Indexing

Debapriyo Majumdar
Information Retrieval
Indian Statistical Institute Kolkata

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.5 µs)
Processor's clock rate	10^{9} per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

Caching

- Accessing data in memory is a few times faster than from disk
- Keep as much information as possible in main memory

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.5 µs)
Processor's clock rate	10^{9} per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

Reading from disk

- About 10 MB data in one contiguous chunk on disk
  - One seek (~5 ms) + Read (~ 0.2 s) ≈ 0.2 s
- About 10 MB data in 1000 non- contiguous chunks on disk
  - Thousand seeks (~5 s) + Read (~ 0.2 s) ≈ 5.2 s

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 µs)
Processor's clock rate	10^{9} per second
Low-level operation (compare, swap a word)	10^{-8} s (1 µs)

- The OS reads and writes data in blocks
  - Block sizes may be 8, 16, 32 or 64KB
  - Reading 1 byte data takes same time as the entire block
  - Buffer: part of main memory where a block is stored after reading or while writing

Hardware parameters

Parameter	HDD
Average seek time	5×10^{-3} s (5 ms)
Average time to read a byte from disk	2×10^{-8} s (50MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 μs)
Processor's clock rate	10^{9} per second
Low-level operation (compare, swap a word)	10^{-8} s (1 μs)

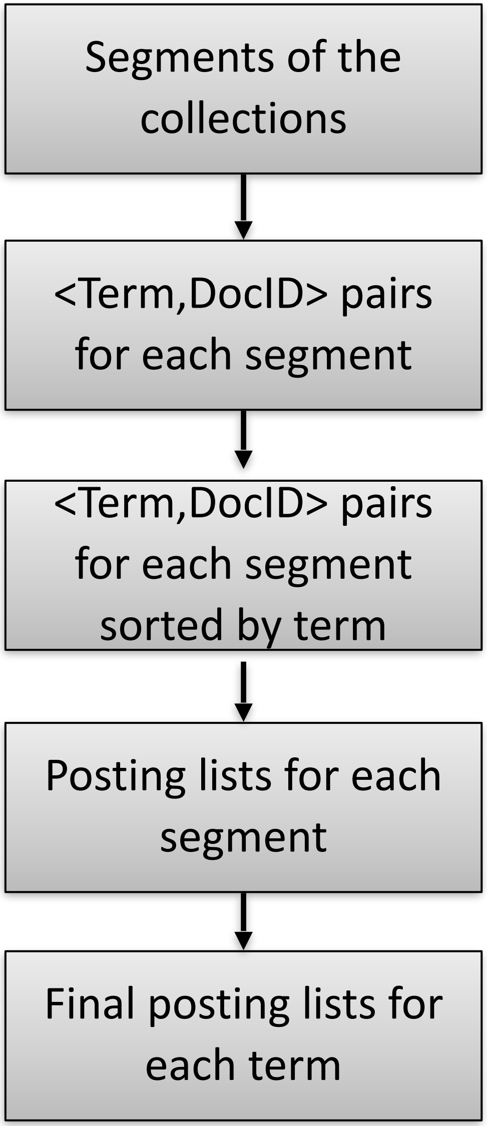
- Simultaneous reading and processing
  - Disk → main memory data transfer is done by system bus
  - Processor is free, can work while reading data
  - The system can read data and decompress at the same time

Hardware parameters

Parameter	HDD	SSD
Average seek time	5×10^{-3} s (5 ms)	0.1 ms
Average time to read a byte from disk	2×10^{-8} s (50MB/s)	5×10^{-9} s (200 MB/s)
Average time to access a byte in memory	5×10^{-9} s (0.2 μs)	
Processor's clock rate	10^{9} per second	
Low-level operation (compare, swap a word)	10^{-8} s (1 μs)	

- The parameters change a lot for SSDs
  - Seek: about 50 times faster than HDD
  - Read: about 4 times faster than HDD

