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

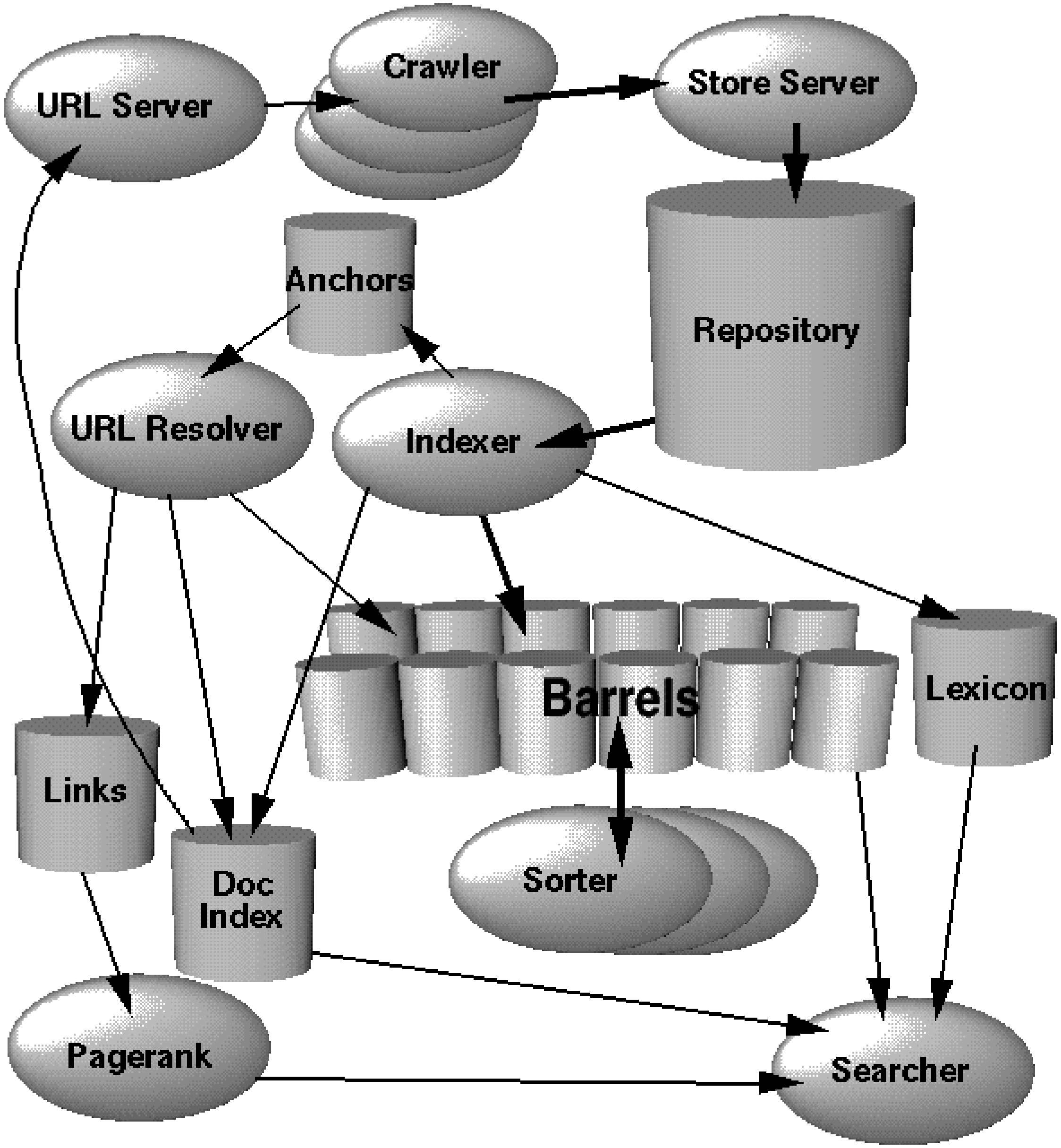
  c) no

  d) no, are you crazy?

Search the web using Google

More Google!

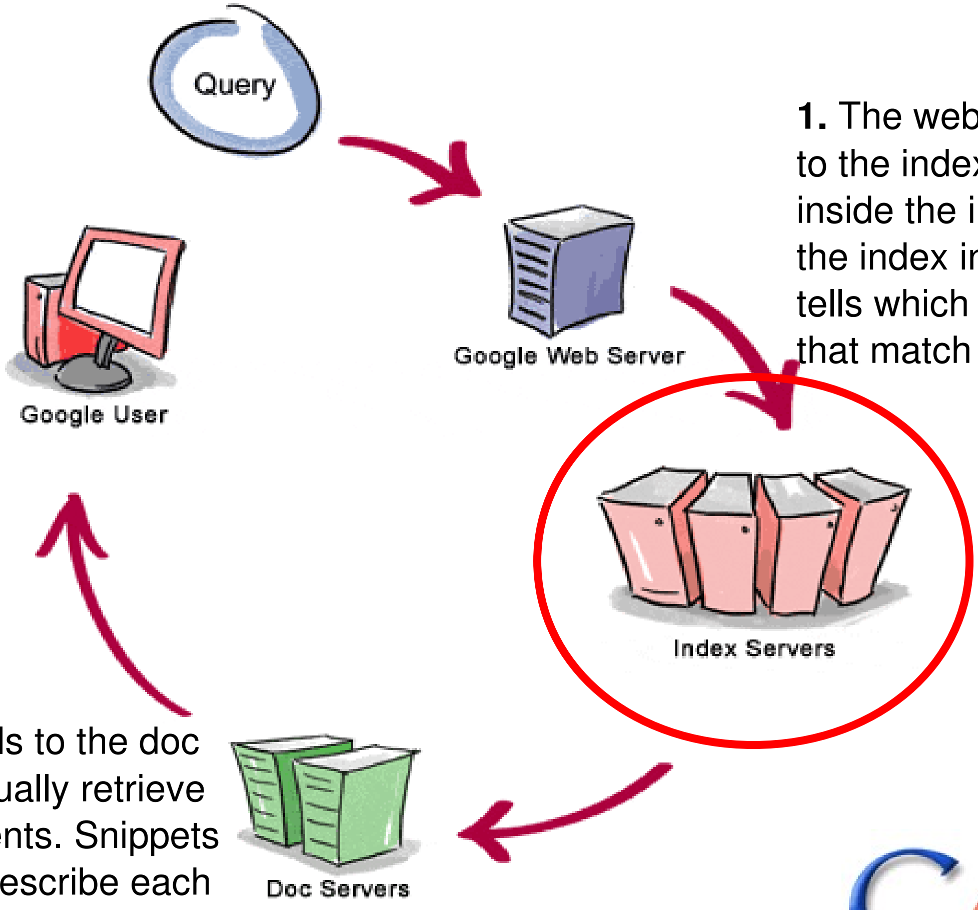
Copyright ©1999 Google Inc.



