Web Search Engine

Deepanshu Bansal & Monica Godara
Student, Department of Electronics & Computer Science,
Dronacharya College of Engineering, Gurgaon, India
Email: deepanshubansal53@yahoo.com

Abstract

In this paper, we present Web search engine, a prototype of a Web search engine that makes heavy use of the structure present in hypertext. Google is designed to crawl, 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 that can exploit the additional information present in hypertext. In addition, 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! 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 automated search engines. We have built a large-scale search engine that 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 10^{100} and fits well with our goal of building very large-scale search engines[12].

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) 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. 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[1]. With the increasing number of users on the web, and automated systems that 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 that 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[6].

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. Further, we expect that the cost to index and store text or HTML will eventually decline relative to the amount that will be available. This will result in favourable scaling properties for centralized systems like Google[6].

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,"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 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. In particular, link structure 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.

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 cause's search engine technology to remain largely a black art and to be advertising oriented. 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[8].

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. Second, Google utilizes link to improve search results[4].

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[4]. 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 (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))

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. Many other details are beyond the scope of this paper[4].

2.1.2 Intuitive Justification

PageRank can be thought of as a model of user behaviour. 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. In addition, 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.

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. In addition, pages that have perhaps only one citation from something like the Yahoo! 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[3].

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 that 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 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 that must be processed. In our current crawl of 24 million pages, we had over 259 million anchors that we indexed[4].

2.3 Other Features

Aside from PageRank and the use of anchor text, Google has several other features. First, it has location information for all hits and so it makes extensive use of proximity in search. Second,

Google keeps track of some visual presentation details such as font size of words. Words in a larger or bolder font are weighted higher than other words. Third, full raw HTML of pages is available in a repository.

3 System Anatomy

First, we will provide a high level discussion of the architecture. Then, there is some in-depth descriptions of important data structures. Finally, the major applications: crawling, indexing, and searching will be examined in

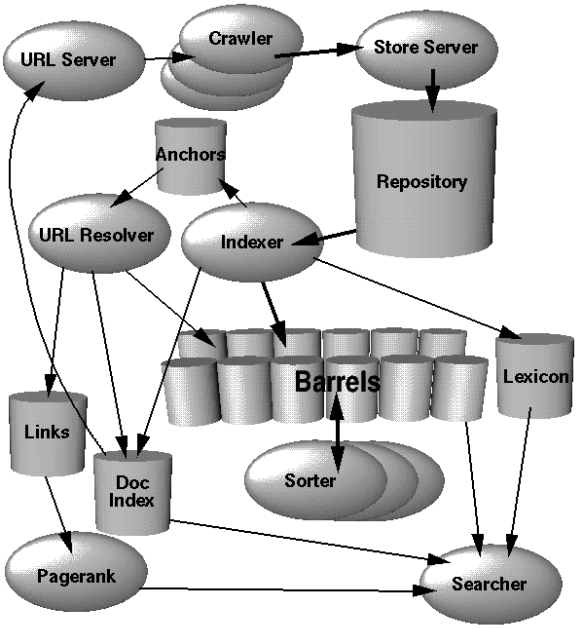

Figure 1. High Level Google Architecture depth.

3.1 Google Architecture Overview

In this section, we will give a high level overview of how the whole system works as pictured in Figure 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, several distributed crawlers do the web crawling (downloading of web pages). There is an URLserver that sends lists of URLs to be fetched to the crawlers. The web pages that are fetched are then sent to the store server. The store server 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[4].

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 that 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 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 Dump Lexicon 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 Dump Lexicon together with the inverted index and the PageRanks to answer queries.

3.2 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[5].

3.3 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[5].

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. This amounts to roughly 600K per second of data. A major performance stress is DNS lookup. Each crawler maintains a its own DNS cache so it does not need to do a DNS lookup before crawling each document. 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.

It turns out that running a crawler which connects to more than half a million servers, and generates tens of millions of log entries generates a fair amount of email and phone calls. Because of the vast number of people coming on line, there are always those who do not know what a crawler is, because this is the first one they have seen. Almost daily, we receive an email something like, "Wow, you looked at a lot of pages from my web site. How did you like it?"[10] There are also some people who do not know about the robots exclusion protocol, and think their page should be protected from indexing by a statement like, "This page is copyrighted and should not be indexed", which needless to say is difficult for web crawlers to understand. Also, because of the huge amount of data involved, unexpected things will happen. For example, our system tried to crawl an online game. This resulted in lots of garbage messages in the middle of their game! It turns out this was an easy problem to fix. But this problem had not come up until we had downloaded tens of millions of pages. Because of the immense variation in web pages and servers, it is virtually impossible to test a crawler without running it on large part of the Internet. Invariably, [7]there are hundreds of obscure problems which may only occur on one page out of the whole web and cause the crawler to crash, or worse, cause unpredictable or incorrect behavior. Systems which access large parts of the Internet need to be designed to be very robust and carefully tested. 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[7].

