How to build Google in 90 minutes
(or any other large web search engine)

Djoerd Hiemstra
University of Twente
http://www.cs.utwente.nl/~hiemstra
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing, query processing
Shake well…
Course objectives

• Get an understanding of the scale of “things”
• Being able to estimate index size and query time
• Applying simple index compressions schemes
• Applying simple optimizations
New web scale search engine

• How much money do we need for our startup?
Dear bank,

- We put the entire web index on a desktop PC and search it in reasonable time:
  a) probably
  b) maybe
  c) no
  d) no, are you crazy?
(Brin & Page 1998)
Google’s 10th birthday
1. The web server sends the query to the index servers. The content inside the index servers is similar to the index in the back of a book - it tells which pages contain the words that match the query.

2. The query travels to the doc servers, which actually retrieve the stored documents. Snippets are generated to describe each search result.

3. The search results are returned to the user in a fraction of a second.
Google’s 10\textsuperscript{th} birthday

- Google maintains the worlds largest cluster of commodity hardware (over 100,000 servers)
- These are partitioned between index servers and page servers (and more)
  - Index servers resolve the queries (massively parallel processing)
  - Page servers deliver the results of the queries: urls, title, snippets
- Over 20(?) billion web pages are indexed and served by Google
Google '98: Zlib compression

- A variant of LZ77 (gzip)

Repository: 53.5 GB = 147.8 GB uncompressed

<table>
<thead>
<tr>
<th>sync</th>
<th>length</th>
<th>compressed packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>sync</td>
<td>length</td>
<td>compressed packet</td>
</tr>
</tbody>
</table>

... Packet (stored compressed in repository)

| docid | ecode | urllen | pagelen | url | page |
Google '98: Forward & 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
  docid  wordid: 24  nhits: 8  hit hit hit hit
  null wordid
  docid  wordid: 24  nhits: 8  hit hit hit hit
  docid  wordid: 24  nhits: 8  hit hit hit hit
  docid  wordid: 24  nhits: 8  hit hit hit hit
  null wordid
...
```

Lexicon: 293MB

