Hands On: Multimedia Methods for Large Scale Video Analysis (Lecture)

Dr. Gerald Friedland, fractor@icsi.berkeley.edu
Today

- Comments on the Mid-Term
- MapReduce
  - Intro
  - Implementation
- Algorithm Design

Slides excerpts by
- Malcolm Slanley, Microsoft Research
- Jimmy Lin, Chris Dyer, UMD
Midterm Results

- Max Score: 40 points = 100%
- Mean Result: 33.2 points = 83%
- Variance: 18.4 points, StdDev: 4.28
Comments on the Mid-Term

Memory thrashing:
If a process does not have enough pages, thrashing is a high paging activity, and the page-fault rate is high.

=> low CPU utilization, high I/O.
Unix Command: top

```
Processes: 137 total, 4 running, 3 stuck, 130 sleeping, 759 threads
Load Avg: 0.95, 0.92, 0.83 CPU usage: 2.50% user, 2.39% sys, 95.9% idle
SharedLibs: 208M resident, 0B data, 23M linkedit.
MemRegions: 52927 total, 2450M resident, 78M private, 1062M shared.
PhysMem: 1371M wired, 4008M active, 2386M inactive, 7765M used, 416M free.
VM: 310G vsize, 1285M framework vsize, 1384644(0) pageins, 484(0) pageouts.
Disks: 3263611/39G read, 1726733/58G written.
```

<table>
<thead>
<tr>
<th>PID</th>
<th>COMMAND</th>
<th>%CPU</th>
<th>TIME</th>
<th>#TH</th>
<th>#WQ</th>
<th>#POR</th>
<th>#MREGS</th>
<th>RPRVT</th>
<th>RSHRD</th>
<th>RSIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>93240</td>
<td>top</td>
<td>8.7</td>
<td>00:01.58</td>
<td>1/1</td>
<td>0</td>
<td>24</td>
<td>30</td>
<td>1436K</td>
<td>216K</td>
<td>2184K</td>
</tr>
<tr>
<td>93232</td>
<td>CVMCompiler</td>
<td>0.0</td>
<td>00:00.13</td>
<td>2</td>
<td>2</td>
<td>37</td>
<td>79</td>
<td>11M</td>
<td>228K</td>
<td>21M</td>
</tr>
<tr>
<td>93223</td>
<td>ocspd</td>
<td>0.0</td>
<td>00:00.00</td>
<td>1</td>
<td>0</td>
<td>22</td>
<td>24</td>
<td>208K</td>
<td>212K</td>
<td>1468K</td>
</tr>
<tr>
<td>92823</td>
<td>mdworker</td>
<td>0.0</td>
<td>00:05.18</td>
<td>4</td>
<td>1</td>
<td>57</td>
<td>94</td>
<td>9420K</td>
<td>19M</td>
<td>19M</td>
</tr>
<tr>
<td>92821</td>
<td>mdworker</td>
<td>0.0</td>
<td>00:00.51</td>
<td>3</td>
<td>1</td>
<td>55</td>
<td>83</td>
<td>2732K</td>
<td>3076K</td>
<td>8960K</td>
</tr>
<tr>
<td>92466</td>
<td>plugin-conta</td>
<td>0.7</td>
<td>00:13.98</td>
<td>10</td>
<td>1</td>
<td>231</td>
<td>658</td>
<td>18M</td>
<td>90M</td>
<td>42M</td>
</tr>
<tr>
<td>92465</td>
<td>VDCAssistant</td>
<td>0.0</td>
<td>00:00.04</td>
<td>4</td>
<td>1</td>
<td>76</td>
<td>91</td>
<td>1204K</td>
<td>6980K</td>
<td>5900K</td>
</tr>
<tr>
<td>92464-</td>
<td>GoogleTalkPl</td>
<td>0.0</td>
<td>00:01.61</td>
<td>10</td>
<td>1</td>
<td>208</td>
<td>311</td>
<td>4288K</td>
<td>52M</td>
<td>12M</td>
</tr>
<tr>
<td>92463-</td>
<td>plugin-conta</td>
<td>0.0</td>
<td>00:00.16</td>
<td>5</td>
<td>1</td>
<td>115</td>
<td>220</td>
<td>4824K</td>
<td>87M</td>
<td>19M</td>
</tr>
<tr>
<td>86892</td>
<td>com.apple.hi</td>
<td>0.0</td>
<td>00:00.02</td>
<td>2</td>
<td>1</td>
<td>44</td>
<td>55</td>
<td>648K</td>
<td>224K</td>
<td>4332K</td>
</tr>
<tr>
<td>86885</td>
<td>Preview</td>
<td>0.0</td>
<td>00:01.09</td>
<td>2</td>
<td>1</td>
<td>149</td>
<td>279</td>
<td>28M</td>
<td>61M</td>
<td>64M</td>
</tr>
<tr>
<td>78993</td>
<td>cfprefsd</td>
<td>0.0</td>
<td>00:00.01</td>
<td>2</td>
<td>2</td>
<td>29</td>
<td>58</td>
<td>592K</td>
<td>220K</td>
<td>1280K</td>
</tr>
<tr>
<td>78991</td>
<td>launchd</td>
<td>0.0</td>
<td>00:00.01</td>
<td>2</td>
<td>0</td>
<td>48</td>
<td>46</td>
<td>484K</td>
<td>384K</td>
<td>976K</td>
</tr>
<tr>
<td>72309</td>
<td>com.apple.se</td>
<td>0.0</td>
<td>00:00.77</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>113</td>
<td>4080K</td>
<td>48M</td>
<td>14M</td>
</tr>
</tbody>
</table>
MapReduce

