f(x) = x Papers We Love

“The Anatomy of a Large-Scale Hypertextual Web Search Engine”

by Sergey Brin and Lawrence Page

Papers We Love Bucharest

Eduard Mucilianu *Stefan Alexandru Adam*

31st of August 2015

TechHub

Short History of Web Search

Altavista and Yahoo Directory in 1994

Google and MSN launch in 1998

- Goto.com annual revenues were close to $ 1 billion
  became Overture and was acquired by Yahoo in 2003
  First Pay-per-click business model for advertising
  Used first-price auctions for keywords, **casino** was expensive

- Google AdWords uses generalized second-price auctions, not necessarily promotes truthfulness

Prior Related Work

Primary benchmark for **Information Retrieval**, the Text Retrieval Conference, TREC, used a fairly small, **well controlled collection** for their benchmarks

“Some argue that on the web, users should specify more accurately what they want and add more words to their query”

Differences Between the Web and Well Controlled Collections

External meta information includes things like reputation of the source, update frequency, quality, popularity or usage, and citations

PageRank Scoring

Propagating weights through the link structure of the web
Aided by Anchor Text to improve search results, especially with non-text information

The PageRank of a page is the long-term visit rate, it has its origins in Citation Analysis

In a steady state, what is the probability to be on that page ?

Need a process that allows us to get into that steady state

Is query independent ! Just a way to measure importance of a page

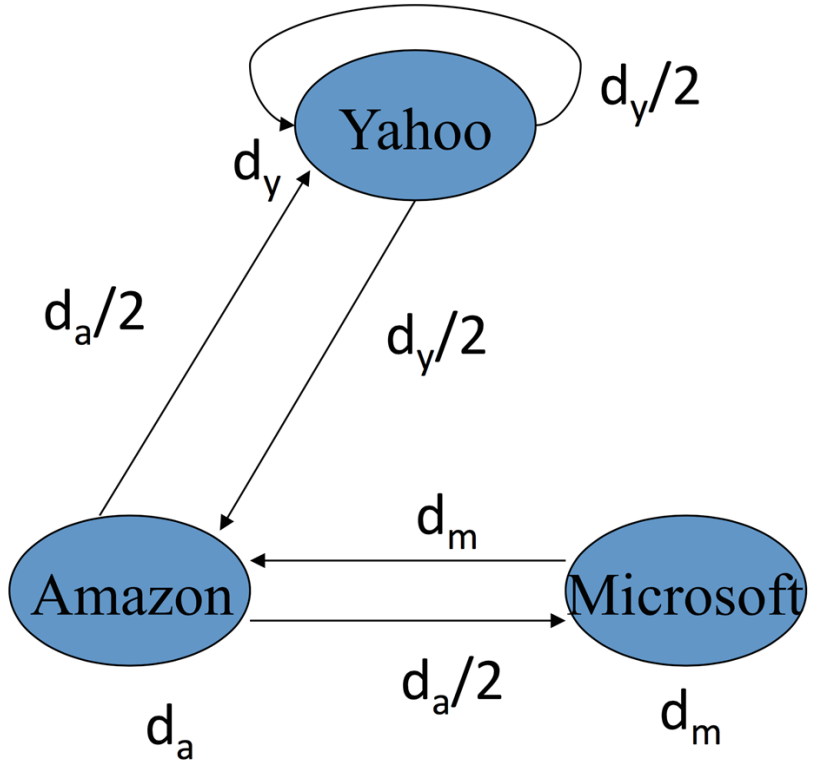
Markov Chains

	dY	dA	dM
dY	1/2	1/2	0
dA	1/2	0	1/2
dM	0	1	0

Set of **n** states and **n** X **n** transition probability matrix for outlinks
A good way for abstracting random walks, the way a web surfer will follow links whilst browsing

For PageRank, a state is actually a web page

Intermediate process for reaching the steady-state probability distribution



Markov Property

Memorylessness of a stochastic process

The probability of next state is ONLY dependent on the current state

Even if you came from Google.com, the probability that you will jump to Yahoo.com randomnly is the same