```
  wordid  ndocs
  wordid  ndocs
  wordid  ndocs
```

Inverted Barrels: 41 GB

```
  docid: 27  nhits: 5  hit hit hit hit
  docid: 27  nhits: 5  hit hit hit
  docid: 27  nhits: 5  hit hit hit
  docid: 27  nhits: 5  hit hit hit
  docid: 27  nhits: 5  hit hit
...
```
Google '98: Query evaluation

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.
Google'98: Storage numbers

<table>
<thead>
<tr>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Size of Fetched Pages</td>
<td>147.8 GB</td>
</tr>
<tr>
<td>Compressed Repository</td>
<td>53.5 GB</td>
</tr>
<tr>
<td>Short Inverted Index</td>
<td>4.1 GB</td>
</tr>
<tr>
<td>Full Inverted Index</td>
<td>37.2 GB</td>
</tr>
<tr>
<td>Lexicon</td>
<td>293 MB</td>
</tr>
<tr>
<td>Temporary Anchor Data (not in total)</td>
<td>6.6 GB</td>
</tr>
<tr>
<td>Document Index Incl. Variable Width Data</td>
<td>9.7 GB</td>
</tr>
<tr>
<td>Links Database</td>
<td>3.9 GB</td>
</tr>
<tr>
<td><strong>Total Without Repository</strong></td>
<td>55.2 GB</td>
</tr>
<tr>
<td><strong>Total With Repository</strong></td>
<td>108.7 GB</td>
</tr>
</tbody>
</table>
# Google'98: Page search

<table>
<thead>
<tr>
<th>Web Page Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Web Pages Fetched</td>
<td>24 million</td>
</tr>
<tr>
<td>Number of URLs Seen</td>
<td>76.5 million</td>
</tr>
<tr>
<td>Number of Email Addresses</td>
<td>1.7 million</td>
</tr>
<tr>
<td>Number of 404's</td>
<td>1.6 million</td>
</tr>
</tbody>
</table>
## Google'98: Search speed

<table>
<thead>
<tr>
<th>Query</th>
<th>Initial Query</th>
<th>Same Query Repeated (IO mostly cached)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU Time(s)</td>
<td>Total Time(s)</td>
<td>CPU Time(s)</td>
</tr>
<tr>
<td>al gore</td>
<td>0.09</td>
<td>2.13</td>
<td>0.06</td>
</tr>
<tr>
<td>vice president</td>
<td>1.77</td>
<td>3.84</td>
<td>1.66</td>
</tr>
<tr>
<td>hard disks</td>
<td>0.25</td>
<td>4.86</td>
<td>0.20</td>
</tr>
<tr>
<td>search engines</td>
<td>1.31</td>
<td>9.63</td>
<td>1.16</td>
</tr>
</tbody>
</table>
## How many pages? (November 2004)

<table>
<thead>
<tr>
<th>Search Engine</th>
<th>Reported Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>8.1 billion</td>
</tr>
<tr>
<td>Microsoft</td>
<td>5.0 billion</td>
</tr>
<tr>
<td>Yahoo</td>
<td>4.2 billion</td>
</tr>
<tr>
<td>Ask</td>
<td>2.5 billion</td>
</tr>
</tbody>
</table>

[http://blog.searchenginewatch.com/blog/041111-084221](http://blog.searchenginewatch.com/blog/041111-084221)
How many pages?

Table 3.1 Parameters of document collections.

(Witten, Moffat, Bell, 1999)
<table>
<thead>
<tr>
<th>Service</th>
<th>Searches per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>180 million</td>
</tr>
<tr>
<td>Yahoo</td>
<td>70 million</td>
</tr>
<tr>
<td>Microsoft</td>
<td>30 million</td>
</tr>
<tr>
<td>Ask</td>
<td>13 million</td>
</tr>
</tbody>
</table>
Popularity (in the US)

Source: NetRatings for SearchEngineWatch.com

- Google, 46.2%
- Yahoo, 22.5%
- MSN, 12.6%
- AOL, 5.4%
- Ask, 1.6%
- My Way, 2.2%
- Netscape, 1.6%
- iWon, 0.9%
- Dogpile, 0.9%
- EarthLink, 0.8%
- Others, 5.3%

http://searchenginewatch.com/reports/
Searching the web

• How much data are we talking about?
  – About 10 billion pages
  – Assume a page contains 200 terms on average
  – Each term consists of 5 characters on average
  – To store the web you need to search:
    • $10^{10} \times 200 \times 5 \approx 10 \text{ TB}$
Some more stuff to store?

- Text statistics:
  - Term frequency
  - Collection frequency
  - Inverse document frequency …

- Hypertext statistics:
  - Ingoing and outgoing links
  - Anchor text
  - Term positions, proximities, sizes, and characteristics …
How fast can we search 10 TB?

• We need to find a large hard disk
  – Size: 1.5 TB
  – Hard disk transfer time 100 MB/s

• Time needed to sequentially scan the
  – 100,000 seconds …
  – … so, we have to wait for 28 hours to get the answer to one (1) query

• We can definitely do better than that!
Problems in web search

- Web crawling
  - politeness, freshness, duplicates, missing links, loops, server problems, virtual hosts, etc.
- Maintain large cluster of servers