(Brin & Page 1998)

Google’s 10th birthday

Architecture today



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.

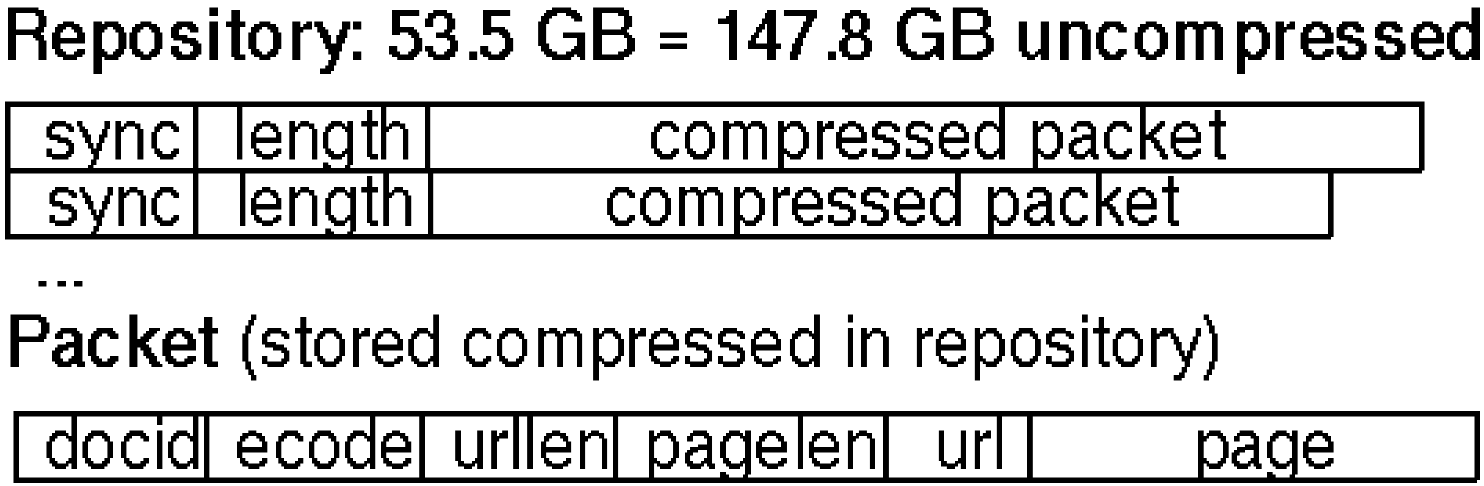
3. The search results are returned to the user in a fraction of a second.

