Index Construction

Introduction to Information Retrieval
INF 141
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org

Reuters collection example (approximate #'s)

- 800,000 documents from the Reuters news feed
- 200 terms per document
- 400,000 unique terms
- number of postings 100,000,000

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 collection example (approximate #'s)

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time

BSBI

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time
  - Sort posting entries by term

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time
  - Sort posting entries by term
    - Then by document in each term

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time
  - Sort posting entries by term
    - Then by document in each term
  - Doing this with random disk seeks is too slow

BSBI

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time
  - Sort posting entries by term
    - Then by document in each term
  - Doing this with random disk seeks is too slow
  - e.g. If every comparison takes 2 disk seeks and N items need to be sorted with N log2(N) comparisons?

Reuters collection example (approximate #'s)

- Sorting 100,000,000 records on disk is too slow because of disk seek time.
  - Parse and build posting entries one at a time
  - Sort posting entries by term
    - Then by document in each term
  - Doing this with random disk seeks is too slow
  - e.g. If every comparison takes 2 disk seeks and N items need to be sorted with N log2(N) comparisons?
    - 306ish days?

Reuters collection example (approximate #'s)

Reuters collection example (approximate #'s)

- 100,000,000 records

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds
- = 442,923.75 minutes

BSBI

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds
- = 442,923.75 minutes
- = 7,382.06 hours

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds
- = 442,923.75 minutes
- = 7,382.06 hours
- = 307.59 days

Reuters collection example (approximate #'s)

- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds
- = 442,923.75 minutes
- = 7,382.06 hours
- = 307.59 days
- = 84% of a year

Reuters collection example (approximate #'s)

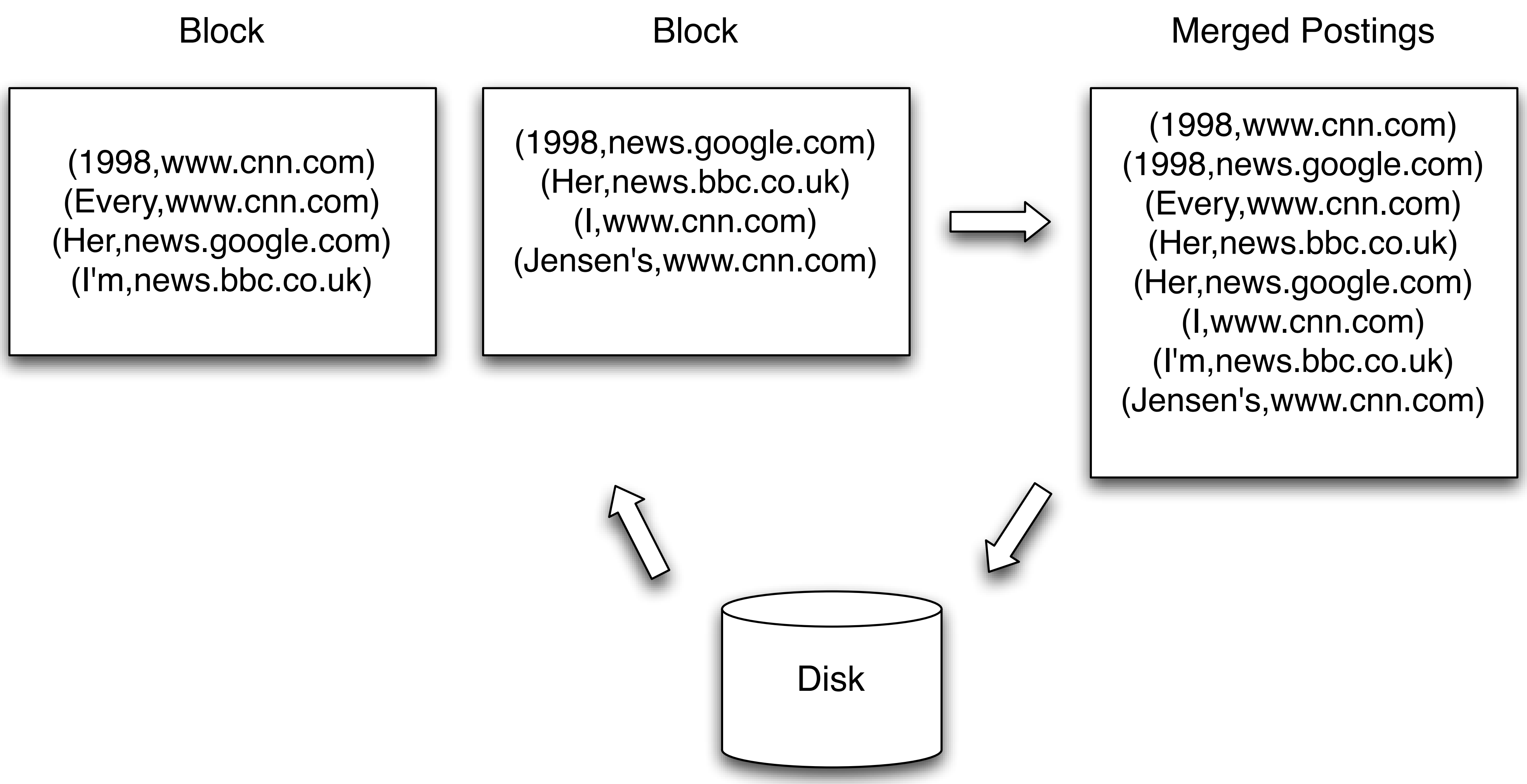
- 100,000,000 records
- Nlog2(N) is = 2,657,542,475.91 comparisons
- 2 disk seeks per comparison = 13,287,712.38 seconds x 2
- = 26,575,424.76 seconds
- = 442,923.75 minutes
- = 7,382.06 hours
- = 307.59 days
- = 84% of a year
- = 1% of your life