  - Page servers: store and deliver the results of the queries
  - Index servers: resolve the queries
- Answer 100 million of user queries per day
  - Caching, replicating, parallel processing, etc.
  - **Indexing, compression, coding**, fast access, etc.
Implementation issues

• Analyze the collection
  – Avoid non-informative data for indexing
  – Decision on relevant statistics and info

• Index the collection
  – How to organize the index?

• Compress the data
  – Data compression
  – Index compression
Ingredients of this talk:

1. A bit of high school mathematics
1. Zipf's law
1. Indexing, query processing

Shake well…
Zipf's law

• Count how many times a term occurs in the collection
  – call this $f$

• Order them in descending order
  – call the rank $r$

• Zipf's claim:
  – For each word, the product of frequency and rank is approximately constant: $f \times r = c$
Zipf distribution

Term count

Terms by rank order

Linear scale
Zipf distribution

Term count

Logarithmic scale

Terms by rank order
Consequences

• Few terms occur very frequently: a, an, the, … => non-informative (stop) words
• Many terms occur very infrequently: spelling mistakes, foreign names, …
• Medium number of terms occur with medium frequency
Word resolving power

Figure 2.1. A plot of the hyperbolic curve relating \( f \), the frequency of occurrence and \( r \), the rank order. (Adapted from Schultz [4] page 122)

(Van Rijsbergen 79)
Heap’s law for dictionary size

number of unique terms

collection size
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
   1. Indexing

Shake well…
### Example

<table>
<thead>
<tr>
<th>Document number</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pease porridge hot, pease porridge cold</td>
</tr>
<tr>
<td>2</td>
<td>Pease porridge in the pot</td>
</tr>
<tr>
<td>3</td>
<td>Nine days old</td>
</tr>
<tr>
<td>4</td>
<td>Some like it hot, some like it cold</td>
</tr>
<tr>
<td>5</td>
<td>Some like it in the pot</td>
</tr>
<tr>
<td>6</td>
<td>Nine days old</td>
</tr>
</tbody>
</table>

Stop words: in, the, it.

(Witten, Moffat & Bell, 1999)
Inverted index

table: term, offset, Documents

cold  2  1, 4

days  4  3, 6

hot   6  1, 4

like  8  4, 5

nine 10  3, 6

old  12  3, 6

pease 14  1, 2

porridge 16  1, 2

pot  18  2, 5

some  20  4, 5

dictionary postings
Size of the inverted index?
Size of the inverted index

- Number of postings (term-document pairs):
  - Number of documents: \( \sim 10^{10} \),
  - Average number of unique terms per document (document size \( \sim 200 \)): \( \sim 100 \)
  - 5 bytes for each posting (why?)
  - So, \( 10^{10} \times 100 \times 5 = 5 \text{ TB} \)
  - postings take half the size of the data
Size of the inverted index

- Number of unique terms is, say, \(10^8\)
  - 6 bytes on average
  - plus off-set in postings, another 8 bytes
  - So, \(10^8 \times 14 = 1.4\) GB
  - So, dictionary is tiny compared to postings (0.03 %)
- Another optimization (Galago):
  - sort dictionary alphabetically
  - at maximum one vocabulary entry for each 32 KB block
Inverted index encoding

• The inverted file entries are usually stored in order of increasing document number

  – [<retrieval; 7; [2, 23, 81, 98, 121, 126, 180]>

  (the term “retrieval” occurs in 7 documents with document identifiers 2, 23, 81, 98, etc.)
Query processing (1)

• Each inverted file entry is an ascending ordered sequence of integers
  – allows merging (joining) of two lists in a time linear in the size of the lists
Query processing (2)

• Usually queries are assumed to be conjunctive queries
  – query: *information retrieval*
  – is processed as *information AND retrieval*

    [<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>]
    [<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>]

  – intersection of posting lists gives:
    [23, 98]
Query processing (3)

• Remember the Boolean model?
  – intersection, union and complement is done on posting lists
  – so, information OR retrieval
    
      [<retrieval; 7; [2, 23, 81, 98, 121, 126, 139]>
    
      [<information; 9; [1, 14, 23, 45, 46, 84, 98, 111, 120]>
    
  – union of posting lists gives:
    