Google's 10th 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

sync | length | compressed packet

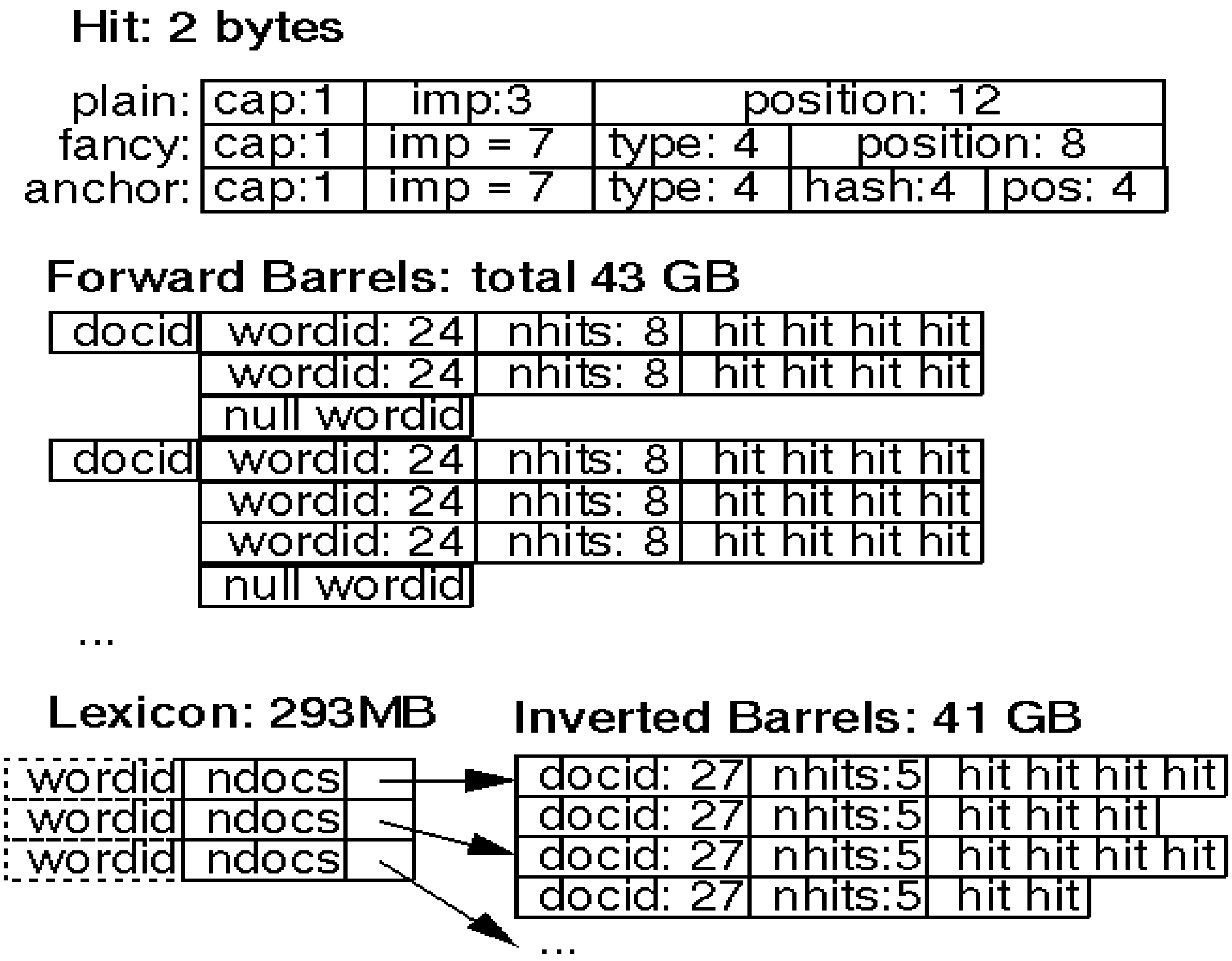
sync | length | compressed packet

...

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
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

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 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

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**

Google'98: Page search

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

Google'98: Search speed

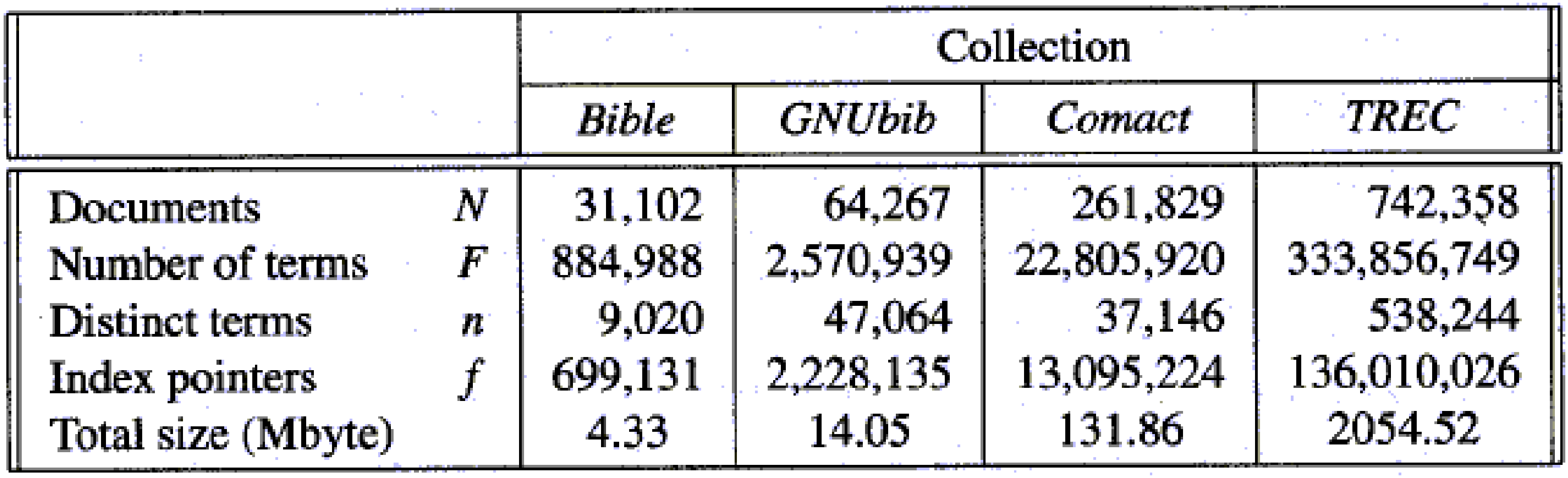
	Initial Query		**Same Query Repeated (IO mostly cached)**	
Query	**CPUTime(s)**	**Total Time(s)**	**CPU Time(s)**	**Total Time(s)**
al gore	0.09	2.13	0.06	0.06
vice president	1.77	3.84	1.66	1.80
hard disks	0.25	4.86	0.20	0.24
search engines	1.31	9.63	1.16	1.16

How many pages? (November 2004)

Search Engine	**Reported Size**
Google	8.1 billion
Microsoft	5.0 billion
Yahoo	4.2 billion
Ask	2.5 billion

http://blog.searchenginewatch.com/blog/041111-084221

How many pages?



		Collection			
		Bible	*GNUbib*	*Comact*	*TREC*
Documents	N	31,102	64,267	261,829	742,358
Number of terms	F	884,988	2,570,939	22,805,920	333,856,749
Distinct terms	n	9,020	47,064	37,146	538,244
Index pointers	f	699,131	2,228,135	13,095,224	136,010,026
Total size (Mbyte)		4.33	14.05	131.86	2054.52

Table 3.1 Parameters of document collections.