• Idea originated in functional programming

• First large use for parallelization at Google for accessing BigTable.

• Killer app: Text indexing!
Basic Parallelization Pattern
Basic Parallelization Pattern

“Work”
Basic Parallelization Pattern
Basic Parallelization Pattern

```
<table>
<thead>
<tr>
<th></th>
<th>W1</th>
<th>W2</th>
<th>W3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work</td>
<td>w1</td>
<td>w2</td>
<td>w3</td>
</tr>
<tr>
<td></td>
<td>w1</td>
<td>w2</td>
<td>w3</td>
</tr>
<tr>
<td></td>
<td>r1</td>
<td>r2</td>
<td>r3</td>
</tr>
</tbody>
</table>
```

"worker"
Basic Parallelization Pattern

```
<table>
<thead>
<tr>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>( w_1 )</td>
</tr>
<tr>
<td>( w_2 )</td>
</tr>
<tr>
<td>( w_3 )</td>
</tr>
<tr>
<td>worker</td>
</tr>
<tr>
<td>( r_1 )</td>
</tr>
<tr>
<td>( r_2 )</td>
</tr>
<tr>
<td>( r_3 )</td>
</tr>
<tr>
<td>Result</td>
</tr>
</tbody>
</table>
```
Basic Parallelization Pattern

Partition

"Work"

$w_1$

"worker"

$r_1$

$w_2$

"worker"

$r_2$

$w_3$

"worker"

$r_3$

"Result"
Basic Parallelization Pattern

Partition

Combine

"Work"

\[ w_1 \]

\[ w_2 \]

\[ w_3 \]

"worker"

\[ r_1 \]

"worker"

\[ r_2 \]

"worker"

\[ r_3 \]

"Result"
Reality
Reality

Fundamental issues
scheduling, data distribution, synchronization,
inter-process communication, robustness, fault
tolerance, …
Reality

Fundamental issues
- scheduling, data distribution, synchronization,
- inter-process communication, robustness, fault
tolerance, ...

Architectural issues
- Flynn’s taxonomy (SIMD, MIMD, etc.),
- network typology, bisection bandwidth
- UMA vs. NUMA, cache coherence
Reality

Fundamental issues
- scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

Architectural issues
- Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth
- UMA vs. NUMA, cache coherence

Different programming models
- Message Passing
- Shared Memory
Fundamental issues
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

Architectural issues
Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth UMA vs. NUMA, cache coherence

Common problems
livelock, deadlock, data starvation, priority inversion…

dining philosophers, sleeping barbers, cigarette smokers, …
Fundamental issues
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, ...

