Week 4: Processing Big Data

Jason S. CHang

Hadoop and MapReduce - A Preview

Typical Big Data Tasks

- Word count
- Inverted list (search engine index)

Handling Big Data is Not New

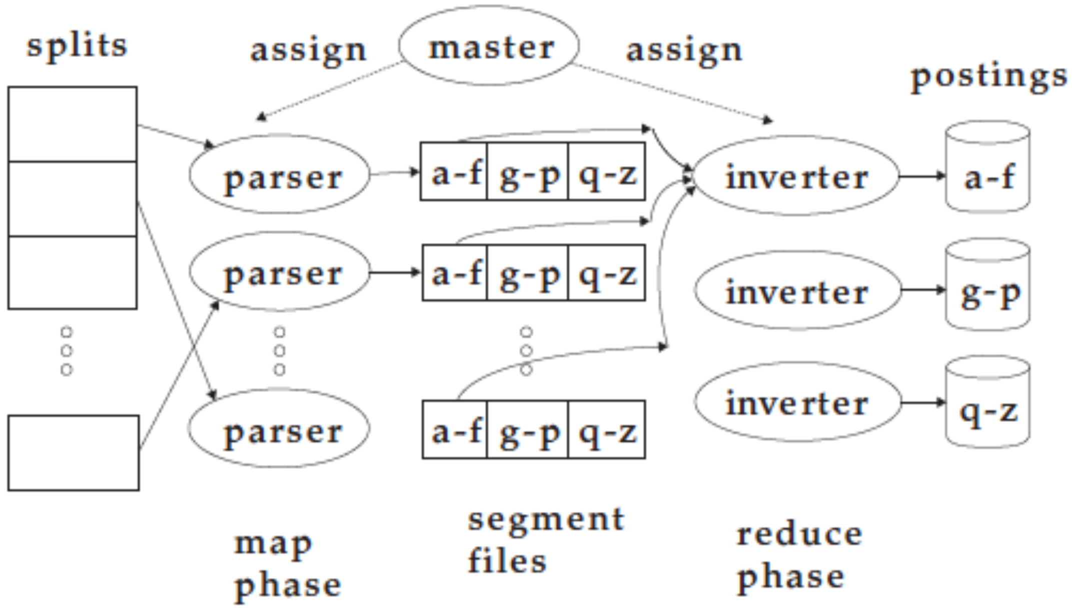
- Information retrieval systems had to deal with gigabyte datasets
  - See Managing Gigabyte: Compressing and Indexing Documents and Images
  - Textbook website: `http://ww2.cs.mu.oz.au/~alistair/mg/`
- Main ideas
  - Split the dataset into managable chunks into hard disks (Data Partition)
  - Process each chunks (in memory)
    - Sort by word (key), document no. (value)
    - Merge document no. into an inverted list
  - Merge processed chunks and output the results to hard disks (memoryless)
  - Repeat the merge process if necessary
  - In the process, compress and decompress

Traditional vs. Modern Search Engines

- Traditional SE uses Blocked Sort-based Indexing
  - Parse document to (word, doc#) pairs (so-called posting)
  - Non-parallel
    * Split pairs (sequencially) into chunks
    * Sort and save chunks to disk files
  - Merge chunks
- Modern SE uses Distributed Indexing (MapReduce)
  - Map document to (word, doc#) pairs
  - Framework of PC cluster (Hadoop)
    * Shuffle (partition/hashing and distribute)
    * Sort
  - Reduce each key group by merging postings (doc#)
- `http://nlp.stanford.edu/IR-book/pdf/04const.pdf`