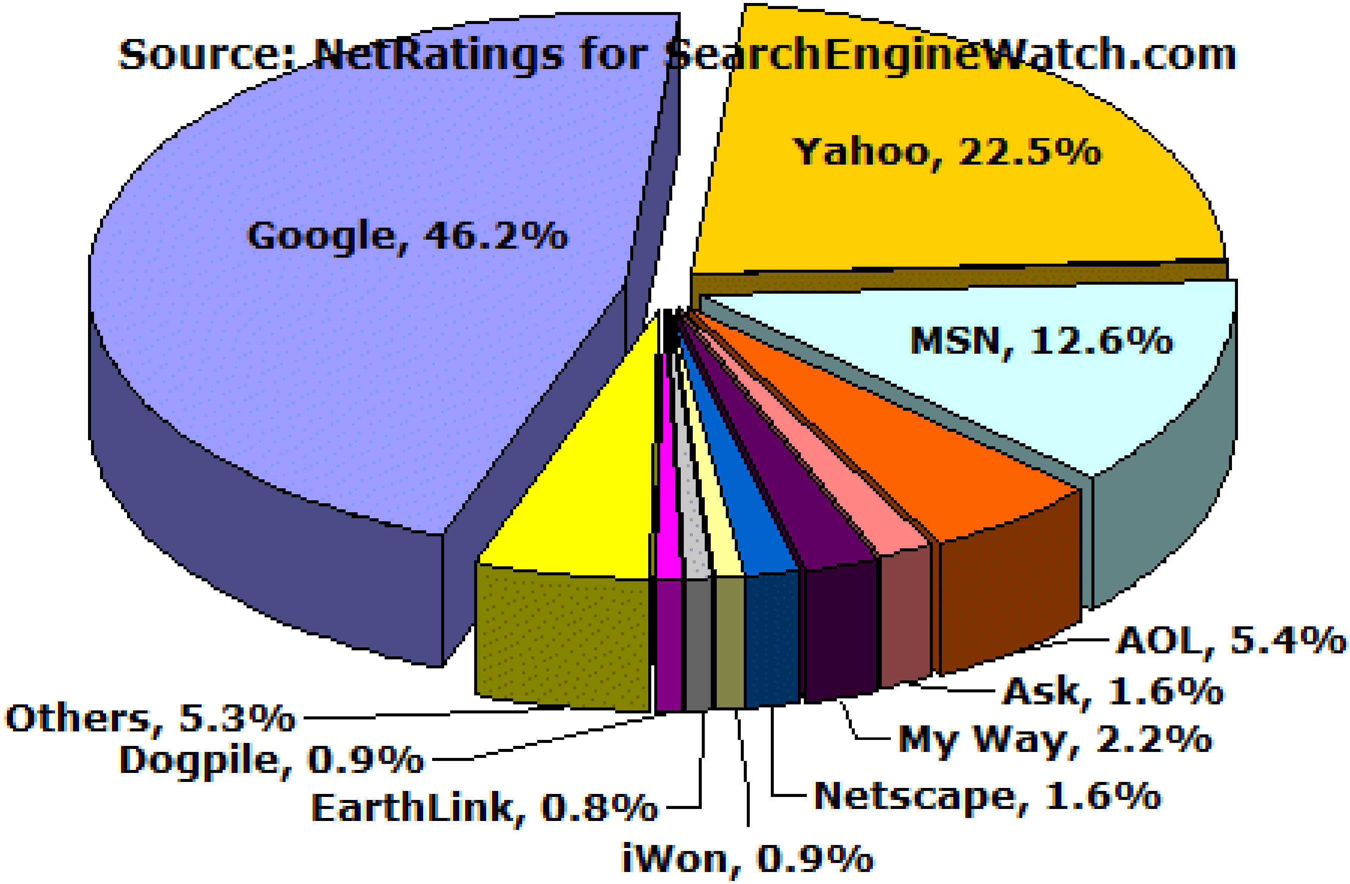
(Witten, Moffat, Bell, 1999)

Queries per day? (December 2007)

Service	**Searches per day**
Google	180 million
Yahoo	70 million
Microsoft	30 million
Ask	13 million

http://searchenginewatch.com/reports/

Popularity (in the US)



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} x 200 x 5 ~= 10 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:

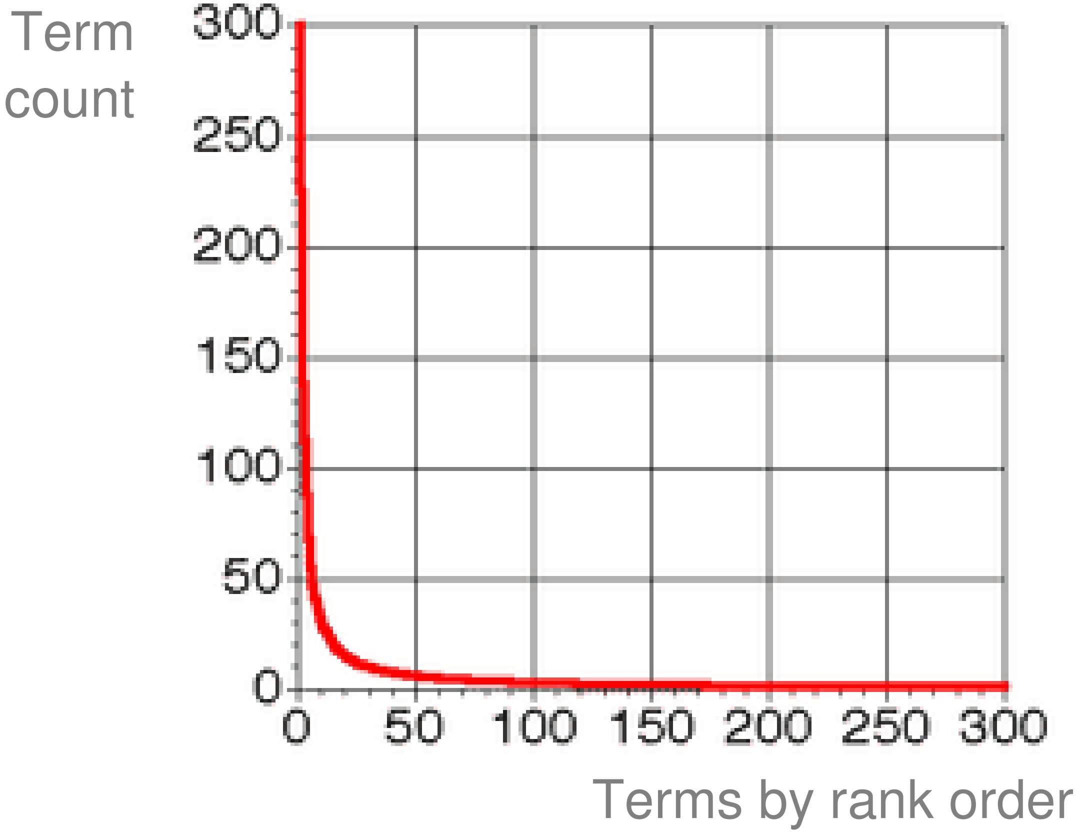
1. A bit of high school mathematics
1. Zipf's law
1. Indexing, query processing