Architectural issues
Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth UMA vs. NUMA, cache coherence

Common problems
livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

Different programming models
Message Passing
Shared Memory

Different programming constructs
mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …
Fundamental issues
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …

Architectural issues
Flynn’s taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth UMA vs. NUMA, cache coherence

Common problems
livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

Different programming models
Message Passing
Shared Memory

Different programming constructs
mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …

The reality: programmer shoulders the burden of managing concurrency… Solutions: See PARLAB
Common Workflow

• Iterate over a large number of records
• Extract something of interest from each
• Shuffle and sort intermediate results
• Aggregate intermediate results
• Generate final output

(Dean and Ghemawat, OSDI 2004)
Common Workflow

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
-Aggregate intermediate results
- Generate final output

(Dean and Ghemawat, OSDI 2004)
Common Workflow

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

(Dean and Ghemawat, OSDI 2004)
Common Workflow

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

**Key idea:** provide a functional abstraction for these two operations

(Dean and Ghemawat, OSDI 2004)
MapReduce as a Diagram
MapReduce as a Diagram
MapReduce as a Diagram
MapReduce as a Diagram

Map

\[
\begin{array}{ccccc}
\text{Map} & f & f & f & f & f \\
\downarrow & & & & & \\
\text{Reduce} & & & & & \\
\end{array}
\]
MapReduce as a Diagram

Map
MapReduce as a Diagram

Map

f f f f f

g g

Thursday, November 15, 12
MapReduce as a Diagram

Map

\[ f \quad f \quad f \quad f \quad f \quad f \]

\[ g \quad g \quad g \]

\[ \text{Reduce} \]
MapReduce as a Diagram

Map

\[
\begin{align*}
\text{Map} & : f \rightarrow g \\
\text{Reduce} & : g \rightarrow g
\end{align*}
\]
MapReduce as a Diagram

Map

Fold

f f f f f f

g g g g g g

Thursday, November 15, 12
MapReduce as a Diagram
MapReduce in Haskell

map :: (A→B) → [A] → [B]
map f [] = []
map f (x:xs) = f x : map f xs

reduce :: (A→B→B) → B → [A] → B
reduce f y [] = y
reduce f y (x:xs) = f x (reduce f y xs)

Very parallelizable!
MapReduce in Practice

• Programmers specify two functions:
  \[
  \text{map} \ (k, v) \rightarrow \langle k', v' \rangle^* \\
  \text{reduce} \ (k', v') \rightarrow \langle k', v' \rangle^*
  \]
  - All values with the same key are reduced together

• Usually, programmers also specify:
  \[
  \text{partition} \ (k', \text{number of partitions}) \\
  \rightarrow \text{partition for } k'
  \]
  - Often a simple hash of the key, e.g. hash(k') mod n
MapReduce
A MapReduce Engine Typically

• Handles scheduling
  - Assigns workers to map and reduce tasks

• Handles “data distribution”
  - Moves the process to the data

• Handles synchronization
  - Gathers, sorts, and shuffles intermediate data

• Handles faults
  - Detects worker failures and restarts
What to do with I/O?

• Don’t move data to workers… Move workers to the data!
  - Store data on the local disks for nodes in the cluster
  - Start up the workers on the node that has the data local

• Why?
  - Not enough RAM to hold all the data in memory
  - Disk access is slow, need to be serialized!
  - Disk becomes bottleneck!
Distributed File System

• A distributed file system is the answer
  - GFS (Google File System)
  - HDFS for Hadoop (= GFS clone)
  - Amazon S3
GFS: Design Decisions

- Files stored as chunks
  - Fixed size (64MB)
- Reliability through replication
  - Each chunk replicated across 3+ chunkservers
- Single master to coordinate access, keep metadata
  - Simple centralized management
- No data caching
  - Little benefit due to large data sets, streaming reads
- Simplify the API
  - Push some of the issues onto the client
MapReduce Implementation
Hadoop
Hadoop Word Count

