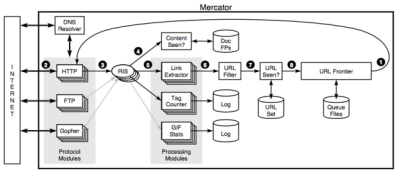
Lecture 13
Web Crawling, Index Building, Search Architecture



Organization

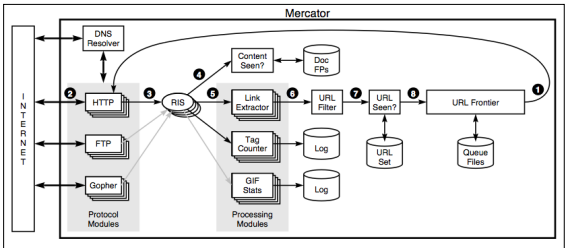
- Today's class contains many search topics we have not yet explored
  - Crawler design, deduplication
  - Inverted-index construction
  - Distributed search architecture
- It's a bit of a grab-bag, but these are still-serious challenges to building a good search engine
- Exams available at end of class

2

Crawler Design

- Mercator was the AltaVista crawler (1998)
- Exceptionally well-documented, even 12 years later

3



- 1. Remove URL from queue
- 2. Network protocols
- 3. Read w/ RewindInputStream (RIS)
- 4. Has document been seen before?
- 5. Extract links
- 6. D'load new URL?
- 7. Has URL been seen before?
- 8. Add URL to frontier

4

Deduplication

- How can you be sure a Web page is worth indexing?
  - Has it changed meaningfully?
  - A clone of another site? (Weirdly common)
- How can you generate a fingerprint of a page?
- What about a near-fingerprint?
- How can we avoid comparing all pairs of Web pages?
  - $O(N^2)$, where N is very, very big

5

Shingling

- Compute the k-shingles for a page
  - If A, B share many k-shingles, they're dups
- Jaccard coefficient of $S(d_1)$ and $S(d_2)$ determines overlap

$$\frac{S(d_1) \cap S(d_2)}{S(d_1) \cup S(d_2)}$$

- But this still requires pairwise comparisons

6

Shingling

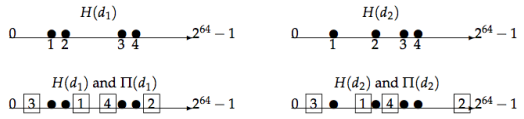
- Let H(d_i) be the hash of single I

Document 1 Document 2

7

Shingling

- Let H(d_i) be the hash of single I

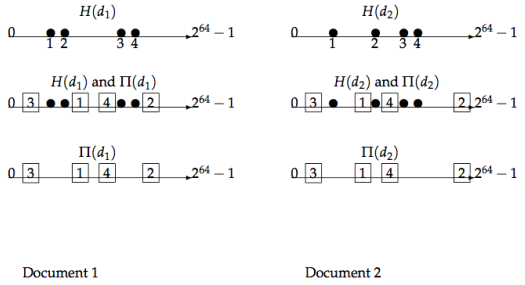


Document 1 Document 2

8

Shingling

- Let H(d_i) be the hash of single I

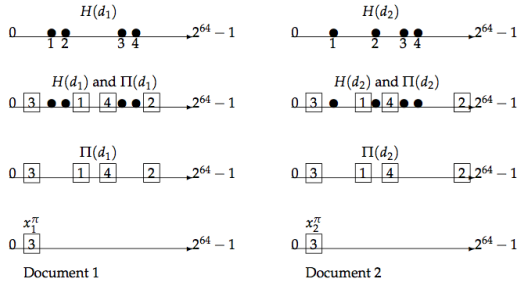


Document 1 Document 2

9

Shingling

- Let H(d_i) be the hash of single I



Document 1 Document 2

10

Shingling

- Why bother with all this?

$$Jaccard(S(d_1),S(d_2)) = P(x_1^{\Pi} = x_2^{\Pi})$$

- By generating repeated random permutations, we perform many tests of whether $x_1^{\Pi} = x_2^{\Pi}$
- Put another way, generate 200 Π to make 200 x_i^{Π} ; call that set the *sketch*(d_i)

$$Jaccard(S(d_i),S(d_j)) = \frac{|\,sketch(S(d_i)) \cap sketch(S(d_j))\,|}{200}$$

11

OK, but get to the point

- We want to compute, for all i, j:

$$\frac{|\,sketch(S(d_i)) \cap sketch(S(d_j))\,|}{200}$$

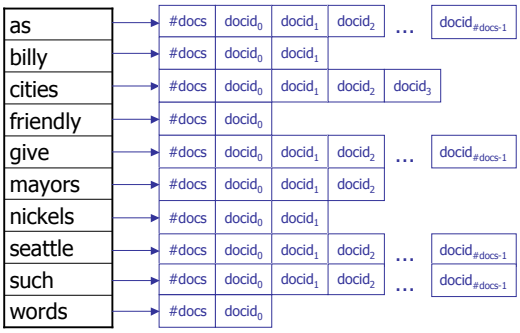
- This is still a quadratic operation!
- Goal: create clusters of similar docs
  - Start with <x_i, d_j>
  - For each x_i, we can find all relevant d, and thus all pairs d_i, d_j
  - This gives you all doc-pairs with non-zero overlap; you can then evaluate as before
  - To have a stronger test, don't evaluate until pair has many x_i in common
- This procedure is heuristic, but v. powerful

12

Inverted Indexes, Revisited

- Remember the inverted index?