   Shake well…

Zipf's law

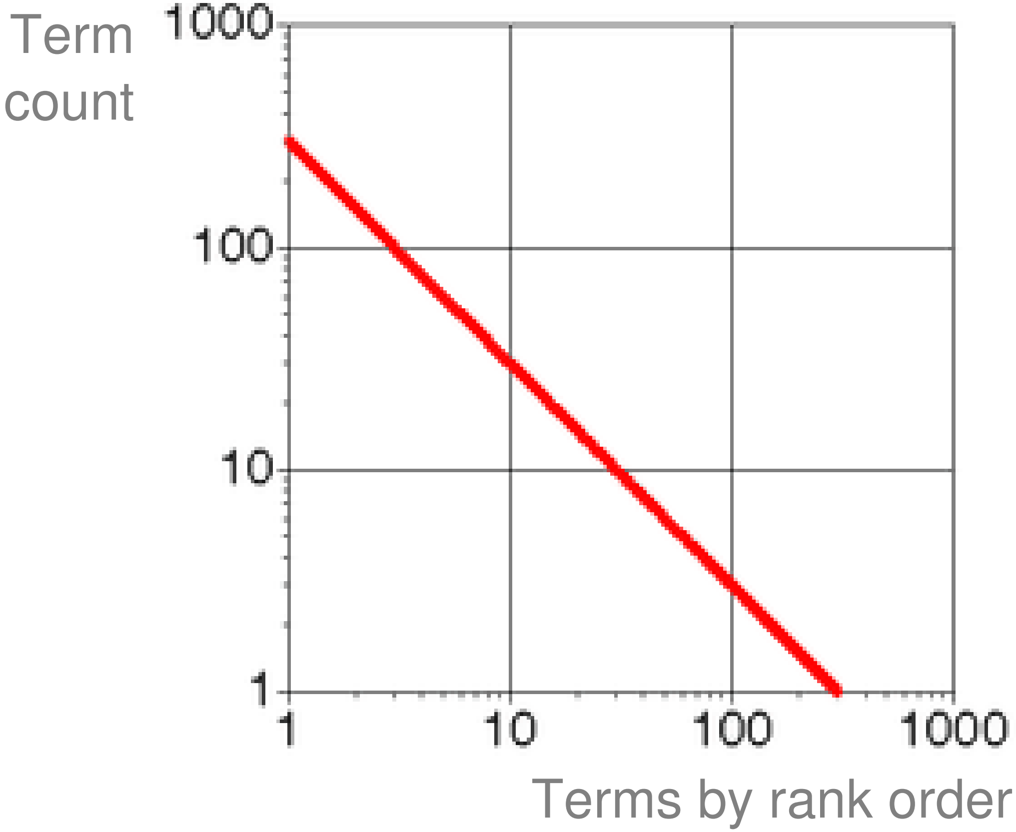
- Count how many times a term occurs in the collection
  - call this *f*
- Order them in descending order
  - call the rank *r*
- Zipf's claim:
  - For each word, the product of frequency and rank is approximatel constant: $f \times r = c$