- `#!/usr/bin/env python`
- `import sys`
- `# input comes from STDIN (standard input)`
- `for line in sys.stdin:`
  - `# remove leading and trailing whitespace`
  - `line = line.strip()`
  - `# split the line into words`
  - `words = line.split()`
  - `# increase counters`
  - `for word in words:`
    - `# write the results to STDOUT (standard output);`
    - `# what we output here will be the input for the`
    - `# Reduce step, i.e. the input for reducer.py #`
    - `# tab-delimited; the trivial word count is 1`
    - `print '%s\t%\'s' % (word, 1)`
Hadoop Word Count

- `#!/usr/bin/env python`
- `import sys`
- `word2count = {}`
- `# input comes from STDIN`
- `for line in sys.stdin:`
  - `# remove leading and trailing whitespace`
  - `line = line.strip()`
  - `# parse the input we got from mapper.py`
  - `word, count = line.split(\"\t\", 1)`
  - `# convert count (currently a string) to int`
  - `try:`
    - `count = int(count)`
    - `word2count[word] = word2count.get(word, 0) + count`
  - `except ValueError:`
    - `# count was not a number, so silently`
    - `# ignore/discard this line`
    - `pass`
Word Count Command

- `bin/hadoop jar contrib/streaming/hadoop-0.19.1streaming.jar`
  - mapper `/home/hadoop/mapper.py`
  - reducer `/home/hadoop/reducer.py`
  - input gutenberg/*
  - output gutenberg-output
def ReadAndDispatch():
    while True:
        theLine = sys.stdin.readline()
        if theLine == '' or theLine == None:
            break
        if theLine[0] == '#':
            continue
        sys.stderr.write("Working on " + theLine + "\n")
        args = theLine.split()
        if len(args) < 1:
            continue
        if args[0] == 'expected':
            GetOneResult(int(args[1]), int(args[2]), \
                          int(args[3]), int(args[4]), \
                          float(args[5]))
        elif args[0] == 'hyperexpected':
            GetOneResult(int(args[1]), int(args[2]), \
                          int(args[3]), int(args[4]), \
                          float(args[5]), hyper=float(args[6]))
        else:
            print "Unknown command: ", args[0]
Run Hadoop Example

- $HADOOP_HOME/bin/hadoopfs -rmrmyOutputDir
- EXP=run.tst8

- $HADOOP_HOME/bin/hadoopfs -rm $EXP
- $HADOOP_HOME/bin/hadoopfs -put $EXP .

- $HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar
  -Dmapred.job.queue.name=keystone
  -Dmapred.map.tasks=50
  -Dmapred.task.timeout=9999000
  -Dmapred.reduce.tasks=0
  -archives hdfs://axoniteblue-nn1.blue.ygrid.yahoo.com:8020/user/malcolm/numpy4python2.5.tgz
  -cmdenv PYTHONPATH=./numpy4python2.5.tgz/
  -cmdenv LD_LIBRARY_PATH=./numpy4python2.5.tgz/numpy
  -input $EXP
  -mapper "${HOD_PYTHON_HOME} TestRecall.py -hadoop"
  -output myOutputDir
  -reducer /bin/cat
  -file ls.py
  -file TestRecall.py

- $HADOOP_HOME/bin/hadoopfs -cat myOutputDir/part* > $EXP.out
### Hadoop Status

**User:** malcolm  
**Job Name:** streamjob30175.jar  
**Job File:** hdfs://axoniteblue-nn1.blue.ygrid.yahoo.com/mapredsystem/hadoop/mapredsystem/job_200906290541_20345/job.xml  
**Job Setup:** Successful  
**Status:** Running  
**Started at:** Fri Jul 17 04:58:41 UTC 2009  
**Running for:** 3hrs, 15mins, 12sec  
**Job Cleanup:** Pending  
**Job Scheduling information:** 681 running map tasks using 681 map slots, 0 running reduce tasks using 0 reduce slots.