Blocked sort based indexing



Segments of the collections

<Term,DocID> pairs for each segment

<Term,DocID> pairs for each segment sorted by term

Posting lists for each segment

Final posting lists for each term

Step 1: partition the collection into segments

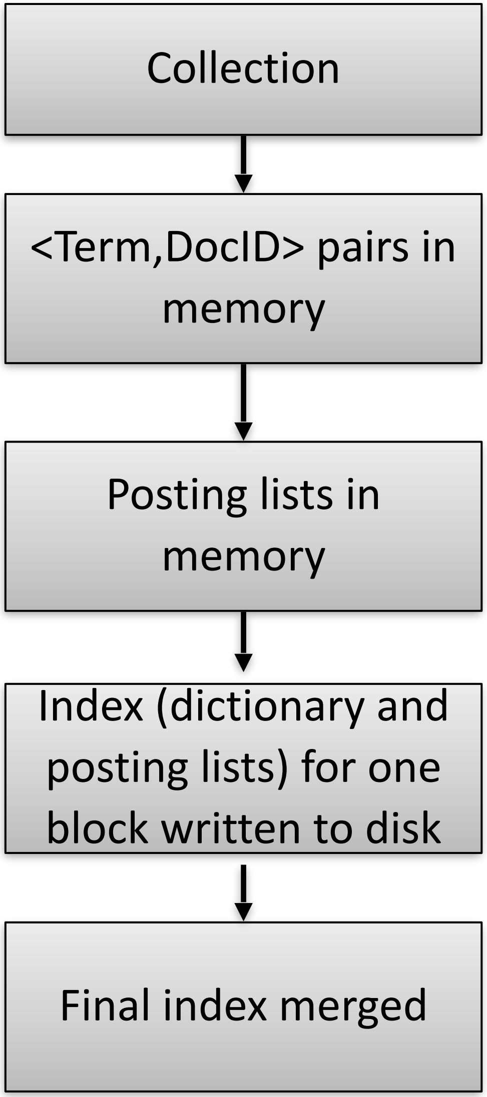
Step 2: make a pass over all documents, write out <term, docId> pairs

Step 3: separately in each segment, sort the <term,docId> pairs by term. May require an external memory sort.

Step 4: separately in each segment, create the posting lists for each term

Step 5: for each term, merge the posting lists from all segments and create one single posting list

Single-pass in memory indexing



Collection

<Term,DocID> pairs in memory

Posting lists in memory

Index (dictionary and posting lists) for one block written to disk

Final index merged

Step 1: Keep parsing documents, convert to <term,docId> pairs

Step 2: invert <term,docId> pairs in memory to make posting lists; immediately available for searching as well

Step 3: When memory is exhausted, write out the whole index to disk after sorting the posting lists and dictionary

Step 4: merge the index files for different blocks

Dynamic indexing

- For fast access, posting lists are written in contiguous blocks
- Changes to existing index requires a lot of moving of the data
- Approach: create an auxiliary index for incremental changes, keep growing that index
  - Searches are performed in both old and auxiliary indices, results are merged
  - When auxiliary index grows significantly large, merge it with the original one (costly operation, but not done often)
- Ease of merging
  - If each posting list can be one file → Good
  - Bad idea! The OS cannot handle too many files (too many terms) well
  - Tradeoff between #of files and #of posting lists per file: keep some posting lists in one file, but not all in one

Security

Search in enterprise data

- Not all users may be allowed to view all documents
- Approach 1: use access control lists
  - For each user / group of users, have a posting list of documents the user / group can access
  - Intersect that list with the search result list
  - Problem: difficult to maintain when access changes
  - Access control lists may be very large
- Approach 2: check at query time
  - From the retrieved results, filter out those which the user is not allowed to access
  - May slow down retrieval

References

- Primarily: IR Book by Manning, Raghavan and Schuetze: http://nlp.stanford.edu/IR-book/