Web Information Retrieval

Lecture 3

Index Construction

Plan

- This time:
  - Index construction

Index construction

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

RCV1: Our collection for this lecture

- Shakespeare's collected works definitely aren't large enough for demonstrating many of the points in this course.
- The collection we'll use isn't really large enough either, but it's publicly available and is at least a more plausible example.
- 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)

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	non-positional postings	100,000,000

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

Recall IIR 1 index construction

- Documents are parsed to extract words and these 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

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

- After all documents have been parsed, the inverted file is sorted by terms.

We focus on this sort step.
We have 100M items to sort.

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

Index construction

- As we build up the index, cannot exploit compression tricks
  - Parse docs one at a time.
  - Final postings for any term – incomplete until the end.
  - (actually you can exploit compression, but this becomes a lot more complex)
- At 10-12 bytes per postings entry, demands several temporary gigabytes
- T = 100,000,000 in the case of RCV1
  - So … we can do this in memory in 2011, but typical collections are much larger. E.g., the *New York Times* provides an index of >150 years of newswire

System parameters for design

- Disk seek ~ 10 milliseconds
- Block transfer from disk ~ 1 microsecond per byte (*following a seek*)
- All other ops ~ 10 microseconds
  - E.g., compare two postings entries and decide their merge order

Bottleneck

- Parse and build postings entries one doc at a time
- Now sort postings entries by term (then by doc within each term)
- Doing this with random disk seeks would be too slow – must sort T=100M records

If every comparison took 2 disk seeks, and T items could be sorted with $T \log_2 T$ comparisons, how long would this take?

Sorting with fewer disk seeks

- 12-byte (4+4+4) records *(term, doc, freq).*
- These are generated as we parse docs.
- Must now sort 100M such 12-byte records by *term*.
- Define a Block ~ 10M such records
  - can “easily” fit a couple into memory.
  - Will have 10 such blocks to start with.
- Will sort within blocks first, then merge the blocks into one long sorted order.

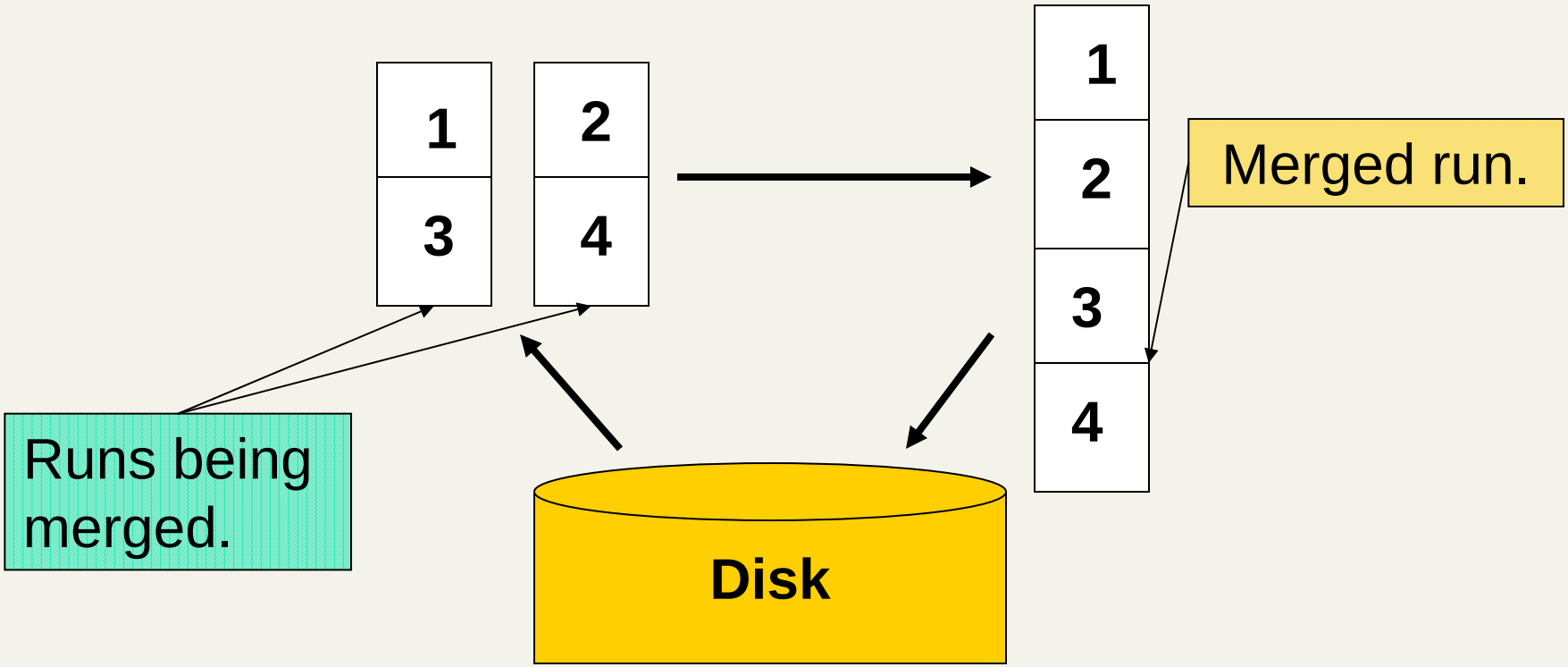
Sorting 10 blocks of 10M records

- First, read each block and sort within:
  - Quicksort takes $2n \ln n$ expected steps
  - In our case 2 x (10M ln 10M) steps
- *Exercise: estimate total time to read each block from disk and quicksort it.*
- 10 times this estimate - gives us 10 sorted *runs* of 10M records each.
- Need 2 copies of data on disk, throughout.

