

## Indexing and Query Processing 2

---

CS 510 Winter 2007

1

## Index building

---

### Basic problem

Have:

d1:  $f_{d1,t3}$ ,  $f_{d1,t6}$ ,  $f_{d1,t19}$ , ...

d2:  $f_{d2,t1}$ ,  $f_{d2,t99}$ ,  $f_{d2,t110}$ , ...

...

Need:

t1:  $f_{d2,t1}$ ,  $f_{d11,t1}$ ,  $f_{d19,t1}$ , ...

t2:  $f_{d7,t2}$ ,  $f_{d8,t2}$ ,  $f_{d55,t2}$ , ...

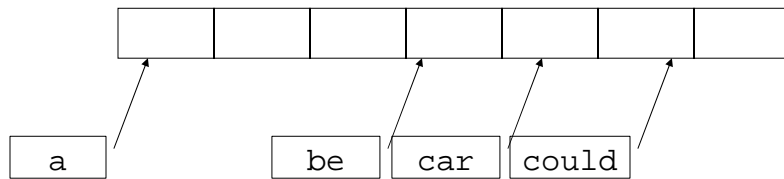
...

CS 510 Winter 2007

2

## In-Memory Inversion

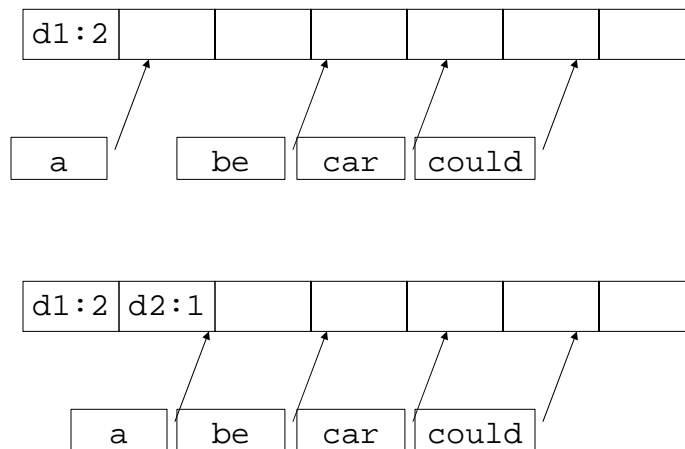
- Make a pass over all documents, figuring out  $f_t$  for each  $t$ .
- Then pre-allocate spans of memory for each inverted list.



CS 510 Winter 2007

3

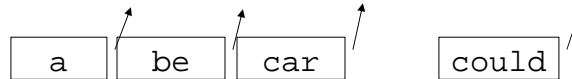
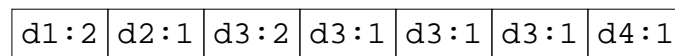
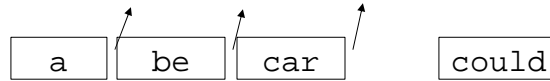
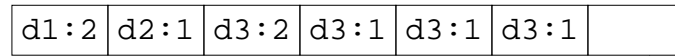
## Example Continued



CS 510 Winter 2007

4

## Example Continued



CS 510 Winter 2007

5

## Sort-based inversion

Have tuples  $(t, d, f_{d,t})$

- Ordered on  $d$
- Can sort on  $t$
- Build inverted lists from sorted sequence
- Want a stable sort, such as merge sort
- Might benefit to compress blocks on disk
- Need to have vocabulary in memory

CS 510 Winter 2007

6

## Merge small indexes

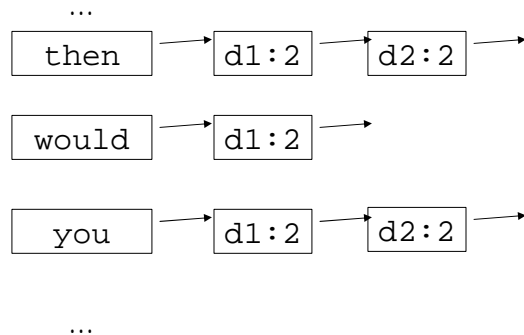
Build inverted indexes directly in memory

- Each term has an extensible list for entries
- Need a directory structure to get from each term to its list (hash table, balanced tree)
- Parse one document at a time, add entries to lists for terms in the document

CS 510 Winter 2007

7

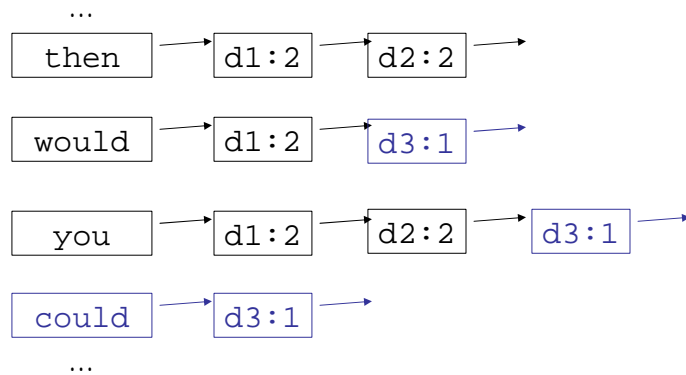
## Example index construction



CS 510 Winter 2007

8

## Example index construction (2)



CS 510 Winter 2007

9

## Combine small indexes

- When memory fills, flush vocabulary and index to disk (in order): a “run”
  - Use a space-efficient representation
  - Start over with next unindexed document
- When all documents processed, merge all runs into one big index

This process has sequential access to runs and to final vocabulary and terms lists

CS 510 Winter 2007

10



## Updating an inverted index

Prohibitive to update disk-based index directly for each new document

- 100's of term lists to update, each might be many pages long.
- Very poor locality of access