Inverted List - MapReduce

- Like unix pipe in parallel

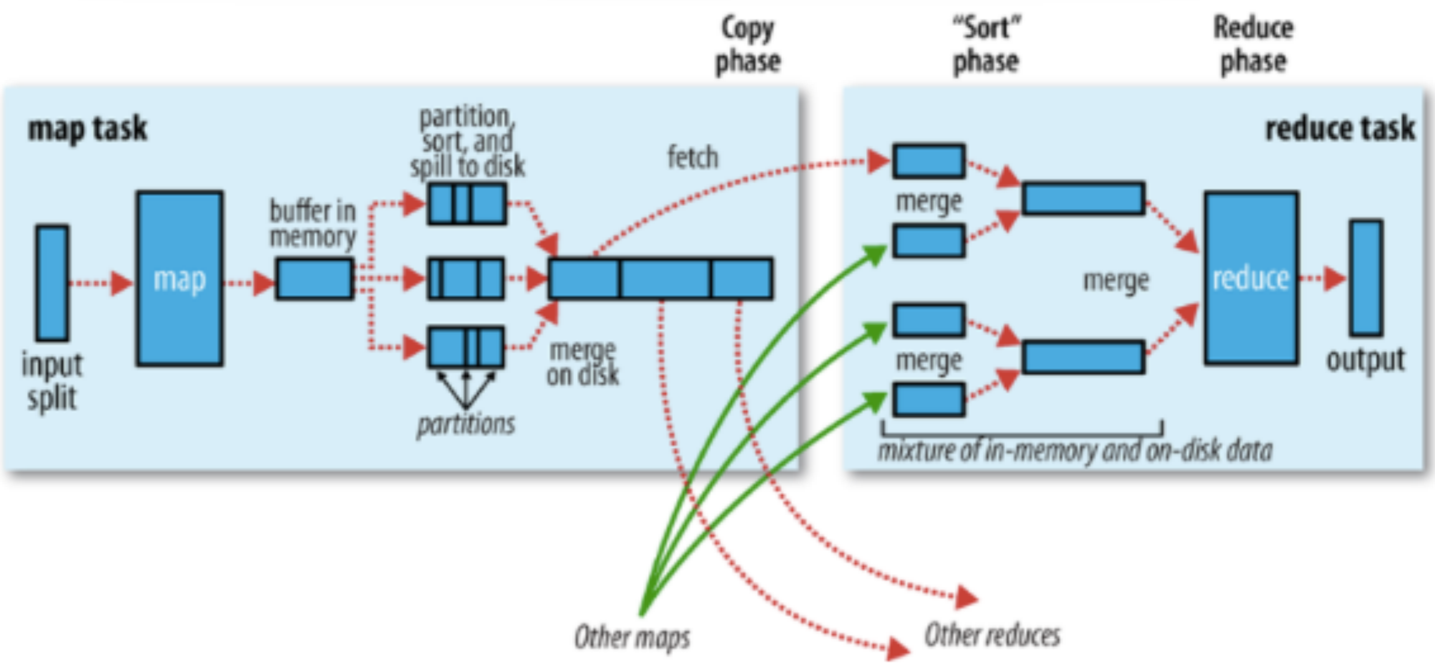


http://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf

(page_76)

Shuffling and Sorting Data

- Suffle and sort in MapReduce



`https://www.inkling.com/read/hadoop-definitive-guide-tom-white-3rd/chapter-6/shuffle-and-sort`

Word Count - Unix-Unix Simulation

- mapper = tr, reducer = uniq -c
- one maper, one reducer

```
$ zcat citeseerx.10.gz | head -100 |
  tr -sc 'A-Za-z' '\012' | sort | uniq -c
  sort -nr | head -7

   156 the
   115 of
    85 and
    65 a
    56 to
    44 in
    37 is
```

