Notice of Violation of IEEE Publication Principles

"The Anatomy of a Large-Scale Hyper Textual Web Search Engine"
by Umesh Sehgal, Kuljeet Kaur, Pawan Kumar
in the Proceedings of the Second International Conference on Computer and Electrical Engineering, 2009. ICCEE '09, December 2009, pp. 491-495

After careful and considered review of the content and authorship of this paper by a duly constituted expert committee, this paper has been found to be in violation of IEEE's Publication Principles.

This paper contains significant portions of original text from the paper cited below. The original text was copied with insufficient attribution (including appropriate references to the original author(s) and/or paper title) and without permission.

Due to the nature of this violation, reasonable effort should be made to remove all past references to this paper, and future references should be made to the following article:

"The Anatomy of a Large-Scale Hypertextual Web Search Engine"
by Sergey Brin and Lawrence Page
Computer Networks and ISDN Systems, Volume 30, Issue 1-7, Elsevier, April 1998, pp. 107-117

The Anatomy of a Large-Scale Hyper Textual Web Search Engine

Dr.Umesh Sehgal, Ms.Kuljeet Kaur, Mr.Pawan Kumar

Department of Computer Applications, Lovely Professional University (Phagwara –[PB]. India)

[kuljeet_brar2000@yahoo.co.in]

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. 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 to define the values and traditional techniques of data in hypertext. 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, Page Rank, Google**

I. INTRODUCTION

The web creates new challenges for information retrieval. The amount of information and users on the web is growing rapidly. People are likely to surf the web using its link graph, often starting with high quality human maintained indices such as Rediff! 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. 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. 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 (from Search Engine Watch). It is foreseeable that by the year 2000, a comprehensive index of the Web had contained over a billion documents.

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.

II. 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 Page Rank. Second, Google utilizes link to improve search results.

1. PAGE RANK: DESCRIPTION OF CALCULATION

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.

IEEE computer society

Also C(A) is defined as the number of links going out of page A. The Page Rank of a page A is given as follows:

$$PR(A) = (1-d) + d (PR(T1)/C(T1) + ... + PR(Tn)/C(Tn))$$

2. OTHER FEATURES

- It has location information for all hits and so it makes extensive use of proximity in search.
- 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.
- HTML of pages is available in a repository.

III DIFFERENCES BETWEEN THE WEB AND WELL CONTROLLED COLLECTIONS

The web is a vast collection of completely uncontrolled heterogeneous documents. Documents on the web have extreme variation internal to the documents, and also in the external meta information that might be available. For example, documents differ internally in their language (both human and programming), vocabulary (email addresses, links, zip codes, phone numbers, product numbers), type or format (text, HTML, PDF, images, sounds), and may even be machine generated (log files or output from a database). Examples of external meta information include things like reputation of the source, update frequency, quality, popularity or usage, and citations. Not only are the possible sources of external meta information varied, but the things that are being measured vary many orders of magnitude as well. For example, compare the usage information from a major homepage, like Yahoo's which currently receives millions of page views every day with an obscure historical article which might receive one view every ten years. Clearly, these two items must be treated very differently by a search engine.

Another big difference between the web and traditional well controlled collections is that there is virtually no control over what people can put on the web. Couple this flexibility to publish anything with the enormous influence of search engines to route traffic and companies which deliberately manipulating search engines for profit become a serious problem. This problem that has not been addressed in traditional closed information retrieval systems. Also, it is interesting to note that metadata efforts have largely failed with web search engines, because any text on the

page which is not directly represented to the user is abused to manipulate search engines. There are even numerous companies which specialize in manipulating search engines for profit.

IV SYSTEM ANATOMIES

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 depth.

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.
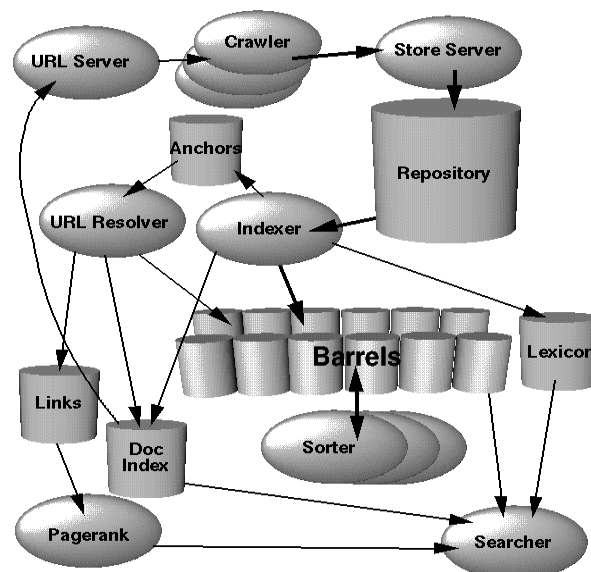

Fig: 1: High Level Google Architecture

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 URL server 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, uncompressed 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 Page Ranks for all the documents.