Dead-ends and Teleporting

Surfer can get stuck whilst browsing in

- a dead-end

or

- spider trap (group of pages that together form a dead-end)

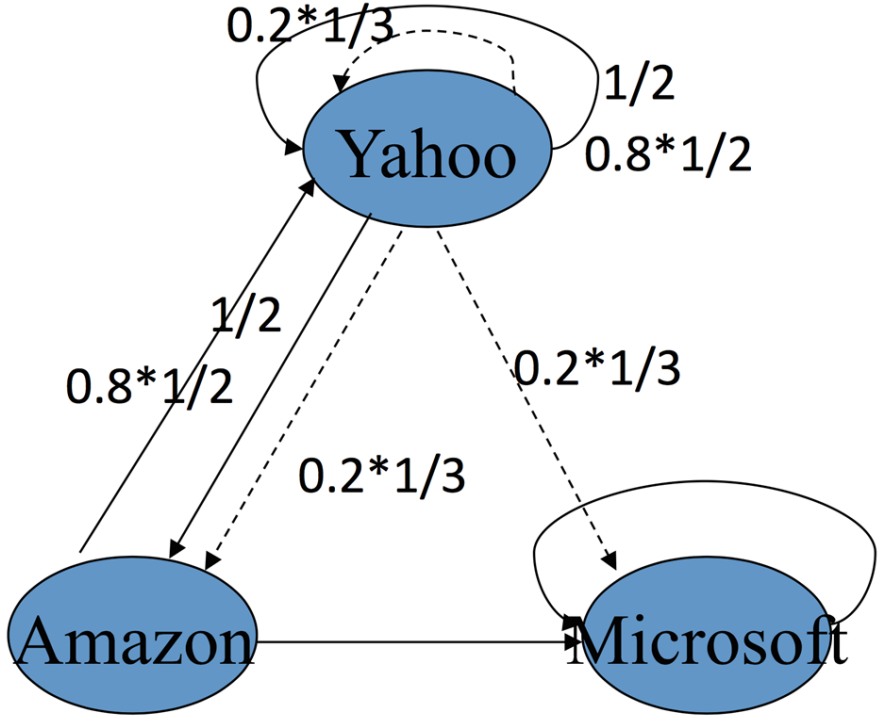
User may choose to randomly switch to an out of context page

At a non-dead-end page jump to a random page with a chosen parameter of around 10%-20%
Use remaining 80%-90% for out-links

Power Method

$x_{k+1} = P^T x_k = (P^T)^k x_1$

x is the probability vector, we want **a**=P^T**a**
Stable probability vector **a** is the PageRank vector



P^T = 0.8

1/2	1/2	0
1/2	0	0
0	1/2	1

+ 0.2

1/3	1/3	1/3
1/3	1/3	1/3
1/3	1/3	1/3

=

7/15	7/15	1/15
7/15	1/15	1/15
1/15	7/15	13/15

Stable state

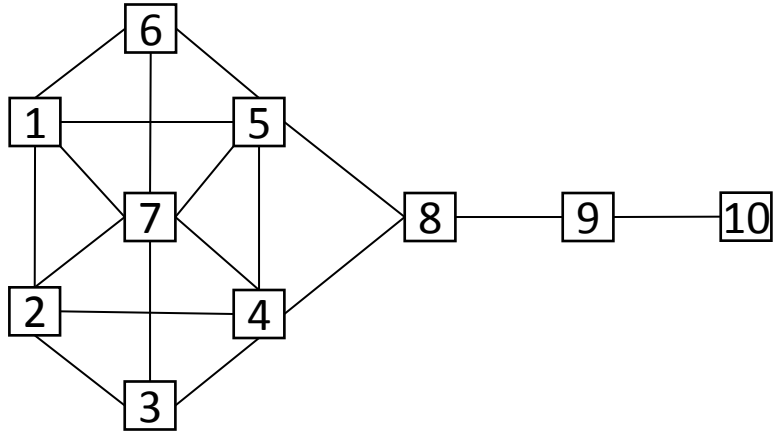
Start with uniformly distributed probability vector, thus giving each page equal chances
Settle on **a** when $| x_{k+1} - x_k |$ is below desired threshold

x_1	$x_2 = P^T x_1$	$x_3 = P^T x_2$	$x_4 = P^T x_3$
1/3	1/3	7/25	97/375
1/3	1/5	1/5	67/375
1/3	7/15	13/25	211/375

. . .