- http://web.stanford.edu/class/cs124/kwc-unix-for-poets.pdf

Word Count - Python-Unix Simulation

- Many files in dataset
  - Dataset: sents.parts-00.gz, sents.part-01.gz, sents.part-02.gz
- Mappers (e.g., 4) output is hashed to many subdir's for reducers (e.g., 8)
  - temp/reducer-1/: mapper-1, mapper-2, mapper-3, mapper-4 (spill files)
  - temp/reducer-2/: mapper-1, mapper-2, mapper-3, mapper-4
  - ...
  - temp/reducer-8/: mapper-1, mapper-2, mapper-3, mapper-4
- System combines and sort mapper files and feeds to each reducer
- Each reducer output data to the results dir
- results/part-1, part-2, ..., part-8
- Mapper and reducer: **wc.map.py**, **wc.reduce.py**
- MapReduce system: **lmr**
  - Open source code written in shell commands (including **parallel, split, hash, sort, mkdir**)
  - Available at `https://github.com/d2207197/local-mapreduce`

lmr: Local MapReduce Simulation

- Visit `https://github.com/d2207197/local-mapreduce`

  ```
  $ cat <data> | ./lmr <chunk size> <#reducer> <mapper> <reducer> <output dir>
  $ ./lmr <chunk size> <num of jobs> <mapper> <reducer> <output directory> < <data>
  <chunk size>: Split input data into chunks with <chunk size>. #chunks = #mappers
  #reducer: Output from mappers are hashed to feed each of <#reducer> reducers
  <mapper>, <reducer>: Any executable shell command can be mapper or reducer
  <output dir>: The output directory.
  ```

- Install lmr as instructed
- Test the example cases
  - word count:
    - $< mapper >$ = tr -sc 'a-zA-Z' '*n' * = blackslash
    - $< reducer >$ = sort -k 1,1 -t $ '*t' | uniq -c
    - cat data | ./lmr 5m 8 "tr -sc 'a-zA-Z' '*n' " "sort -k 1,1 -t $ '*t' | uniq -c" result
  - ngram: nc.map.py, nc.reduce.py
    - cat data | ./lmr 5m 8 "nc.map.py" "nc.reduce.py" result

Word Count - wc.map.py

```
import sys, gzip, re

def tokens(str1): return re.findall('[a-z]+', str1.lower())

N_SEGS = 3
seg_file = [ gzip.open('temp.part-%02d.gz' % seg_id, 'a') \
                for seg_id in range(N_SEGS) ]

line = sys.stdin.readline()
while True:
    if line == '': break
    for word in tokens(line):
        seg_file[hash(word) % N_SEGS].write('%s\t%s\n' % (word, 1))
    line = sys.stdin.readline()

for seg_id in range(N_SEGS): part_file[seg_id].close()
```

Word Count - wc.reduce.py

```
import fileinput
from collections import Counter, defaultdict

ngram_counts = defaultdict(Counter)
for line in fileinput.input():
    ngram, count = line.split('\t', 1)
    ngram, count = tuple(ngram.split(' ')), int(count)
    length = len(ngram)
    ngram_counts[length][ngram] += count

for length in ngram_counts:
    for ngram, count in ngram_counts[length].iteritems():
        print '{}\t{}'.format(' '.join(ngram), count)
```

Testing wc.map.py and wc.reduce.py

- Test the mapper
  - head -100 lab4.sent | python wc.map.py | less
- Test the reducer
  - head -100 lab4.sent | python wc.map.py | sort | python wc.reduce.py | less
- Test mapreduce with lmr (local mapreduce)
  - rm -r result ; time pv lab4.sent — ./lmr 8m 8 'python nc.map.py' 'python nc.reduce.py' result

Lab #4 Collocations via MapReduce

- Dataset: **lab4.sent.gz** (unzip and produce **lab4.sent**)
- Modify nc.map.py to emit (key, value)
  - Key = word
  - Value = collocate + distance + count
- Modify nc.reduce.py to calcuate
  - Raw statistics: total, average, standard deviation,
  - Smadja's 3 Conditions: strength, spread, peaks
- Test the mapper (**coll.map.py**)
  - Same as testing wc.map.py
- Test the reducer (**coll.reduce.py**)
  - Same as testing wc.map.py
- Run mapper and reducer together
  - rm -r result ; time pv lab4.sent — ./lmr 8m 8 'python coll.map.py' 'python coll.reduce.py' result