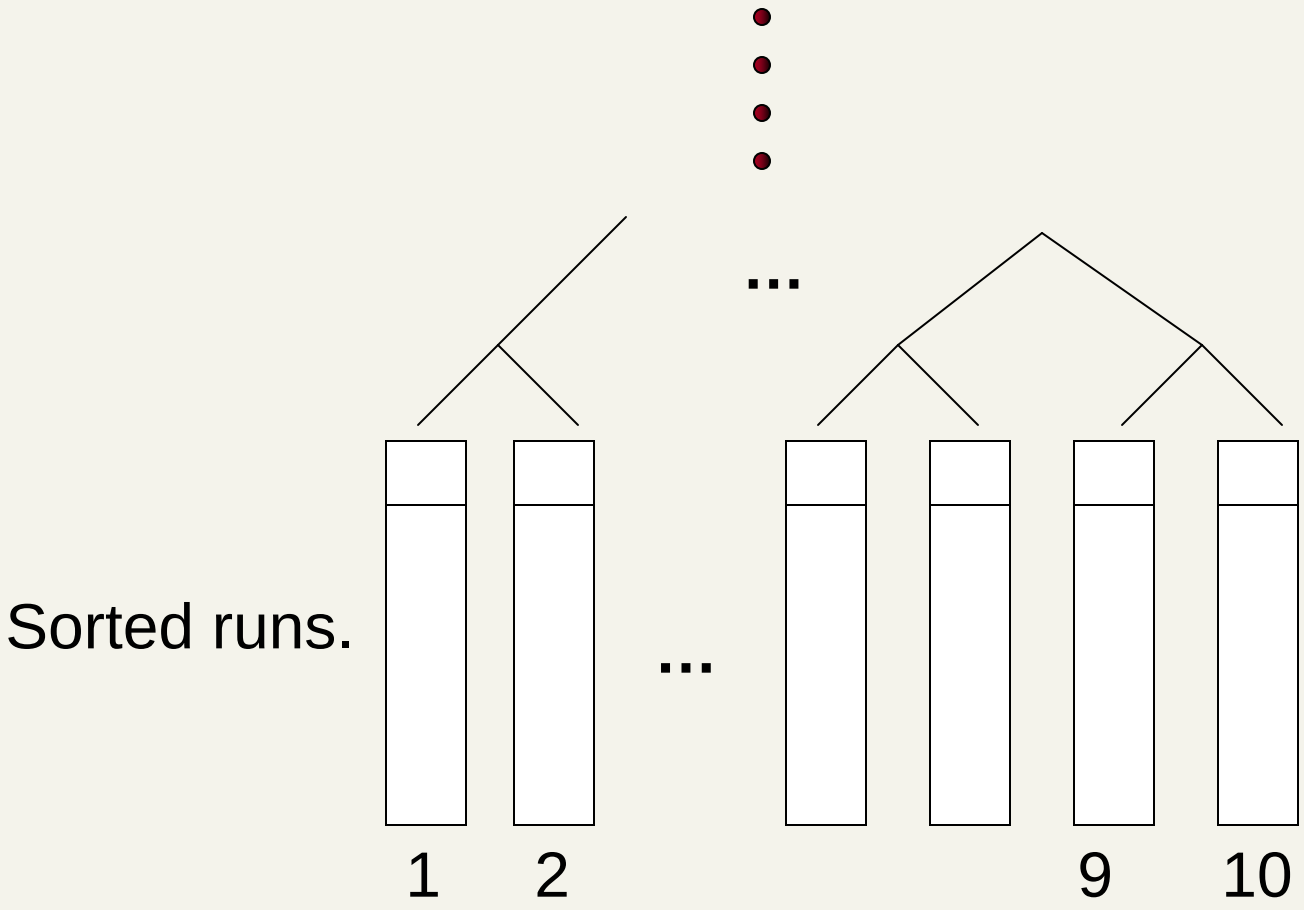
```
BSBIndexConstruction()
 1  n ← 0
 2  while (all documents have not been processed)
 3  do n ← n + 1
 4     block ← ParseNextBlock()
 5     BSBI-Invert(block)
 6     WriteBlockToDisk(block, f_n)
 7  MergeBlocks(f_1, ..., f_n; f_merged)
```

Merging 10 sorted runs

- Merge tree of $\log_2 10 = 4$ layers.
- During each layer, read into memory runs in blocks of 10M, merge, write back.



Merge tree



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
- 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

Distributed indexing

- For web-scale indexing (don’t try this at home!):
  must use a distributed computing cluster
- Individual machines are fault-prone
  - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?

Web search engine data centers

- Web search data centers (Google, Bing, Baidu) mainly contain commodity machines.
- Data centers are distributed around the world.
- Estimate: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

Web search engine data centers

- Web search data centers (Google, Bing, Baidu) mainly contain commodity machines.
- Data centers are distributed around the world.
- Estimate: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

- Use of **MapReduce**
  - An architecture for distributed computing
  - We will cover it in the labs

Resources

- IIR Chapters 4.1, 4.2