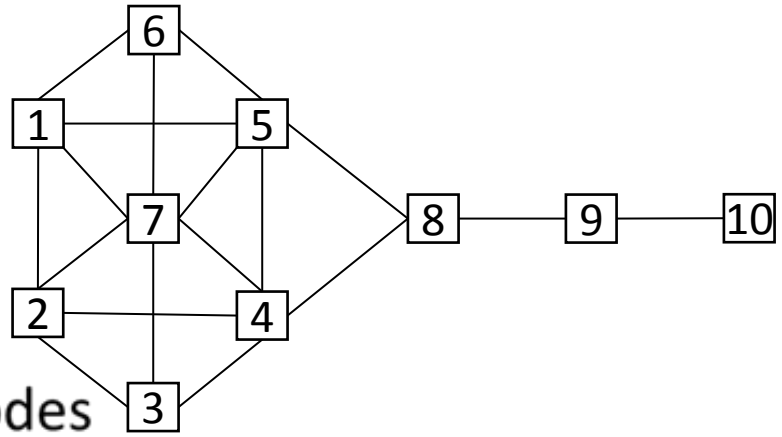
PR(Yahoo)	7/33
PR(Amazon)	5/33
PR(Microsoft)	7/11

Centrality measures



- Degrees Centrality
  - Measures immediate risk of catching something
- Closeness Centrality
  - Measures how close a node is in relation with the other
- Betweenness Centrality
  - Measures the number of times a node acts like a bridge
- Eigenvector Centrality
  - Measures the influence, prestige of a node. Depends on neighbors centrality
  - PageRank is a deviation

Other Centrality measures



- Degrees Centrality
  - $\boldsymbol{C_D(x) = deg(x)}$, deg(x) the number of adjacent nodes
- Closeness Centrality
  - $\boldsymbol{C_C(x) = \frac{1}{\sum_y d(y,x)}}$, $d(y,x)$ *the distance between* x *and* y
- Betweenness Centrality
  - $\boldsymbol{C_B(x) = \sum_{s \neq x \neq t} \frac{\sigma_{st}(x)}{\sigma_{st}}}$ where σ_{st} represents the total number of shortest paths and $\sigma_{st}(x)$ represents the number of shortest paths that pass throw x

Eigenvector Centrality

- Solve eq. $\boldsymbol{A} \cdot \boldsymbol{v} = \boldsymbol{\lambda} \cdot \boldsymbol{v}$, where $\boldsymbol{A}$ is the adjacency matrix. The solution represents the eigenvector for the biggest eigenvalue or eigenvalue 1 (in the stochastic variant)
- Consider the stochastic version of $\boldsymbol{A}$, and consider an equal distribution vector v. The probability is $\mathbf{v}^{*}=\boldsymbol{A}^{\boldsymbol{n}}\cdot\mathbf{v}$
- Google PageRank algorithm (overcome disconnected networks)