Different way to sort index

- 12-byte records (term, doc, meta-data)
- Need to sort T= 100,000,000 such 12-byte records by term
- Define a block to have 1,600,000 such records
  - can easily fit a couple blocks in memory
  - we will be working with 64 such blocks
- Accumulate postings for each block (real blocks are bigger)
- Sort each block
- Write to disk
- Then merge

BSBI - Block sort-based indexing

Different way to sort index



BSBI - Block sort-based indexing

BlockSortBasedIndexConstruction

```
BlockSortBasedIndexConstruction()
n ← 0
while (all documents not processed)
    do block ← ParseNextBlock()
       BSBI-Invert(block)
       WriteBlockToDisk(block, f_n)
MergeBlocks(f_1, f_2..., f_n, f_merged)
```

Block merge indexing

- Parse documents into (TermID, DocID) pairs until "block" is full
- Invert the block
  - Sort the (TermID,DocID) pairs
- Write the block to disk
- Then merge all blocks into one large postings file
  - Need 2 copies of the data on disk (input then output)

Analysis of BSBI

- The dominant term is O(NlogN)
  - N is the number of TermID,DocID pairs
- But in practice ParseNextBlock takes the most time
- Then MergingBlocks
- Again, disk seeks times versus memory access times

Analysis of BSBI

- 12-byte records (term, doc, meta-data)
- Need to sort T= 100,000,000 such 12-byte records by term
- Define a block to have 1,600,000 such records
  - can easily fit a couple blocks in memory
  - we will be working with 64 such blocks
- 64 blocks * 1,600,000 records * 12 bytes = 1,228,800,000 bytes
- Nlog2N comparisons is 5,584,577,250.93
- 2 touches per comparison at memory speeds (10e-6 sec) =
  - 55,845.77 seconds = 930.76 min = 15.5 hours

Overview

- Introduction
- Hardware
- BSBI - Block sort-based indexing
- SPIMI - Single Pass in-memory indexing
- Distributed indexing
- Dynamic indexing
- Miscellaneous topics

SPIMI

- BSBI is good but,
  - it needs a data structure for mapping terms to termIDs
  - this won't fit in memory for big corpora
  - A lot of redundancy in (T,D) pairs
- Straightforward solution
  - dynamically create dictionaries (intermediate postings)
  - store the dictionaries with the blocks
  - integrate sorting and merging

Single-Pass In-Memory Indexing

This is just data structure management

```
SPIMI-INVERT(tokenStream)
outputFile ← NEWFILE()
dictionary ← NEWHASH()
while (free memory available)
    do token ← next(tokenStream)
        if term(token) ∉ dictionary
          then postingsList ← ADDTODICTIONARY(dictionary, term(token))
          else postingsList ← GETPOSTINGSLIST(dictionary, term(token))
        if full(postingsList)
          then postingsList ← DOUBLEPOSTINGSLIST(dictionary, term(token))
        ADDTOPOSTINGSLIST(postingsList, docID(token))
sortedTerms ← SORTTERMS(dictionary)
WRITEBLOCKTODISK(sortedTerms, dictionary, outputFile)
return outputFile
```

14. Final step is merging

Single-Pass In-Memory Indexing

- So what is different here?
  - SPIMI adds postings directly to a posting list.
    - BSBI first collected (TermID,DocID pairs)
      - then sorted them
      - then aggregated the postings
  - Each posting list is dynamic so there is no term sorting
  - Saves memory because a term is only stored once
  - Complexity is O(T) (sort of, see book)
  - Compression (aka posting list representation) enables each block to hold more data

Large Scale Indexing

- Key decision in block merge indexing is block size
- In practice, crawling often interlaced with indexing
- Crawling bottlenecked by WAN speed and other factors

Overview

- Introduction
- Hardware
- BSBI - Block sort-based indexing
- SPIMI - Single Pass in-memory indexing
- Distributed indexing
- Dynamic indexing
- Miscellaneous topics

Distributed Indexing

- Web-scale indexing
  - Must use a distributed computing cluster
  - "Cloud computing"
- Individual machines are fault-prone
  - They slow down unpredictably or fail
    - Automatic maintenance
    - Software bugs
    - Transient network conditions
    - A truck crashing into the pole outside
    - Hardware fatigue and then failure