<table>
<thead>
<tr>
<th>Kind</th>
<th>% Complete</th>
<th>Num Tasks</th>
<th>Pending</th>
<th>Running</th>
<th>Complete</th>
<th>Killed</th>
<th>Failed/Killed Task Attempts</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>99.99%</td>
<td>2001</td>
<td>0</td>
<td>681</td>
<td>1320</td>
<td>0</td>
<td>0 / 0</td>
</tr>
<tr>
<td>reduce</td>
<td>0.00%</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 / 0</td>
</tr>
</tbody>
</table>

### Job Counters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rack-local map tasks</td>
<td>0</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td>0</td>
<td>2,001</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

### FileSystemCounters

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS_BYTES_READ</td>
<td>255,342,725</td>
<td>0</td>
<td>255,342,725</td>
</tr>
<tr>
<td>HDFS_BYTES_WRITTEN</td>
<td>1,270,793,148</td>
<td>0</td>
<td>1,270,793,148</td>
</tr>
</tbody>
</table>

### Map-Reduce Framework

<table>
<thead>
<tr>
<th>Counter</th>
<th>Map</th>
<th>Reduce</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map input records</td>
<td>17,280</td>
<td>0</td>
<td>17,280</td>
</tr>
<tr>
<td>Spilled Records</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Map input bytes</td>
<td>2,478,624</td>
<td>0</td>
<td>2,478,624</td>
</tr>
<tr>
<td>Map output records</td>
<td>34,353,483</td>
<td>0</td>
<td>34,353,483</td>
</tr>
</tbody>
</table>
Pricing

Amazon Elastic MapReduce currently is available in the US and EU Regions. Pay only for what you use – there is no minimum fee. Amazon Elastic MapReduce pricing is in addition to normal Amazon EC2 and Amazon S3 pricing.

<table>
<thead>
<tr>
<th>United States</th>
<th>Europe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard On-Demand Instances</td>
<td>Amazon EC2 Price per hour (On-Demand Instances)</td>
</tr>
<tr>
<td>Small (Default)</td>
<td>$0.10 per hour</td>
</tr>
<tr>
<td>Large</td>
<td>$0.40 per hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.80 per hour</td>
</tr>
<tr>
<td>High CPU On-Demand Instances</td>
<td>Amazon EC2 Price per hour (On-Demand Instances)</td>
</tr>
<tr>
<td>Medium</td>
<td>$0.20 per hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.80 per hour</td>
</tr>
</tbody>
</table>
MapReduce Algorithm Design
Managing Dependencies
Managing Dependencies

• Remember: Mappers run in isolation
  - You have no idea in what order the mappers run
  - You have no idea on what node the mappers run
  - You have no idea when each mapper finishes
Managing Dependencies

• Remember: Mappers run in isolation
  - You have no idea in what order the mappers run
  - You have no idea on what node the mappers run
  - You have no idea when each mapper finishes

• Tools for synchronization:
  - Ability to hold state in reducer across multiple key–value pairs
  - Sorting function for keys
  - Partitioner
  - Cleverly–constructed data structures
Motivating Example
Motivating Example

- Term co-occurrence matrix for a text collection
  - $M = N \times N$ matrix ($N =$ vocabulary size)
  - $M_{ij}$: number of times $i$ and $j$ co-occur in some context
    (for concreteness, let’s say context = sentence)
Motivating Example

• Term co-occurrence matrix for a text collection
  - $M = N \times N$ matrix ($N =$ vocabulary size)
  - $M_{ij}$: number of times $i$ and $j$ co-occur in some context
    (for concreteness, let’s say context = sentence)

• Why?
  - Distributional profiles as a way of measuring semantic distance
  - Semantic distance useful for many language processing tasks
• Term co-occurrence matrix for a text collection = specific instance of a large counting problem
  - A large event space (number of terms)
  - A large number of observations (the collection itself)
  - Goal: keep track of interesting statistics about the events

• Basic approach
  - Mappers generate partial counts
  - Reducers aggregate partial counts
MapReduce: Large

- Term co-occurrence matrix for a text collection = specific instance of a large counting problem
  - A large event space (number of terms)
  - A large number of observations (the collection itself)
  - Goal: keep track of interesting statistics about the events

- Basic approach
  - Mappers generate partial counts
  - Reducers aggregate partial counts

How do we aggregate partial counts efficiently?
First Try: “Pairs”

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit \((a, b) \rightarrow \text{count}\)
- Reducers sums up counts associated with these pairs
- Use combiners!
“Pairs” Analysis