Design requirements

- Ability to scale
- Improved search quality
- Strong performance
- Efficiency

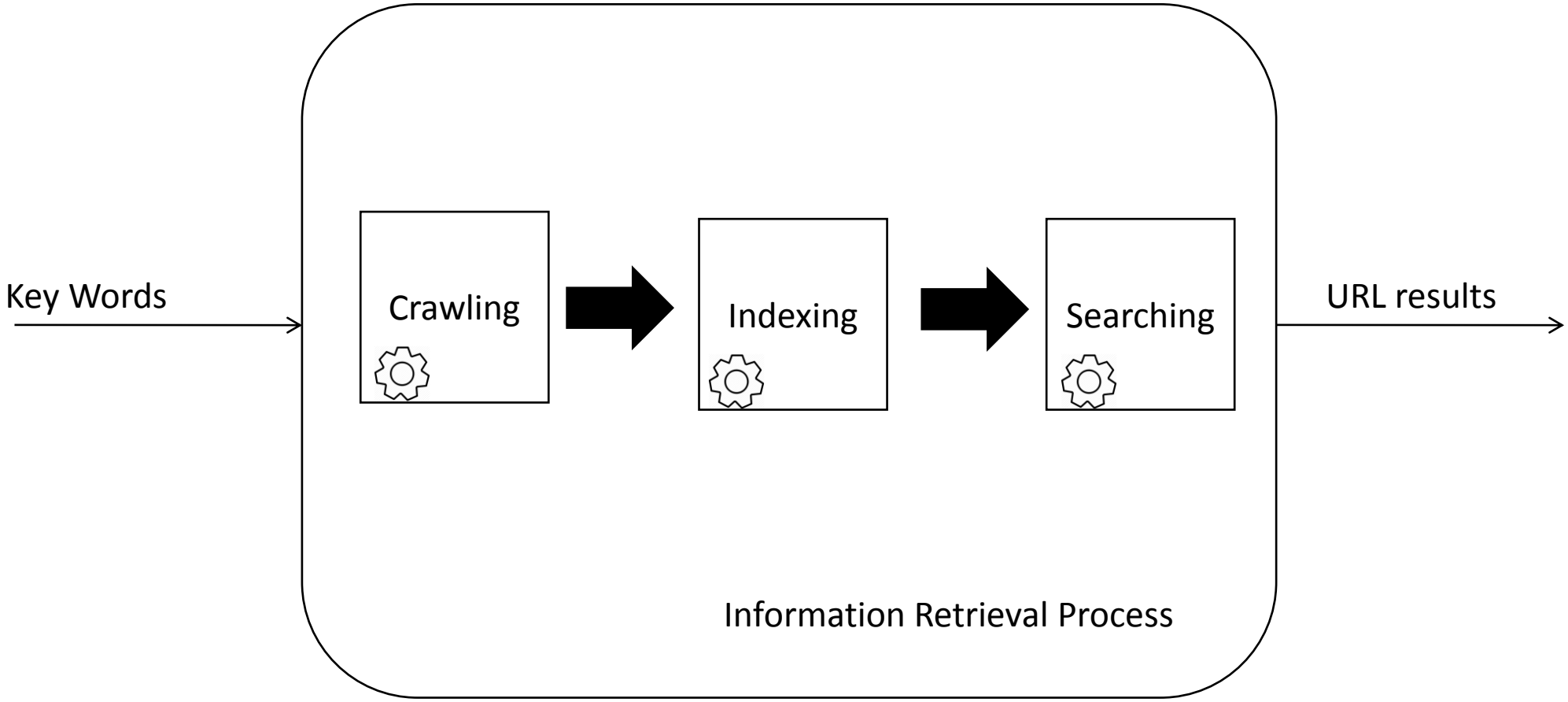
Design strategies

- Modular based architecture
- Define a ranking system
- Most of google engine is written in C and C++ and runs on Linux/Solaris
- Avoid disk seeks whenever possible