Zipf distribution



Linear scale

Zipf distribution



Logarithmic scale

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

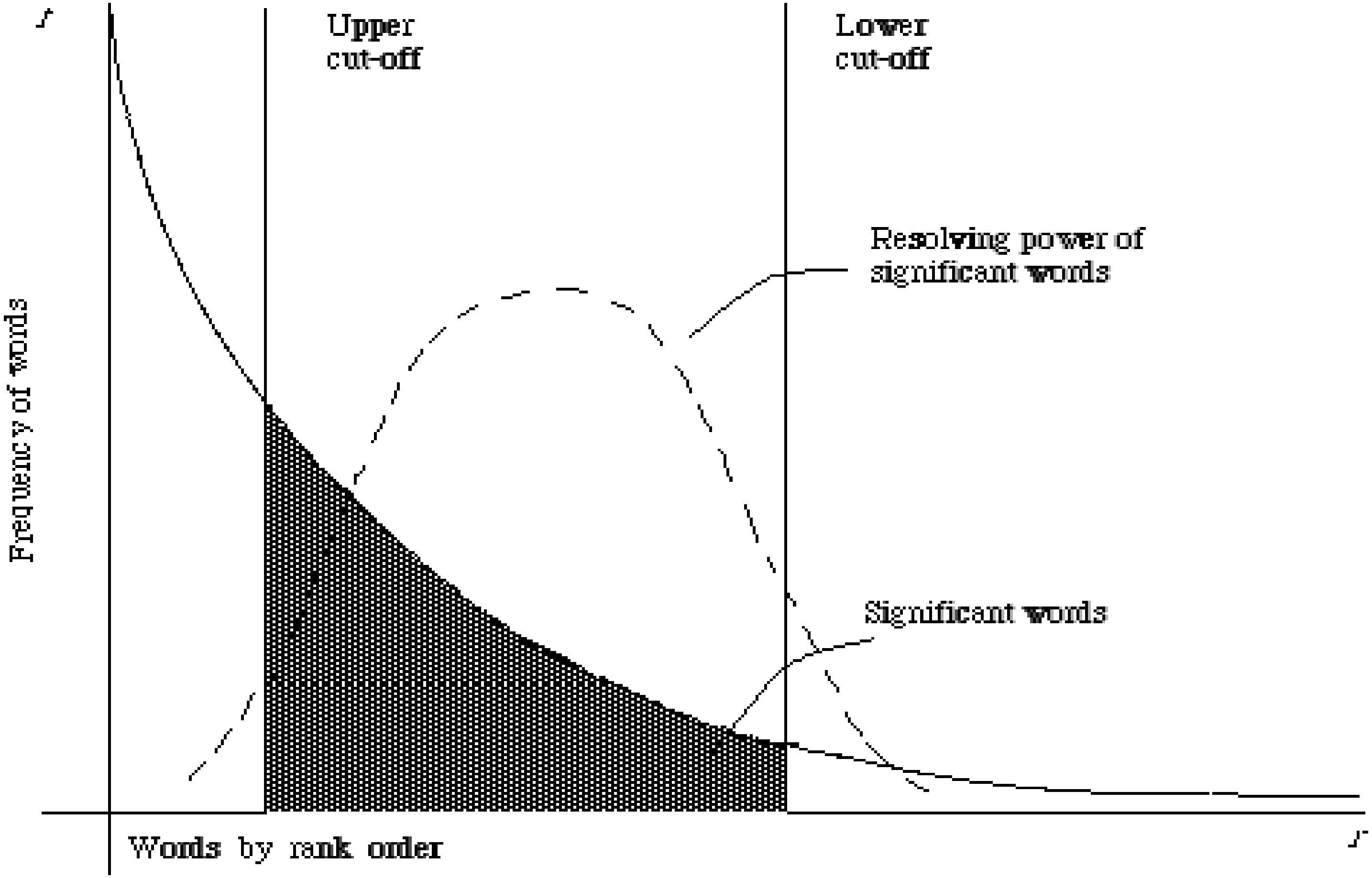
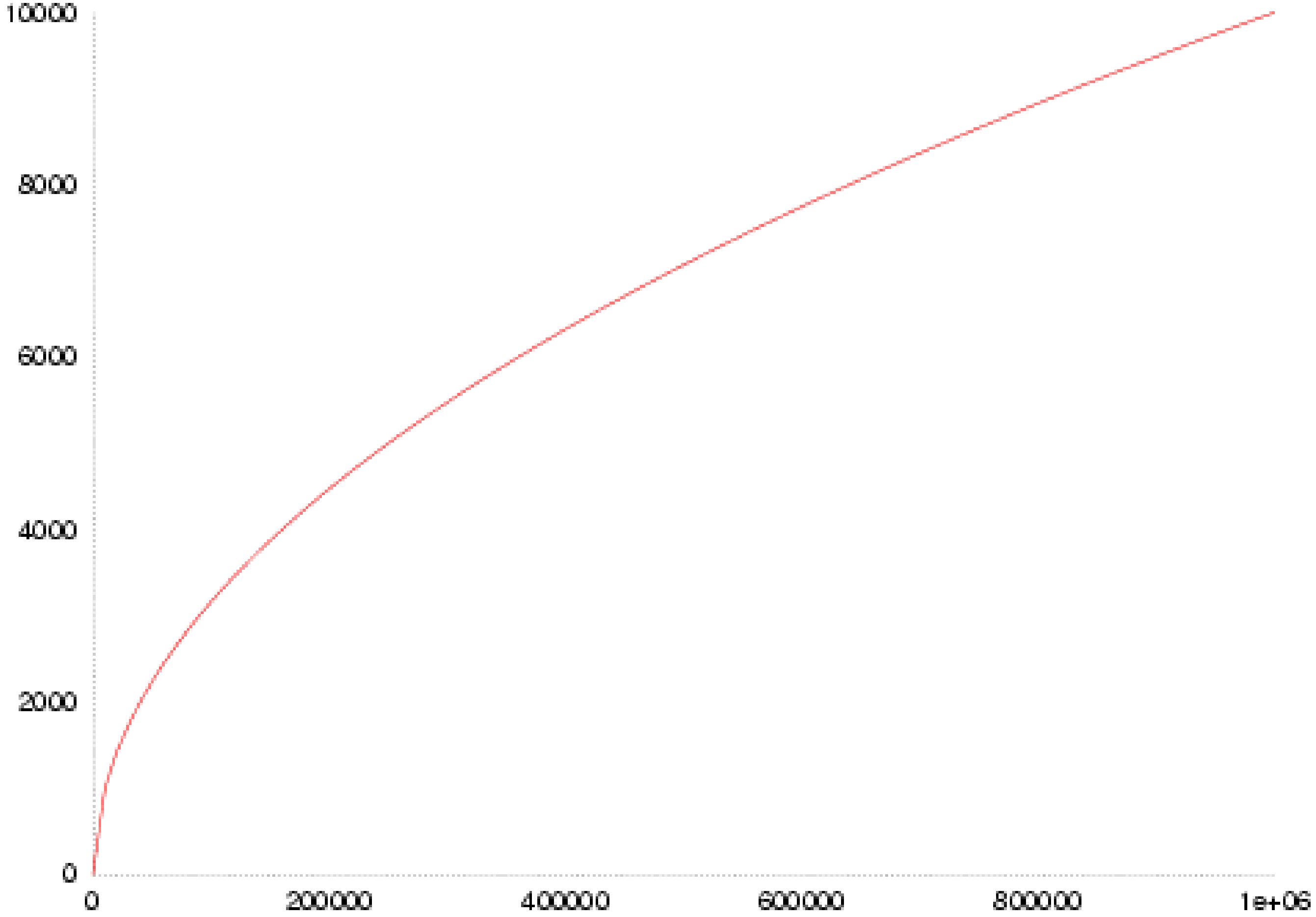