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.

a) BIG FILES

Big Files are virtual files spanning multiple file systems and are addressable by 64 bit integers. The allocation among multiple file systems is handled automatically.

Repository: 53.5 GB = 147.8 GB uncompressed

sync	length	compressed packet
sync	length	compressed packet
...		

Packet (stored compressed in repository)

docid	ecode	urllen	pagelen	url	page

Fig: 2: Repository Data Structure

The Big Files package also handles allocation and de allocation of file descriptors, since the operating systems do not provide enough for our needs. Big Files also support rudimentary compression options.

b) REPOSITORY

The repository contains the full HTML of every web page. Each page is compressed using zlib. The choice of compression technique is a tradeoff between speed and compression ratio. We chose zlib's speed over a significant improvement in compression offered by bzip. The compression rate of bzip was approximately 4 to 1 on the repository as compared to zlib's 3 to 1 compression. In the repository, the documents are stored one after the other and are prefixed by docID, length, and URL as can be seen in Figure 2. The repository requires no other data structures to be used in order to access it. This helps with data consistency and makes development much easier; we can rebuild all the other data structures from only the repository and a file which lists crawler errors.

c) FORWARD INDEX
d) INVERTED INDEX
e) HIT LISTS

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.

Hit: 2 bytes

plain:	cap:1	imp:3	position: 12		
fancy:	cap:1	imp = 7	type: 4	position: 8	
anchor:	cap:1	imp = 7	type: 4	hash:4	pos: 4

Forward Barrels: total 43 GB

docid	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	null wordid		
docid	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	wordid: 24	nhits: 8	hit hit hit hit
	null wordid		
...			

Lexicon: 293MB   **Inverted Barrels: 41 GB**

wordid	ndocs	→	docid: 27	nhits:5	hit hit hit hit
wordid	ndocs		docid: 27	nhits:5	hit hit hit
wordid	ndocs		docid: 27	nhits:5	hit hit hit hit
			docid: 27	nhits:5	hit hit
			...		

Fig: 3: Forward and Reverse Indexes and the Lexicon

This limits it to 8 and 5 bits respectively (there are some tricks which allow 8 bits to be borrowed from the wordID). If the length is longer than would fit in that many bits, an escape code is used in those bits, and the next two bytes contain the actual length.

3. 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 word ID 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 word ID'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.

- Sorting -- In order to generate the inverted index, the sorter takes each of the forward barrels and sorts it by word ID 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 word ID and doc ID. Then the sorter loads each basket into memory, sorts it and writes its contents into the short inverted barrel and the full inverted barrel.

V RESULTS AND PERFORMANCE

Query: Lovely   University
http://www.lpu.co.in/
100.00% ▬▬▬   (no   date)   (0K)
http://www.lpu.co.in
          Office   of   the   President
99.67% ▬▬▬ (Dec 23 1996) (2K)
                    http://www.umslpu.co.in
     Welcome To The Lovely University
99.98% ▬▬▬ (Nov 09 2007) (5K)
          http://www.lpu/Welcome.html
Send Electronic Mail to the President
99.86% ▬▬▬ (Jul 14 2008) (5K)
                    http://www.lpuums.co.in
http://192.168.5.98/lpuums
     Mr.Mittal   Meets   Mr.Rohit   Dhand
86.27% ▬▬▬ (Jun 29 2009) (63K)
                    http://192.168.25.100
President   Mr.Mittal   -   The   Dark   Side
97.27% ▬▬▬ (Aug 10 2009) (15K)
http://www.lpuums.co.in
$3   Mr.Mittal
94.73% ▬▬▬ (no date) (4K)

Fig: 4: Sample Results from Google

VI CONCLUSION

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. But in these days Google search engine is mainly used for terrorism, because the destructive mind people will always use this for planning disasters. Military uses this for the condition of safety, but many times these are used crime and terrorism. Solution is that to do the

search location for same pattern but some identifiers and access points, web portal will be installed for, to do and search these sites.

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. We must have smart algorithms to decide what old web pages should be recrawled and what new ones should be crawled. 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 of military areas, but not in the crime areas.

VII REFERENCES

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

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

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

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