Search Text Engine - Web Object Model

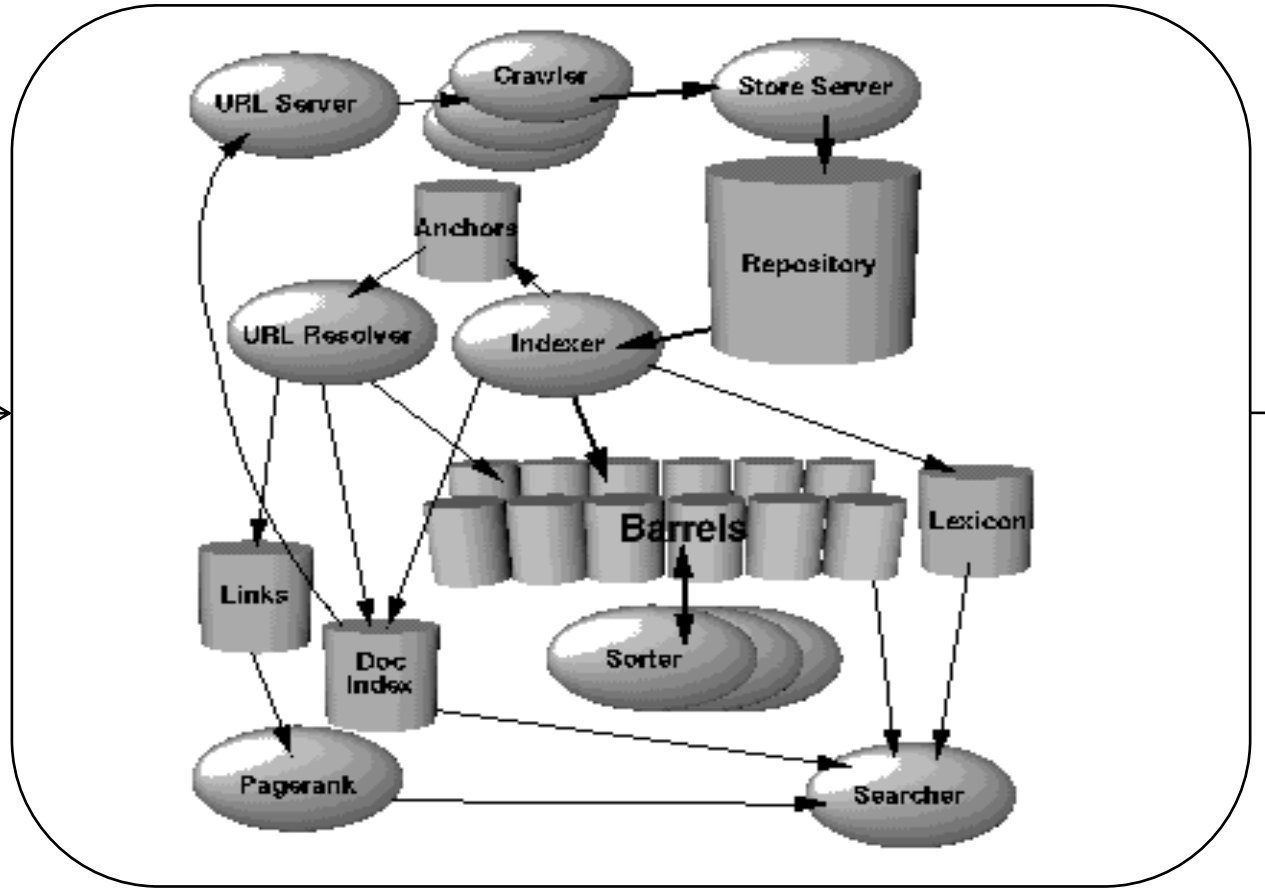
- URL - identify a web resource
- Document - the html content of a web resource
- DocID - uniquely identifies a document
- Word
  - unit of meaning part of a Lexicon
  - Appears in document having different attributes (position, font size, link, color)
- WordID - uniquely identifies a word
- Lexicon the totality of lexemes