      [1, 2, 14, 23, 45, 46, 81, 84, 98, 111, 120, 121, 126, 139]
Query processing (4)

• Estimate of selectivity of terms:
  – Suppose *information* occurs on 1 billion pages
  – Suppose *retrieval* occurs on 10 million pages
Query processing (4)

• Estimate of selectivity of terms:
  – Suppose *information* occurs on 1 billion pages
  – Suppose *retrieval* occurs on 10 million pages

• size of postings (5 bytes per docid):
  – 1 billion * 5B = 5 GB for *information*
  – 10 million * 5B = 50 MB for *retrieval*

• Hard disk transfer time:
  – 50 sec. for *information* + 0.5 sec. for *retrieval*
  – (ignore CPU time and disk latency)
Query processing (5)

- We just brought query processing down from 28 hours to just 50.5 seconds (!)
  :-)

- Still... way too slow...
  :-(

Inverted file compression (1)

• **Trick 1**, store sequence of doc-ids:
  – [<retrieval; 7; [2, 23, 81, 98, 121, 126, 180]>

as a sequence of gaps
  – [<retrieval; 7; [2, 21, 58, 17, 23, 5, 54]>

• No information is lost.
• Always process posting lists from the beginning, so easily decoded into the original sequence
Inverted file compression (2)

• Does it help?
  – maximum gap determined by the number of indexed web pages...
  – infrequent terms coded as a few large gaps
  – frequent terms coded by many small gaps

• Trick 2: use variable byte length encoding.
Variable byte encoding (1)

<table>
<thead>
<tr>
<th>Gap $x$</th>
<th>Coding Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unary</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>1110</td>
</tr>
<tr>
<td>5</td>
<td>11110</td>
</tr>
<tr>
<td>6</td>
<td>111110</td>
</tr>
<tr>
<td>7</td>
<td>1111110</td>
</tr>
<tr>
<td>8</td>
<td>11111110</td>
</tr>
<tr>
<td>9</td>
<td>111111110</td>
</tr>
<tr>
<td>10</td>
<td>1111111110</td>
</tr>
</tbody>
</table>

Table 3.5  Example codes for integers.

(Witten, Moffat & Bell, 1999)
Variable byte encoding (2)

- γ code: represent number $x$ as:
  - first bits as the unary code for $1 + \left\lfloor 2 \log x \right\rfloor$
  - remainder bits as binary code for $x - 2^{\left\lfloor \log x \right\rfloor}$
  - unary part (minus 1) specifies how many bits are required to code the remainder part

- For example $x = 5$:
  - first bits: 110 ($1 + \left\lfloor 2 \log 5 \right\rfloor = 1 + \left\lfloor 2.32 \right\rfloor = 3$)
  - remainder: 01 ($5 - 2^{\left\lfloor \log 5 \right\rfloor} = 5 - 2^2 = 1$)
## Index sizes

<table>
<thead>
<tr>
<th>Method</th>
<th>Bits per pointer</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bible</td>
<td>GNUbib</td>
<td>Comact</td>
<td>TREC</td>
</tr>
<tr>
<td><strong>Global methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unary</td>
<td>264</td>
<td>920</td>
<td>490</td>
<td>1719</td>
</tr>
<tr>
<td>Binary</td>
<td>15.00</td>
<td>16.00</td>
<td>18.00</td>
<td>20.00</td>
</tr>
<tr>
<td>Bernoulli</td>
<td>9.67</td>
<td>11.65</td>
<td>10.58</td>
<td>12.61</td>
</tr>
<tr>
<td>γ</td>
<td>6.55</td>
<td>5.69</td>
<td>4.48</td>
<td>6.43</td>
</tr>
<tr>
<td>δ</td>
<td>6.26</td>
<td>5.08</td>
<td>4.36</td>
<td>6.19</td>
</tr>
<tr>
<td>Observed frequency</td>
<td>5.92</td>
<td>4.83</td>
<td>4.21</td>
<td>5.83</td>
</tr>
<tr>
<td><strong>Local methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bernoulli</td>
<td>6.13</td>
<td>6.17</td>
<td>5.40</td>
<td>5.73</td>
</tr>
<tr>
<td>Hyperbolic</td>
<td>5.77</td>
<td>5.17</td>
<td>4.65</td>
<td>5.74</td>
</tr>
<tr>
<td>Skewed Bernoulli</td>
<td>5.68</td>
<td>4.71</td>
<td>4.24</td>
<td>5.28</td>
</tr>
<tr>
<td>Batched frequency</td>
<td>5.61</td>
<td>4.65</td>
<td>4.03</td>
<td>5.27</td>
</tr>
</tbody>
</table>

Table 3.7  Compression of inverted files, in bits per pointer.

(Witten, Moffat & Bell, 1999)
Index size of our search engine?
Index size of our search engine

• Number of postings (term-document pairs):
  – 10 billion documents
  – 100 unique terms on average
  – Assume on average 6 bits per doc-id
  – $10^{10} \times 100 \times 6 \text{ bits} \approx 750 \text{ GB}$
  – about 15% of the uncompressed inverted file.

• It nicely fits our 1 TB hard drive :-}
Query processing on compressed index