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

(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…

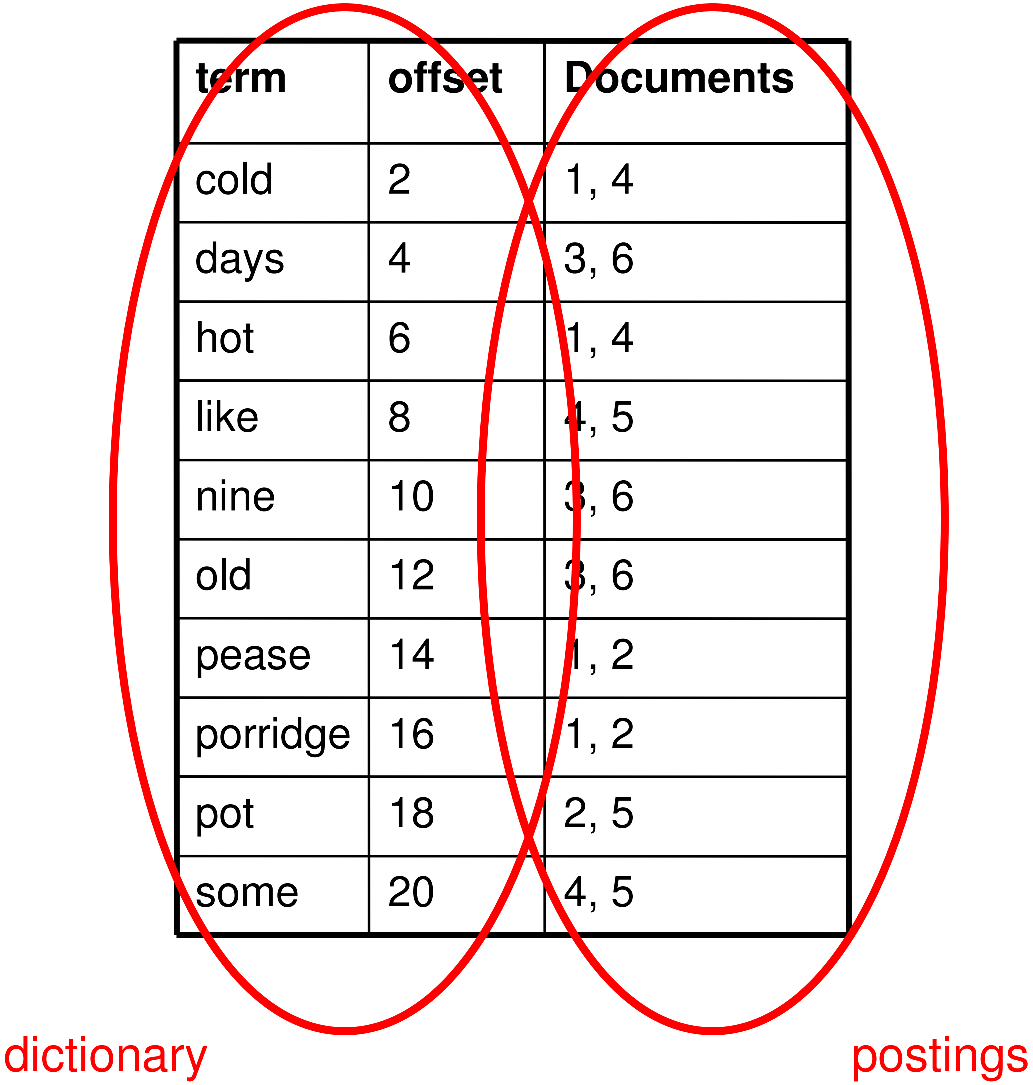
Example

Document number	Text
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Some like it in the pot
6	Nine days old

Stop words: in, the, it.

(Witten, Moffat & Bell, 1999)

Inverted index



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

Size of the inverted index

?

Size of the inverted index

- Number of postings (term-document pairs):
  - Number of documents: ~10^{10},
  - Average number of unique terms per document (document size ~200): ~100
  - 5 bytes for each posting (why?)
  - So, 10^{10} x 100 x 5 = 5 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 x 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]>
  - <u>intersection</u> 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]>
  - <u>union</u> 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]>

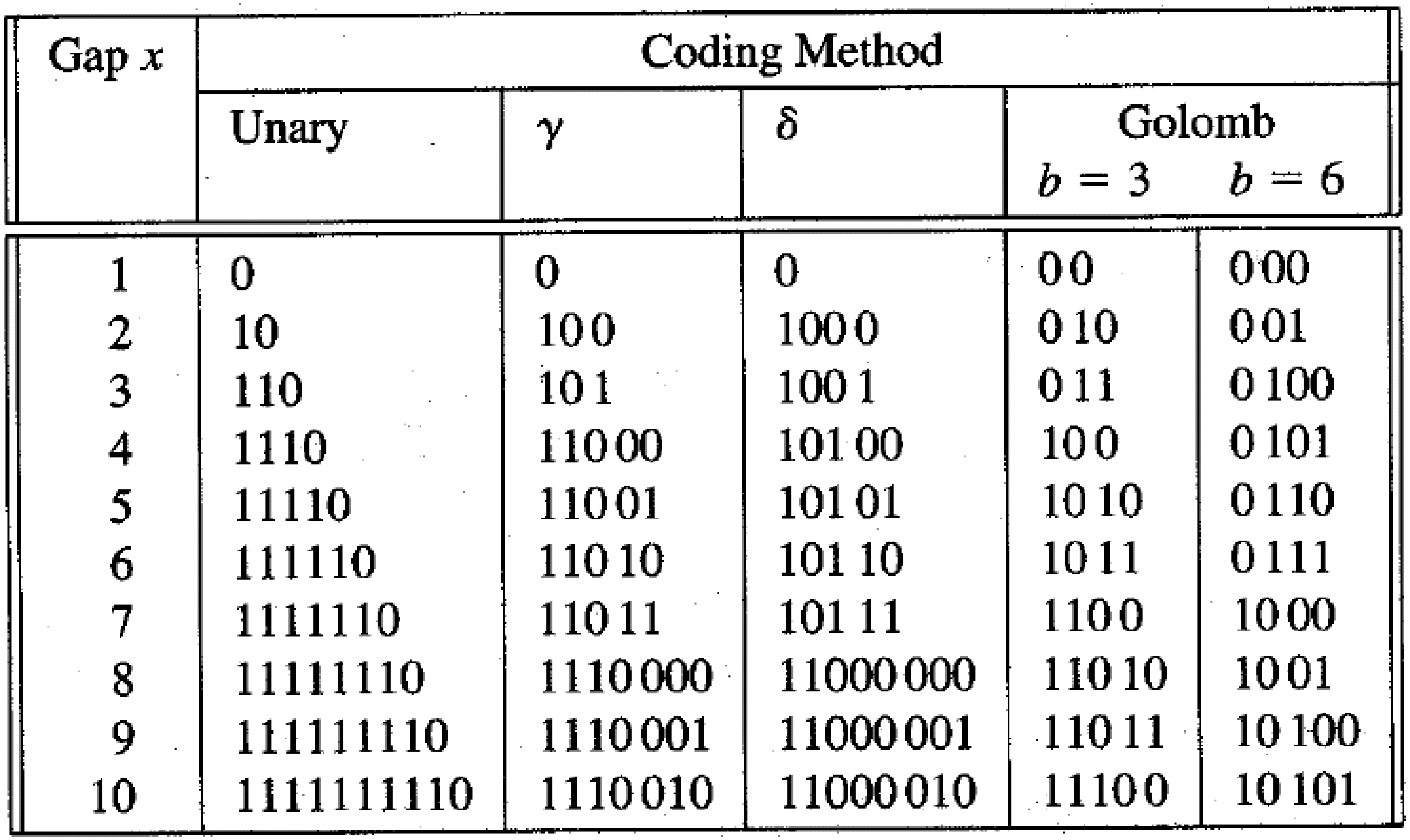
- No information is lost.
- Always process posting lists from the beginning, so easily decoded into the original sequence