Search Text Engine - Base Model



Search Text Engine - Model

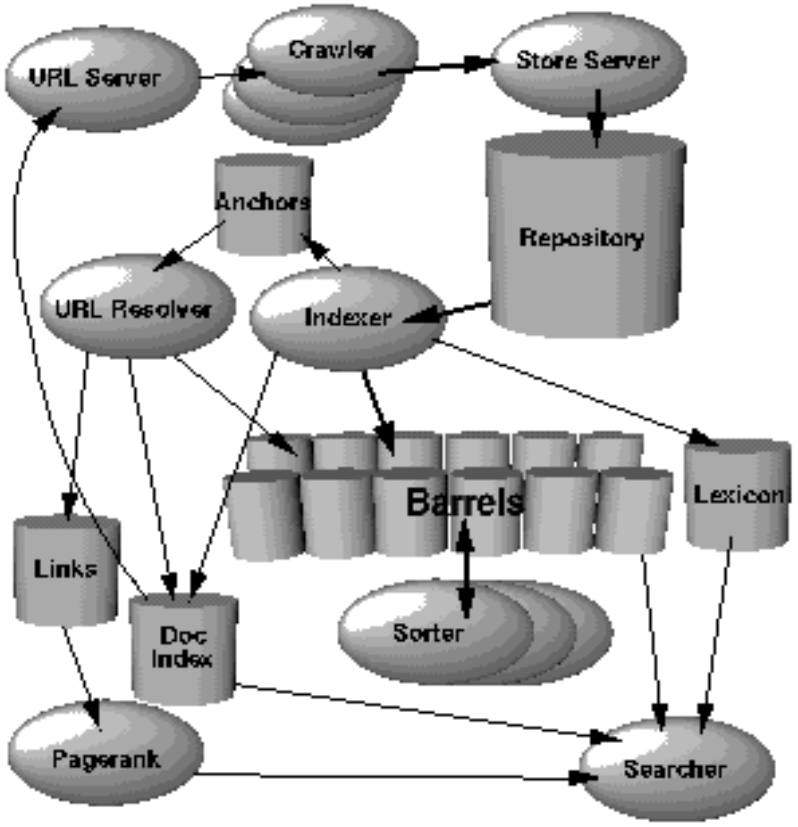
Key Words



URL results

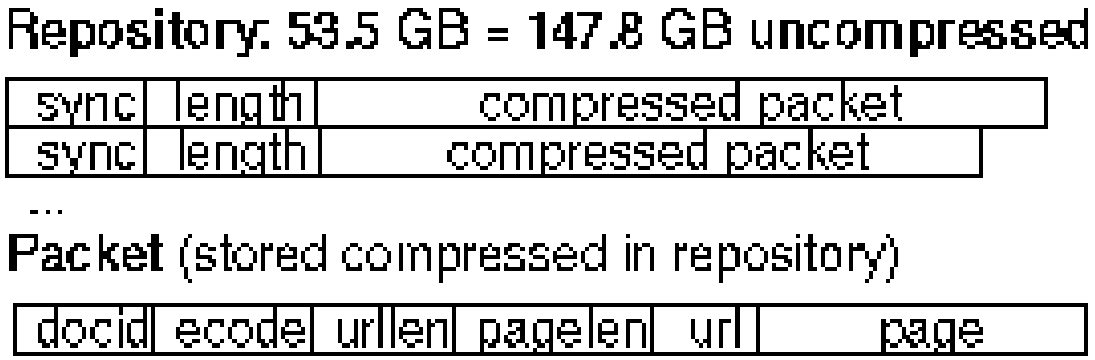
Modules Responsibilities

- **URLServer** sends url to **Crawlers**
- **Crawler** fetches data from a URL
- **Store Server** compresses data and saves it to **Repository** storage
- **Indexer**
  - converts the documents from Repository into words and **Hits**. Hits then are stored in **barrels** and new words are stored in **Lexicon**
  - Extracts links and saves them into **Anchors** storage
  - Indexes documents and save the indexes into the **Document Index** storage
- **URL Resolver** converts relative links to absolute links and saves them to **Links** storage
- **PageRank** computes the page rank
- **Sorter** sorts hits
- **Searcher** computes the search