13



Inverted Indexes, Revisited

- Remember the inverted index?
  - How can we build it efficiently?
- Remember:
  - Disk seeks are very expensive (5ms)
  - Continuous disk reads or writes are OK (50-100MB/sec)
  - Machines can have a lot of memory (often up to 24GB), but disk is always much cheaper
  - Input is the tokenized document set

15

Basic Tasks

1. Compile term-termid mapping
   - First, compile vocabulary
   - Second, compile index (single-pass also possible)
2. Assemble all termid-docid pairs
3. Sort first by termid, then docid
4. Write out in inverted-index form

- **EASY!**
- Well, not if docs won't fit into memory

16

Block sort-based indexing

- *External sort algorithms* work on sets larger than memory
- Block-Sort-Based Index Algorithm:

```
n = 0
While docsRemain
    n++
    block = ParseNextBlock()
    BSBI-Invert(block)
    WriteToDisk(block, fn)
MergeBlocks(f1, …, fn) => fmerged
```

17

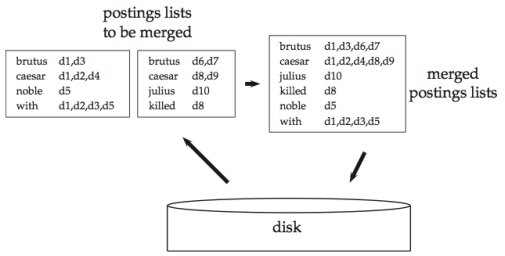
BSBI, cont'd.

- ParseNextBlock accumulates termid-blockid pairs in memory until block size
- BSBI-Invert generates small in-memory inverted index
- So: we build a series of small in-memory inverted indexes, writing each one to disk
- Finally: we merge them

18

Block merging

- Merging two lists:



19

Distributed Indexing

- Document sizes huge; must divide work over many machines in cluster
- Phase 1: Parsing
  - Break input pages into n splits
  - Machines in cluster process a split at a time (A split consists of documents)
  - Each breaks into (termID,docID) pairs
  - Write to local segment files: a-f,g-p,q-z

20

Distributed Indexing

- Phase 2: Inverting
  - Break keyspace into partitions; assign a machine to each partition, *E.g.*, a-f, or g-p
  - Each inverter machine collects the segment files that are useful for its partition
  - Each inverter then combines the segment files for each termID within its keyspace, and outputs a new segment file
- How many segment files do you get?
- How do you choose the number of inverters? Of parsers?

21

Distributed Indexing

- Recap:
  - A Parser is assigned a region of input
  - Parsers break docs into (termID, docID)
  - Parsers write pairs into segment files
  - An Inverter is assigned region of keyspace
  - Inverters collect segment files appropriate for its keyspace
  - Inverters combine segment info for each termID, then write out index for that termID
- An instance of MapReduce; more later

22

23

Distributed Indexing

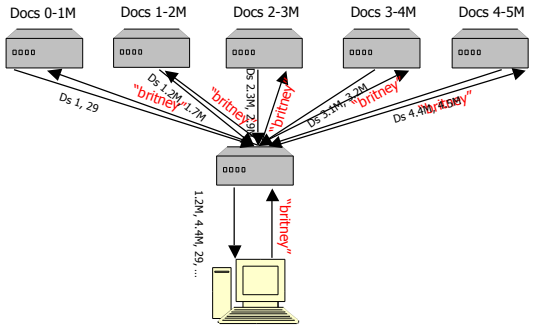
24

Distributed Searching

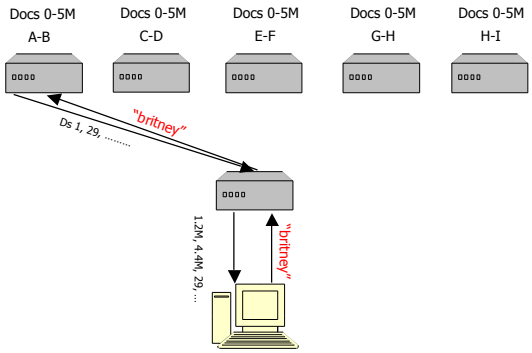
- Not even the inverted index can handle billions of docs and hundreds of millions of queries on a single machine
- Also, what if machine fails?
- Need to parallelize query
  - Segment by document
  - Segment by search term

25

Scaling - doc segmenting



Scaling - term segmenting



Segmentation

- Segment by document
  - Easy to partition (just MOD the docid)
  - Easy to add new documents
  - If machine fails, quality goes down but queries don't die
- Segment by term
  - Harder to partition (terms uneven)
  - Trickier to add a new document (need to touch many machines)
  - If machine fails, search term might disappear, but not critical pages (e.g., yahoo.com/index.html)

28

Finale

- Web Search has many moving parts
  - Crawling & Deduplication
  - Text Analysis & Indexing
  - Ranking & Query Processing
- Four lectures, and only scratched the surface
- After the break, some search-related issues:
  - Ad auctions
  - Recommendation systems
  - Search-related research topics

29