• size of postings (6 bits per docid):
  – 1 billion * 6 bits = 750 Mb for "information"
  – 10 million * 6 bits = 7.5 Mb for "retrieval"

• Hard disk transfer time:
  – 7.5 sec. for information + 0.08 sec. for retrieval
  – (ignore CPU time and disk latency)
We already brought down query processing from more than 1 day to 50.5 seconds...
and brought that down to 7.58 seconds :-)

but that is still too slow... :-(

Query processing – Continued (1)
Google PageRank
(Brin & Page 1998)

• Suppose a million monkeys browse the www by randomly following links
• At any time, what percentage of the monkeys do we expect to look at page \( D \)?
• Compute the probability, and use it to rank the documents that contain all query terms
Google PageRank

- Given a document \( D \), the documents page rank at step \( n \) is:

\[
P_n(D) = (1 - \lambda) P_0(D) + \lambda \left( \sum_{I \text{ linking to } D} P_{n-1}(I) P(D|I) \right)
\]

- where

\( P(D | I) \) : probability that the monkey reaches page \( D \) through page \( I \) (\( = 1 / \# \text{outlinks of } I \))

\( \lambda \) : probability that the follows a link

\( 1 - \lambda \) : probability that the monkey types a url
Early termination (1)

- Suppose we re-sort the document ids for each posting such that the best documents come first
  - e.g., sort document identifiers for "retrieval" by their tf.idf values.
  - [<retrieval; 7; [98, 23, 180, 81, 98, 121, 2, 126,]>]
  - then: top 10 documents for the query "retrieval" can be retrieved very quickly: stop after processing the first 10 document ids from the posting list!
  - but compression and merging (multi-word queries) of postings no longer possible...
Early termination (2)

- **Trick 3**: define a static (or global) ranking of all documents
  - such as Google PageRank (!)
  - re-assign document identifiers by ascending PageRank
  - For every term, documents with a high PageRank are in the initial part of the posting list
  - Estimate the selectivity of the query and only process part of the posting files.

(see e.g. Croft, Metzler & Strohman 2009)
Early termination (3)

- Probability that a document contains a term:
  - $1 \text{ billion} / 10 \text{ billion} = 0.1$ for *information*
  - $10 \text{ million} / 10 \text{ billion} = 0.001$ for *retrieval*

- Assume independence between terms:
  - $0.1 \times 0.001 = 0.0001$ of the documents contains both terms
  - so, every $1 / 0.0001 = 10,000$ documents on average contains *information AND retrieval*.
  - for top 30, process $3,000,000$ documents.
  - $3,000,000 / 10 \text{ billion} = 0.0003$ of the posting files
Query processing on compressed index with early termination

• process about 0.0003 of postings:
  – 0.0003 * 750 Mb = 225 kb for information
  – 0.0003 * 7.5 Mb = 2.25 kb for retrieval

• Hard disk transfer time:
  – 2 msec. for information + 0.02 msec. for retrieval
  – (NB now, ignoring CPU time, disk latency and decompressing time is no longer reasonable, so it is likely that it takes some more time)
Query processing – Continued (2)

• We just brought query processing down from 1 day to about 2 ms. !

:-)

“This engine is incredibly, amazingly, ridiculously fast!”

(from “Top Gear”)
Indexing - Recap

• Inverted files
  – dictionary & postings
  – merging of posting lists
  – delta encoding + variable byte encoding
  – static ranking + early termination

• Put the entire web index on a desktop PC and search it in reasonable time:
  
a) probably
Ingredients of this talk:

1. A bit of high school mathematics
2. Zipf's law
3. Indexing

Shake well…
Summary

• Term distribution and statistics
• Indexing techniques (inverted files)
• Compression, coding, and querying
References


• Bruce Croft, Donald Metzler, and Trevor Strohman, Search Engines: information retrieval in practice, Pearson, 2009

• Keith van Rijsbergen, Information Retrieval, Butterworths, 1979

• Ian H. Witten, Alistar Moffat, Timothy C. Bell, “Managing Gigabytes”, Morgan Kaufmann, pages 72-115 (Section 3), 1999
Acknowledgements

- Thanks to the following people for contributing slides:
  - Vojkan Mihajlovic (Philips Research)