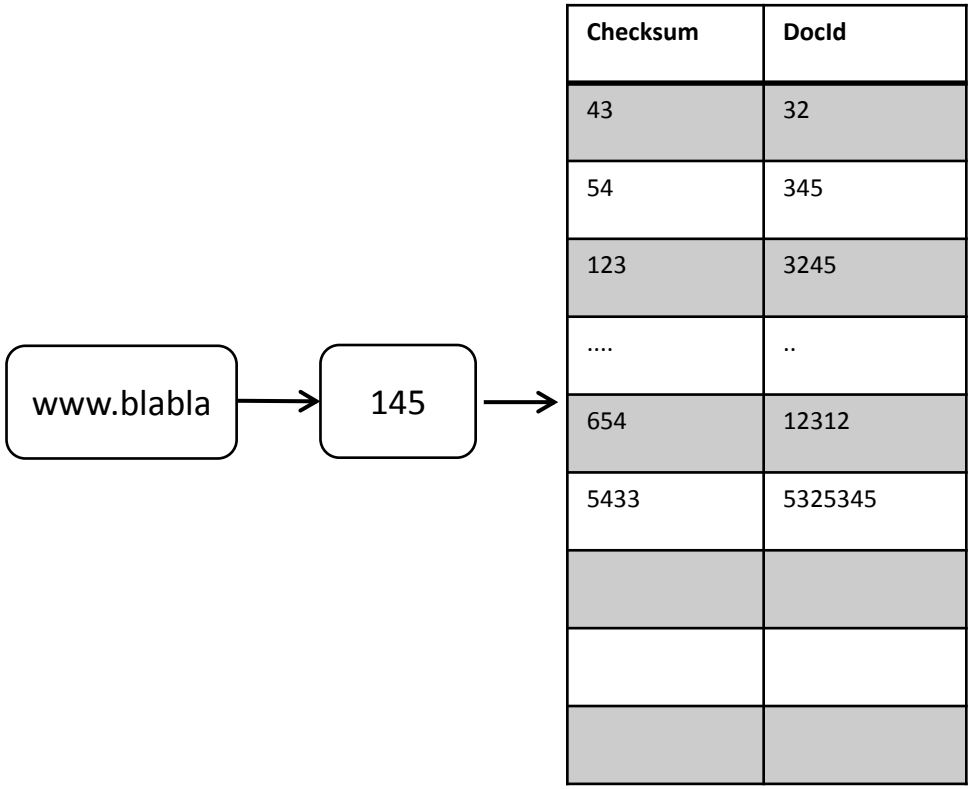
Repository

- Contains the full HTML of every page
- Records are stored one after another
- Documents are compressed using zlib
- Every record contains
  - DocID - document identifier
  - Length
  - URL
  - Document (html content)
- approx. 53 GB (stored on Large File)



Document Index

- It is a ISAM index ordered by DocID
- Any record contains
  - pointer to the document record in Repository
  - Document checksum
- Contains a file used to convert URLs into docIDs
  - File is ordered by checksum (an int)
  - Binary search is applied for getting the DocID
  - Supports batch updates
  - URL resolver uses it to convert URL to DocId
- 9.7 GB

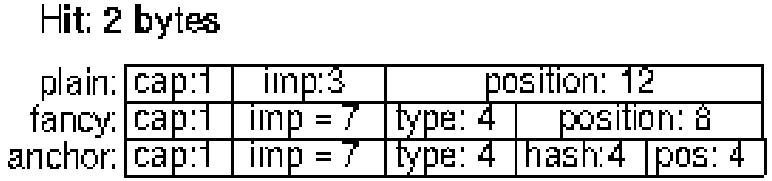


Lexicon

- The inventory of lexemes.
  - Lexemes examples : “Eat”, “Ate”, “Eaten” represents a single lexeme; “Flesh and blood” = expression = a single lexeme; “Flesh” = a single lexeme
- Contains the full list of words (word ~ lexeme)
- Fit in memory (293 MB)
- 14 million records
- Records are separated by null
- Each record contains
  - WordID
  - A pointer to the wordID in the Inverted Index

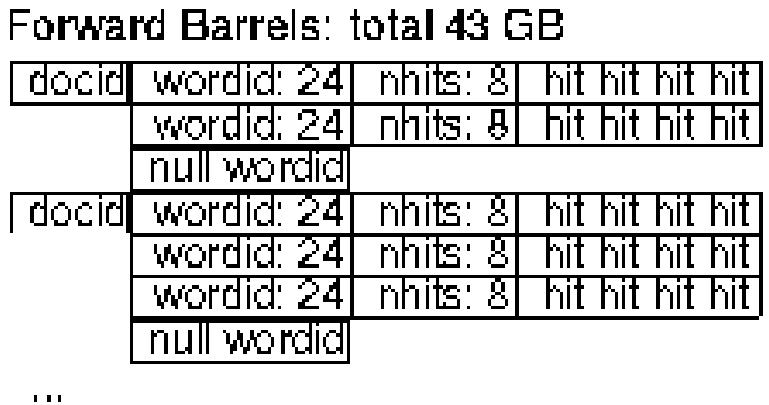
! Stemming was not supported

Hit List



- A list of occurrences of a particular word
- Represents the most precious information and is a result of a long and expensive chain of processing
- Are stored into the Forward Index and Inverted Index
- Each hit is characterized by:
  - Fancy/Plain
  - Font Size (relative to the document)
  - Capitalization
  - Position
- Each hit is stored in two bytes in an encoded fashion