• Advantages
  - Easy to implement, easy to understand

• Disadvantages
  - Lots of pairs to sort and shuffle around (upper bound?)
• Idea: group together pairs into an associative array

\[
\begin{align*}
(a, b) & \rightarrow 1 \\
(a, c) & \rightarrow 2 \\
(a, d) & \rightarrow 5 \\
(a, e) & \rightarrow 3 \\
(a, f) & \rightarrow 2 
\end{align*}
\]

• Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For each term, emit \( a \rightarrow \{ b: \text{count}_b, c: \text{count}_c, d: \text{count}_d \ldots \} \)

• Reducers perform element-wise sum of associative arrays

\[
\begin{align*}
a & \rightarrow \{ b: 1, d: 5, e: 3 \} \\
+ a & \rightarrow \{ b: 1, c: 2, d: 2, f: 2 \} \\
a & \rightarrow \{ b: 2, c: 2, d: 7, e: 3, f: 2 \}
\end{align*}
\]
“Stripes” Analysis

• Advantages
  - Far less sorting and shuffling of key-value pairs
  - Can make better use of combiners

• Disadvantages
  - More difficult to implement
  - Underlying object is more heavyweight
  - Fundamental limitation in terms of size of event space
Efficiency comparison of approaches to computing word co-occurrence matrices

Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

Thursday, November 15, 12
Conditional Probabilities

Approach

• How do we estimate conditional probabilities from counts?

\[
P(B \mid A) = \frac{\text{count}(A, B)}{\text{count}(A)} = \frac{\text{count}(A, B)}{\sum_{B'} \text{count}(A, B')}
\]

• Why do we want to do this?

• How do we do this with MapReduce?
P(B|A): “Stripes”

\[ a \rightarrow \{ b_1 : 3, b_2 : 12, b_3 : 7, b_4 : 1, \ldots \} \]

- Easy!
  - One pass to compute (a, *)
  - Another pass to directly compute P(B|A)
**P(B|A): “Pairs”**

- **For this to work:**
  - Must emit extra (a, *) for every $b_n$ in mapper
  - Must make sure all a’s get sent to same reducer (use partitioner)
  - Must make sure (a, *) comes first (define sort order)
  - Must hold state in reducer across different key-value pairs

| (a, b₁) | 3   |
| (a, b₂) | 12  |
| (a, b₃) | 7   |
| (a, b₄) | 1   |

Reduction:

- (a, *) $\rightarrow$ 32
- (a, b₁) $\rightarrow$ 3 / 32
- (a, b₂) $\rightarrow$ 12 / 32
- (a, b₃) $\rightarrow$ 7 / 32
- (a, b₄) $\rightarrow$ 1 / 32

Reducer holds this value in memory
Synchronization in Hadoop

• Approach 1: turn synchronization into an ordering problem
  - Sort keys into correct order of computation
  - Partition key space so that each reducer gets the appropriate set of partial results
  - Hold state in reducer across multiple key-value pairs to perform computation
  - Illustrated by the “pairs” approach
Synchronization in Hadoop

• Approach 2: construct data structures that “bring the pieces together”
  - Each reducer receives all the data it needs to complete the computation
  - Illustrated by the “stripes” approach
Issues and Tradeoffs

• Number of key–value pairs
  - Object creation overhead
  - Time for sorting and shuffling pairs across the network

• Size of each key–value pair
  - De/serialization overhead

• Combiners make a big difference!
  - RAM vs. disk and network
  - Arrange data to maximize opportunities to aggregate partial results
In general: Issues
In general: Issues

• The optimally-parallelized version doesn’t exist!
In general: Issues

• The optimally-parallelized version doesn’t exist!

• It’s all about the right level of abstraction
In general: Issues

• The optimally-parallelized version doesn’t exist!

• It’s all about the right level of abstraction
In general: Issues

• The optimally-parallelized version doesn’t exist!

• It’s all about the right level of abstraction

• Hadoop has Overhead!
Next Week (Project Meeting)

- Stephanie Pancoast (Stanford)
Next Week (Lecture)

Mehmet Emre Sargin
(Google Research/YouTube)