3.4 Indexing the Web

Parsing -- Any parser which is designed to run on the entire Web must handle a huge array of possible errors. These range from typos in HTML tags to kilobytes of zeros in the middle of a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of

other errors that challenge anyone's imagination to come up with equally creative ones. For maximum speed, instead of using YACC to generate a CFG parser, we use flex to generate a lexical analyzer which we outfit with its own stack. Developing this parser which runs at a reasonable speed and is very robust involved a fair amount of work.

- **Indexing Documents into Barrels** -- After each document is parsed, it is encoded into a number of barrels. Every word is converted into a wordID by using an in-memory hash table -- the lexicon. New additions to the lexicon hash table are logged to a file. Once the words are converted into wordID's, their occurrences in the current document are translated into hit lists and are written into the forward barrels. The main difficulty with parallelization of the indexing phase is that the lexicon needs to be shared. Instead of sharing the lexicon, we took the approach of writing a log of all the extra words that were not in a base lexicon, which we fixed at 14 million words. That way multiple indexers can run in parallel and then the small log file of extra words can be processed by one final indexer[9].
- **Sorting** -- In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by wordID to produce an inverted barrel for title and anchor hits and a full text inverted barrel. This process happens one barrel at a time, thus requiring little temporary storage. Also, we parallelize the sorting phase to use as many machines as we have simply by running multiple sorters, which can process different buckets at the same time. Since the barrels don't fit into main memory, the sorter further subdivides them into baskets which do fit into memory based on wordID and docID. Then the sorter, loads each basket into memory, sorts it and writes its contents into the short inverted barrel and the full inverted barrel[9].

3.5 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. The google query evaluation processes to put a limit on response time, once a certain number (currently 40,000) of matching documents are found, the searcher automatically goes to step 8 . This means that it is possible that sub-optimal results would be returned. We are currently investigating other ways to solve this problem. In the past, we sorted the hits according to PageRank, which seemed to improve the situation.

4.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, Figure 4 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. This is

because it was not crawled. 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. 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[2].

4.1 Storage Requirements

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. Due to compression the total size of the repository is about 53 GB, just over one third of the total data it stores. At current disk prices this makes the
repository a relatively cheap source of useful data. More importantly, the total of all the data used by the search engine requires a comparable amount of storage, about 55 GB. Furthermore, most queries can be answered using just the short inverted index. With better encoding and compression of the Document Index, a high quality web search engine may fit onto a 7GB drive of a new PC.

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**

Table 1. Statistics

Web Page Statistics	
Number of Web Pages Fetched	24 million
Number of Urls Seen	76.5 million
Number of Email Addresses	1.7 million
Number of 404's	1.6 million

4.2 System Performance

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. For Google, the major operations are Crawling, Indexing, and Sorting. It is difficult to measure how long crawling took overall because disks filled up, name servers crashed, or any number of other problems which stopped the system. In total it took roughly 9 days to download the 26 million pages (including errors). However, once the system was running smoothly, 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. We ran

the indexer and the crawler simultaneously. The indexer ran just faster than the crawlers. This is largely because we spent just enough time optimizing the indexer so that it would not be a bottleneck. These optimizations included bulk updates to the document index and placement of critical data structures on the local disk. 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[11].

	Initial Query		**Same Query Repeated (IO mostly cached)**	
Query	**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

4.3 Search Performance

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 disks are spread over a number of machines). Furthermore, Google does not have any optimizations such as query caching, subindices on common terms, and other common optimizations. We intend to speed up Google considerably through distribution and hardware, software, and algorithmic improvements. Our target is to be able to handle several hundred queries per second. Table 2 has some sample query times from the current version of Google. They are repeated to show the speedups resulting from cached IO[11].

5 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.

5.1 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. 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 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.

5.2 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. 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.

5.3 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.

5.4 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 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.

6. References

[1] [1]Best of the Web 1994 -- Navigators http://botw.org/1994/awards/navigators.html

[2] [2]Bill Clinton Joke of the Day: April 14, 1997. http://www.io.com/~cjburke/clinton/970414.html.

[3] [3]Bzip2 Homepage http://www.muraroa.demon.co.uk/

[4] [4]Google Search Engine http://google.stanford.edu/

[5] [5]Harvest http://harvest.[2].com/

[6] [6]Mauldin, Michael L. Lycos Design Choices in an Internet Search Service, IEEE Expert Interview http://www.computer.org/pubs/expert/1997/trends/x1008/mauldin.htm

[7] [7]The Effect of Cellular Phone Use Upon Driver Attention http://www.webfirst.com/aaa/text/cell/cell0toc.htm

[8] [8]Search Engine Watch http://www.searchenginewatch.com/

[9] [9]RFC 1950 (zlib) ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html

[10] [10]Robots Exclusion Protocol: http://info.webcrawler.com/mak/projects/robots/exclusion.htm

[11] [11]Web Growth Summary: http://www.mit.edu/people/mkgray/net/web-growth-summary.html

[12] [12]Yahoo! http://www.yahoo.com/