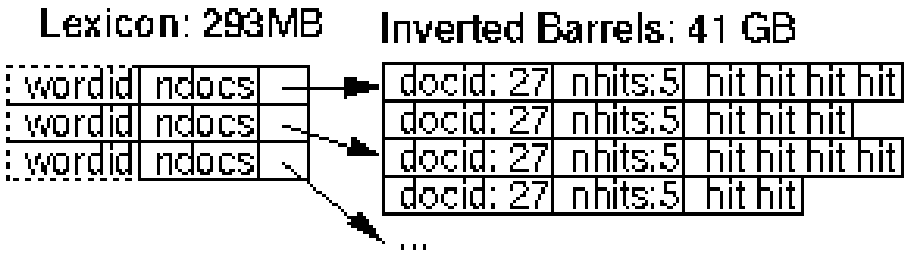
Forward Index



- Stored in 64 barrels
- If a document contains words that fall into a particular barrel, the docID is recorded into the barrel, followed by a list of wordID's with hit lists which correspond to those words
- Each wordID stored as a relative difference from the minimum wordID that falls into the barrel the wordID is in
- The Forward Index represents a transitory storage state

  ! Querying the forward index would require sequential iteration through each document and to each word to verify a matching document

Inverted Index



- Represents the data in the fully processed state
- Consists of the same barrels as the Forward Index, except that they have been processed by the Sorter
- Sorter takes each forward barrel and sorts them by wordID
- Each wordID contains a list of documents in which appears and the corresponding hit list
- Two sets of inverted barrels
  - One set which contains fancy hits (where the initial search is done)
  - Another set which contains plain hits
- Approx. 40 GB

Crawling the web

- The crawling is done by crawlers in a distributed fashion
- The crawlers were implemented in Python
  - Keeps 300 open connections
  - Keeps a DNS cache to avoid DNS lookups (requests to DNS servers for getting the ip address of a host)
- Speed: 100 webpages/second
- Cultural impact, people were not familiar with robots exclusion protocol

Indexing the Web

- Parsing
  - A complex operation which must handle a lot of possible errors
  - The parser is based on the flex tool available on http://flex.sourceforge.net/
    - Using flex you can generate lexical parsers written in C
    - Has strong performances and is very robust
- Indexing documents into barrels
  - The process in which the document content is converted into hit lists and then saved into the barrels. It results in a Forward index
  - This operation is handled by the Indexer. Multiple Indexers can run in parallel
- Sorting
  - The operation in which Forward Barrels are sorted based on wordID building the Reverted Index (short inverted barrels and long inverted barrels)
  - Because the barrels don't fit into memory they are splitted in baskets and then sorted

Searching

- Involves 8 distinct steps
- Is focused on quality
- Is limited to 40k items to limit the response time
- Has a complex ranking system
- Supports feedback

1. Parse the query.
2. Convert words into wordIDs.
3. Seek to the start of the doclist in the short barrel for every word.
4. Scan through the doclists until there is a document that matches all the search terms.
5. Compute the rank of that document for the query.
6. If we are in the short barrels and at the end of any doclist, seek to the start of the doclist in the full barrel for every word and go to step 4.
7. If we are not at the end of any doclist go to step 4. Sort the documents that have matched by rank and return the top k.

Searching - Ranking System

- The hits are classified based on type (title, anchor, URL, plain text large font)
- Every type is weighted based on their importance
- The formula for ranking a word inside a page is the following
  - $\boldsymbol{r(w) = \sum_i weight_i \cdot C_w(count_i(w))}$, where $\boldsymbol{count_i(w)}$ represents the number of appearances of the word w for type $\boldsymbol{i}$, $\boldsymbol{C_w}$ represents the "countweight" function, and $weight_i$ is the weight for type $\boldsymbol{i}$
- The total rank ***R(w)*** is a combination of **PageRank** and ***r(w)***

System Performance

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 Email Addresses	1.7 million
Number of 404's	1.6 million