search engine that exploits content-link hypertext clustering, in: *Proc. of the 7th ACM Conference on Hypertext*, New York, 1996.

Sergey Brin received his B.S. degree in mathematics and computer science from the University of Maryland at College Park in 1993. Currently, he is a Ph.D. candidate in computer science at Stanford University where he received his M.S. in 1995. He is a recipient of a National Science Foundation Graduate Fellowship. His research interests include search engines, information extraction from unstructured sources, and data mining of large text collections and scientific data.

Lawrence Page was born in East Lansing, Michigan, and received a B.S.E. in Computer Engineering at the University of Michigan Ann Arbor in 1995. He is currently a Ph.D. candidate in Computer Science at Stanford University. Some of his research interests include the link structure of the Web, human computer interaction, search engines, scalability of information access interfaces, and personal data mining.