INFORMATION RETRIEVAL

Dr. Reda M. Hussien

Index construction

Index construction

- How do we construct an index?
- What strategies can we use with limited main memory?

Recall index construction

- Documents are parsed to extract words
- words are saved with the Document ID.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Sec. 4.2

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Key step

- the inverted file is sorted by terms.

We focus on this sort step.

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

→

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

RCV1: Our collection for this lecture

- As an example for applying scalable index construction algorithms, we will use the Reuters RCV1 collection.
  - This is one year of Reuters newswire (part of 1995 and 1996)

- The collection isn't really large enough, but it's publicly available and is a plausible example.

A Reuters RCV1 document

REUTERS

You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enoug

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

Email This Article | Print This Article | Reprints

[-] Text [+]

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Reuters RCV1 statistics

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
T	Tokens	100,000,000

4.5 bytes per word token vs. 7.5 bytes per word type: why?

Sort-based index construction

- As we build the index, we parse docs one at a time.
  - The final postings for any term are incomplete until the end.
- At 8 bytes per (termID, docID), demands a lot of space for large collections.
- T = 100,000,000 in the case of RCV1
  - So … we can do this in memory today, but typical collections are much larger. E.g., the New York Times provides an index of >150 years of newswire
- Thus: We need to store intermediate results on disk.

Scaling index construction

- In-memory index construction does not scale
  - Can't stuff entire collection into memory, sort, then write back
- How can we construct an index for very large collections?
- Taking into account hardware constraints. . .
  - Memory, disk, speed, etc.
- Let's review some hardware basics

Hardware basics

- Servers used in IR systems now typically have several GB of main memory, sometimes tens of GB.
- Available disk space is several (2–3) orders of magnitude larger.
- Fault tolerance is very expensive: It's much cheaper to use many regular machines rather than one fault tolerant machine.

Hardware basics

- Access to data in memory is ***much*** faster than access to data on disk.
- Disk seeks: No data is transferred from disk while the disk head is being positioned.
- Therefore: Transferring one large chunk of data from disk to memory is faster than transferring many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.

Sort using disk as "memory"?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?
- No: Sorting T = 100,000,000 records on disk is too slow – too many disk seeks.
- We need an external sorting algorithm.

External memory indexing

BSBI: Blocked sort-based Indexing (Sorting with fewer disk seeks)

- 8-byte records (termID, docID). These are generated as we parse docs
- Must now sort 100M such 8-byte records by termID
- Define a Block ~ 10M such records
  - Can easily fit a couple into memory
  - Will have 10 such blocks to start with
- Basic idea of algorithm:
  - Accumulate postings for each block, sort, write to disk
  - Then merge the blocks into one long sorted order

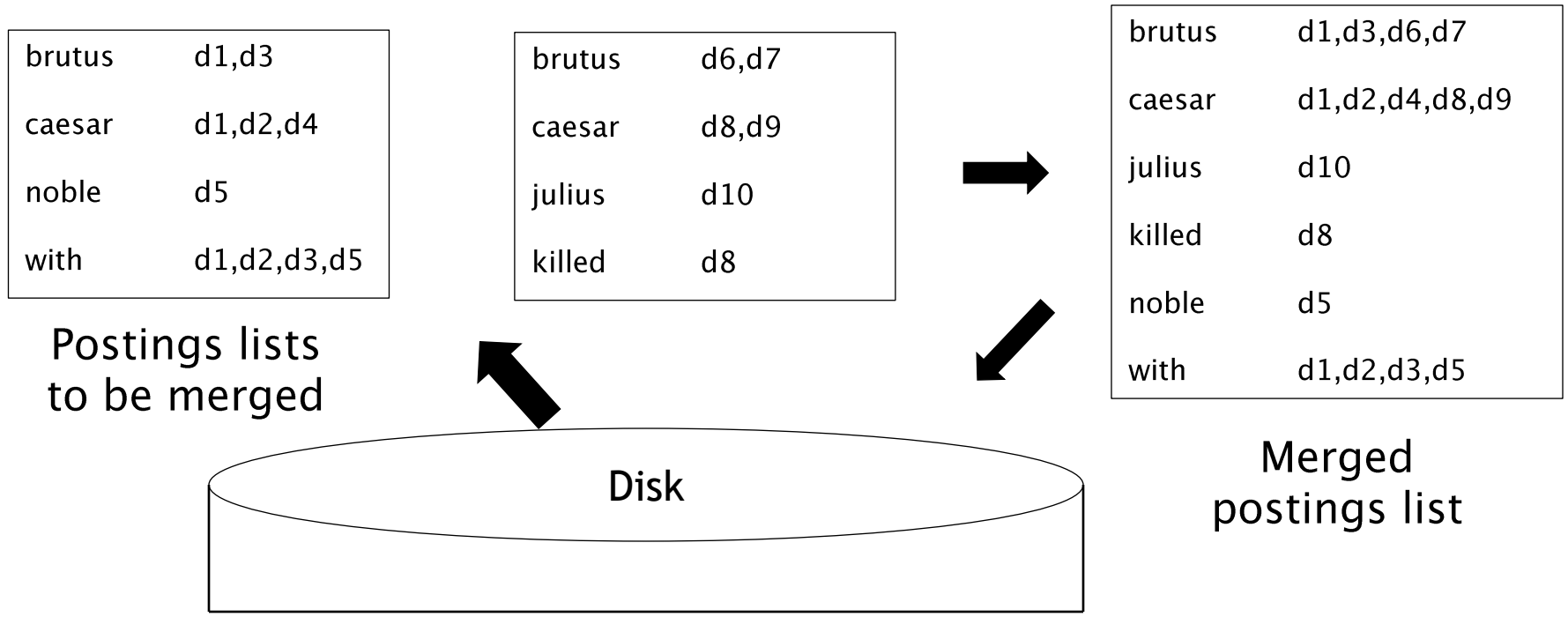
```
BSBIndexConstruction()
 1  n ← 0
 2  while (all documents have not been processed)
 3  do n ← n + 1
 4       block ← ParseNextBlock()
 5       BSBI-Invert(block)
 6       WriteBlockToDisk(block, f_n)
 7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Sorting 10 blocks of 10M records

- First, read each block and sort within:
  - Quicksort takes O(N ln N) expected steps
  - In our case N=10M
- 10 times this estimate – gives us 10 sorted runs of 10M records each.
- Done straightforwardly, need 2 copies of data on disk
  - But can optimize this

How to merge the sorted runs?

- Can do binary merges, with a merge tree of log210 = 4 layers.
- During each layer, read into memory runs in blocks of 10M, merge, write back.



How to merge the sorted runs?

- But it is more efficient to do a multi-way merge, where you are reading from all blocks simultaneously
  - Open all block files simultaneously and maintain a read buffer for each one and a write buffer for the output file
  - In each iteration, pick the lowest termID that hasn't been processed using a priority queue
  - Merge all postings lists for that termID and write it out
- Providing you read decent-sized chunks of each block into memory and then write out a decent-sized output chunk, then you're not killed by disk seeks

Remaining problem with sort-based algorithm

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.