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)



Gap x	Coding Method				
	Unary	γ	δ	Golomb $b = 3$	Golomb $b = 6$
1	0	0	0	0 0	0 00
2	10	10 0	100 0	0 10	0 01
3	110	10 1	100 1	0 11	0 100
4	1110	110 00	101 00	10 0	0 101
5	11110	110 01	101 01	10 10	0 110
6	111110	110 10	101 10	10 11	0 111
7	1111110	110 11	101 11	110 0	10 00
8	11111110	1110 000	11000 000	110 10	10 01
9	111111110	1110 001	11000 001	110 11	10 100
10	1111111110	1110 010	11000 010	1110 0	10 101

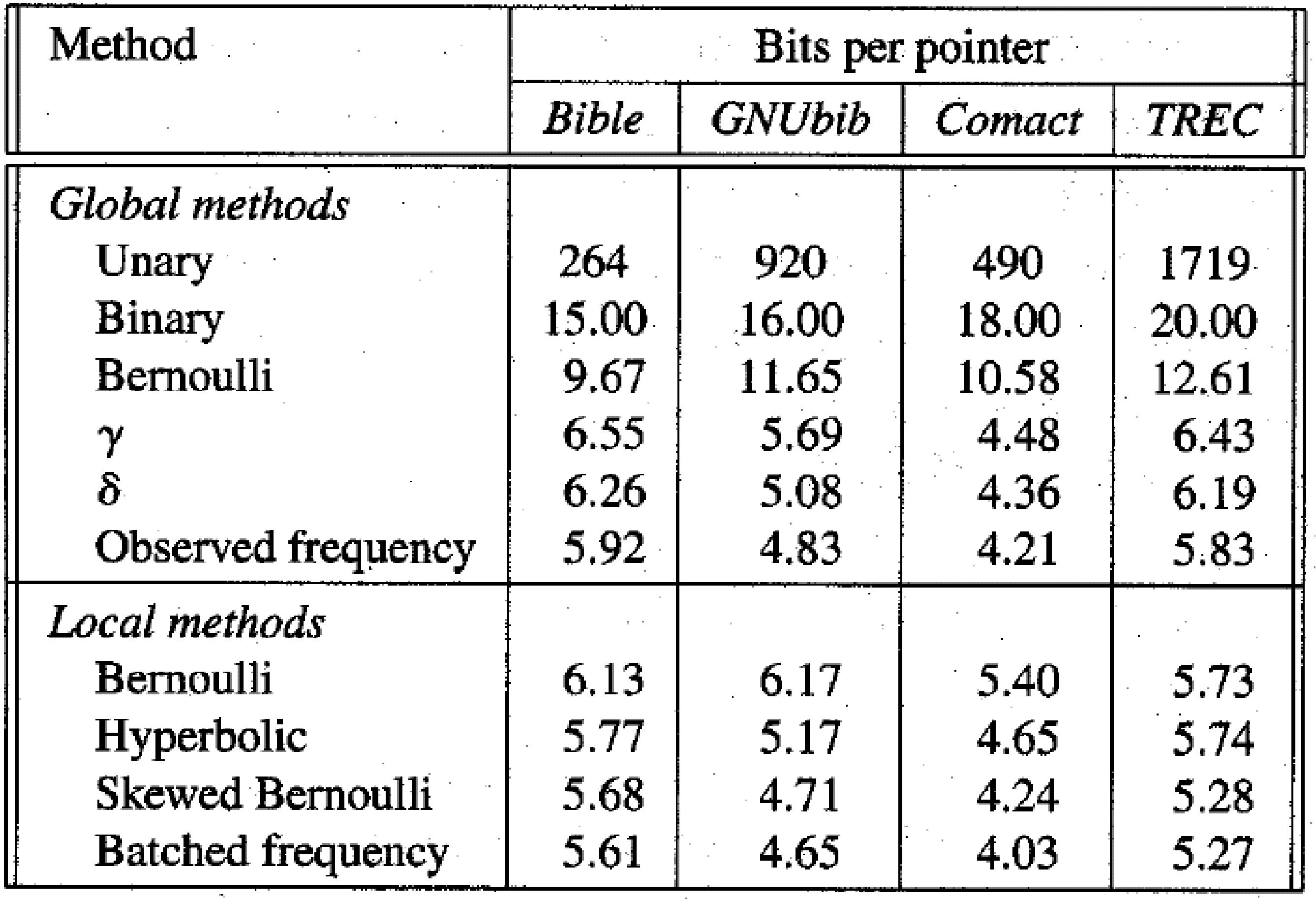
Table 3.5 Example codes for integers.

(Witten, Moffat & Bell, 1999)

Variable byte encoding (2)

- γ code: represent number x as:
  - first bits as the unary code for $1+\lfloor {}^{2}\log x \rfloor$
  - remainder bits as binary code for $x-2^{\lfloor {}^{2}\log x \rfloor}$
  - unary part (minus 1) specifies how many bits are required to code the remainder part
- For example $x = 5$:
  - first bits: 110 $\left(1+\lfloor {}^{2}\log 5 \rfloor = 1+\lfloor 2.32 \rfloor = 3\right)$
  - remainder: 01 $\left(5-2^{\lfloor {}^{2}\log 5 \rfloor} = 5-2^{2} = 1\right)$

Index sizes



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

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} x 100 x 6 bits ~= 750 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)

Query processing – Continued (1)

- 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...

  :-(

INTERMEZZO

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

INTERMEZZO

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

$P(D \mid I)$: probability that the monkey reaches page *D* through page *I* (= 1 / #outlinks of *I*)

λ : 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 Page-Rank 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 billion / 10 billion = 0.1 for *information*
  - 10 million / 10 billion = 0.001 for *retrieval*
- Assume independence between terms:
  - 0.1 x 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 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

- Sergey Brin and Lawrence Page, “**The Anatomy of a Large-Scale Hypertextual Web Search Engine**”, Computer Networks and ISDN Systems, 1998
- 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)