CS 510 Winter 2007

11



## Update alternatives

### 1. Rebuild

- Works if document set small or index can be out of date
- Might need old and new index at same time for availability

### 2. Multiple indexes

- New documents indexed in memory, less recent index on disk
- Periodically merge the two

### 3. Incremental update

- Same as previous, but only do merge term-by-term
- (For example, when reading a term list)

CS 510 Winter 2007

12



## How do you throw hardware at the problem?

---

Need to partition index, distribute work

Two main choices

*Term partitioning* (TP): one node indexes all documents for a subset of terms

*Document partitioning* (DP): one node indexes a subset of documents for all terms



## Tradeoffs

---

- TP – every node needs access to all documents
- DP – every node needs full vocabulary
- Communications
  - TP moves term lists or partial  $S_{q,d}$ 's
  - DP moves ordered slits of doc-ids and  $S_{q,d}$ 's
- Balancing – probably easier in TP: don't have to worry about "hot" search terms

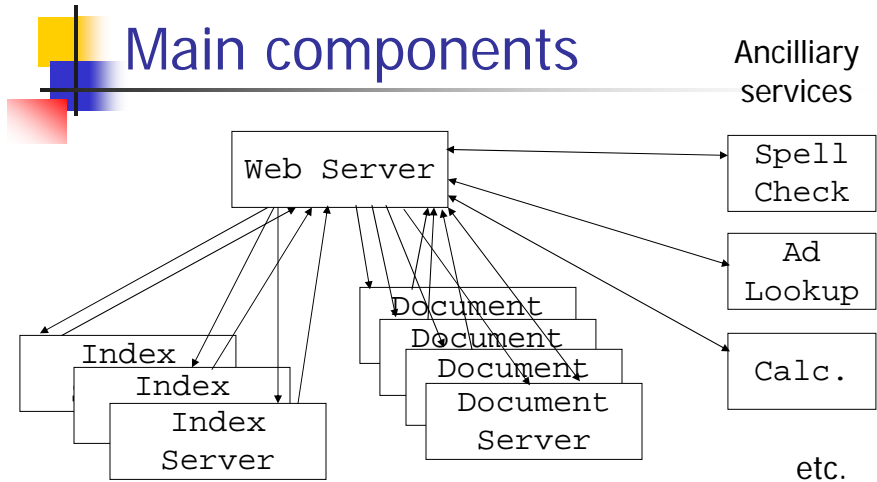
# What does Google do?

## Document partitioning

See: Barroso, Dean, Hoeltzle. "Web search for a planet: The Google cluster architecture." *IEEE Micro* March-April 2003

Goal: High throughput per \$

- Use commodity PCs (not the latest generation)
- Clusters of 1000s of nodes; software-based reliability





## Responsibilities

---

Web server: Receives user query, coordinates it, builds HTML result

Index is divided into *shards*, each with a random subset of documents

- Shard is supported by a *pool* of machines
- To answer a query, need one machine in the pool of each shard
- Results get merged into ordered list of doc-ids

CS 510 Winter 2007

17



## Responsibilities (2)

---

Doc servers also broken into shards, each supported by a pool

Retrieve snippets of documents for keyword in context part of result

Index is compressed. Index servers spend most of their cycles decompressing data.

CS 510 Winter 2007

18



## Compressing index structures

Doc-ids in a term list form an increasing sequence of integers

As do term positions in one document

Have talked about representing sequences of increasing integers as gaps

- Gives more smaller #'s than larger #'s
- Same is true of frequencies

CS 510 Winter 2007

19



## Variable-length integer coding

Want shorter codes for smaller integers

Need to know where each codeword ends

Have seen unary

$i$  is  $1^{i-1}0$

3 is 110

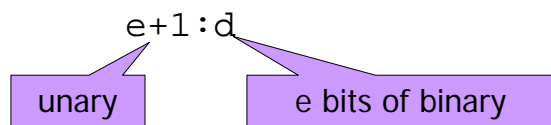
CS 510 Winter 2007

20

## Gamma code

Decompose  $I$  as  $2^e + d$  (where  $e$  as large as possible)

Write as



19 is  $16 + 3 = 2^4 + 3$

111100011

CS 510 Winter 2007

21

## Exercise

- What is the code for 74?  
2?
- What does 111111101001001 encode?
- Is every sequence of bits a gamma code for some integer?

CS 510 Winter 2007

22



## Delta code

---

Again, consider  $i$  as  $2^e + d$ , but write  $e+1$  in gamma code

For example

$$137 = 128 + 9 = 2^7 + 9$$

$e+1$ : 0001001

$$7 = 2^2 + 3$$

full code is 110 11 0001001



## Which code to use?

---

Other codes are given

Which one to use (and with which parameters) depends on distribution of values you want represent



## Byte-aligned codes

Might be expensive to manipulate individual bits; want codeword to be an even number of bytes

Use first bit of each byte to indicate if there are more bytes coming (0=no, 1=yes). Other bits are binary rep.

10110 is 00010110

1001101 1110100 1101 is

11001101 11110100 00001101

CS 510 Winter 2007

25



## Entries in a term list

Coding a list of  $\langle d, f_{d,t} \rangle$  entries

The  $d$  values are increasing, so compute "d-gaps", store as delta or gamma

Represent in  $f_{d,t}$  (fixed) binary or gamma

Note that as doc-id grows, we expect the gaps to stay small

$\langle 3, 6 \rangle \langle 18, 1 \rangle \langle 20, 3 \rangle \langle 41, 2 \rangle \dots$

$\langle 2017, 1 \rangle \langle 2019, 3 \rangle \langle 20125, 2 \rangle \dots$

CS 510 Winter 2007

26