Distributed Indexing - Architecture

- The design of Google’s indexing as of 2004

Distributed Indexing - Architecture

- Use two classes of parallel tasks
  - Parsing
  - Inverting
- Corpus is split broken into splits
  - Each split is a subset of documents
  - analogous to distributed crawling
- Master assigns a split to an idle machine
  - Parser will read a document and sort (t,d) pairs
  - Inverter will merge, create and write postings

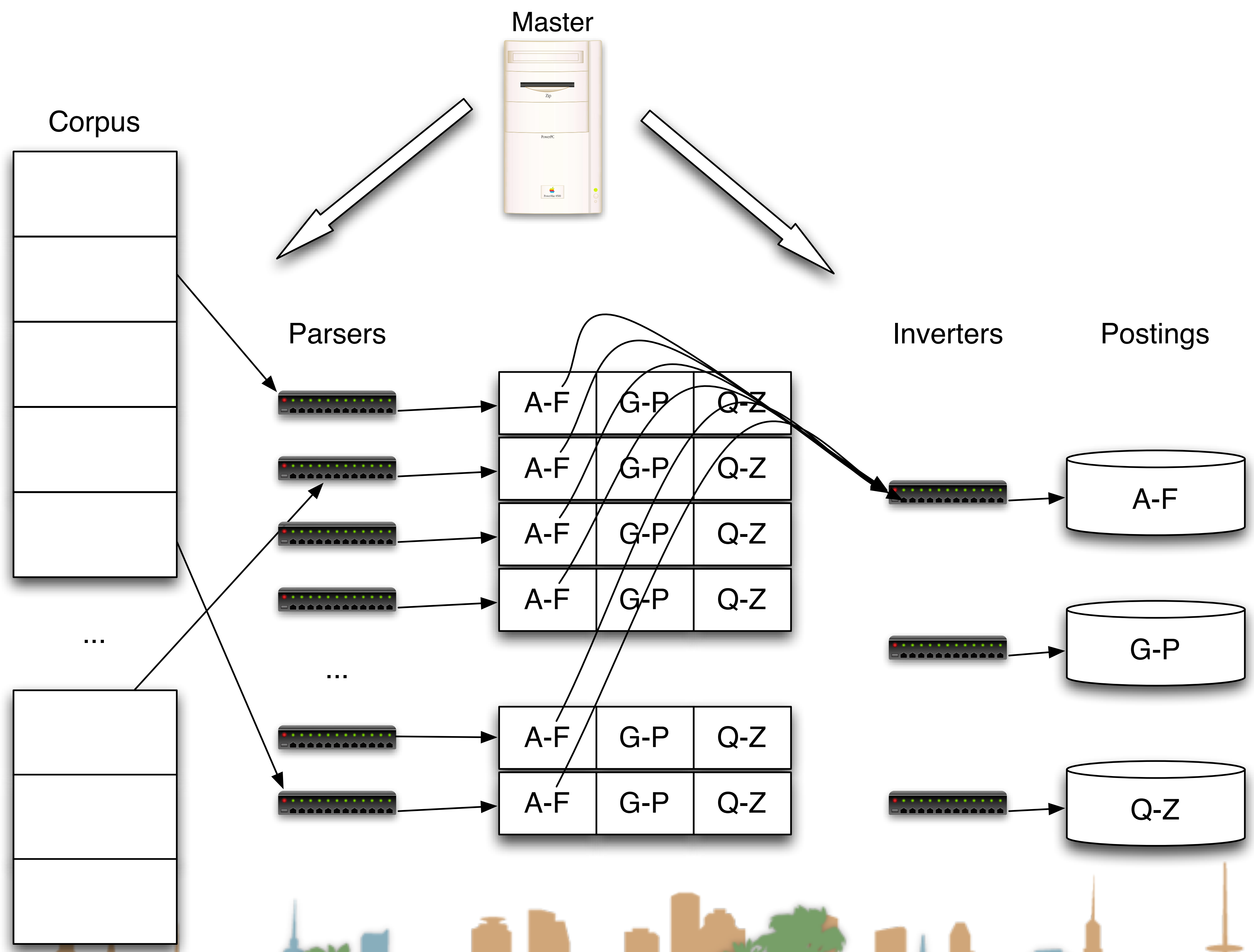
Distributed Indexing - Architecture

- Use an instance of MapReduce
  - An general architecture for distributed computing
  - Manages interactions among clusters of
    - cheap commodity compute servers
    - aka nodes
  - Uses Key-Value pairs as primary object of computation
  - An open-source implementation is “Hadoop” by apache.org

Distributed Indexing - Architecture

- Use an instance of MapReduce
  - There is a map phase
    - This takes splits and makes key-value pairs
    - this is the “parse/invert” phase of BSBI and SPIMI
  - The map phase writes intermediate files
    - Results are bucketed into buckets indexed by key
  - There is a reduce phase
    - This is the “merge” phase of BSBI and SPIMI
    - There is one inverters for each bucket

Distributed Indexing - Architecture



LUCI