LOOKING AHEAD: PARALLELIZING STREAM AND SORT
Stream and Sort Counting \(\rightarrow\) Distributed Counting

- example 1
- example 2
- example 3
- ...

Counting logic

\[ C[x] += D \]

Machines A1, ...

Standardized message routing logic

- \( C[x1] += D1 \)
- \( C[x1] += D2 \)
- ...

Logic to combine counter updates

Machines C1, ...

Trivial to parallelize!

Easy to parallelize!
Stream and Sort Counting → Distributed Counting

- example 1
- example 2
- example 3
- ....

“C[x] += D”

Machines A1, ...

Logic to combine counter updates

- C[x1] += D1
- C[x1] += D2
- ....

Machines C1, ..,
Stream and Sort Counting → Distributed Counting

- example 1
- example 2
- example 3
- ....

Counter Machine

```
C[x] += D
```

Logic to combine counter updates

```
C[x1] += D1
C[x1] += D2
....
```

Combiner Machine

```
```

Sort

Merge Spill Files

Spill 1

Spill 2

Spill 3

....
Stream and Sort Counting → Distributed Counting

Counter Machine 1
- Example 1
- Example 2
- Example 3
- ...

Counting logic
Partition/Sort
- Spill 1
- Spill 2
- Spill 3
- ...
- Spill n

Logic to combine counter updates
Merge Spill Files

Counter Machine 2
- Example 1
- Example 2
- Example 3
- ...

Counting logic
Partition/Sort
- Spill 1
- Spill 2
- Spill 3
- ...

Logic to combine counter updates
Merge Spill Files

Combiner Machine 1
- \( C[x1] += D1 \)
- \( C[x1] += D2 \)
- ...

Combiner Machine 2
- \( C[x1] += D1 \)
- \( C[x1] += D2 \)
- ...

Combine counter updates
From stream+sort to hadoop

- We’ve looked at some simple algorithms expressed as
  - Sorting (to organize messages)
  - Streaming (low-memory, line-by-line) file transformations ("map" operations)
  - Streaming “reduce” operations, like summing counts, that input files sorted by keys and operate on contiguous runs of lines with the same keys
Today: from stream+sort to hadoop

• Important point:
  • Our code is *not* CPU-bound
  • It’s I/O bound
  • To speed it up, we need to add more *disk drives*, *not* more *CPUs*.
  • Example: finding a particular line in 1 TB of data
INADVISABLE SCIENCE

THE ASSIGNMENT THAT NEVER EXISTED

A HORROR that could ONLY be IMAGINED!
Write code to run assignment 1 in parallel

- What infrastructure would you need?
- How could you run a generic “stream-and-sort” algorithm in parallel?
  - `cat input.txt | MAP | sort | REDUCE > output.txt`

Key-value pairs (one/line)
e.g., labeled docs

Key-value pairs (one/line)
e.g., event counts

Sorted key-val pairs

Key-value pairs (one/line)
e.g., aggregate counts
How would you run assignment 1 in parallel?

- What infrastructure would you need?
- How could you run a generic “stream-and-sort” algorithm in parallel?
  
  - `cat input.txt` | MAP | sort | REDUCE > output.txt

Key-value pairs (one/line) e.g., labeled docs

Step 1: split input data, by key, into “shards” and ship each shard to a different box
How would you run assignment 1 in parallel?

- What infrastructure would you need?
- How could you run a generic stream-and-sort algorithm in parallel?

- `cat input.txt | MAP | sort | REDUCE > output.txt`

**Step 1:** split input data, by key, into “shards” and ship each shard to a different box.
How would you run assignment 1 in parallel?

- Open sockets to receive data to boxk:/kludge/mapin.txt on each of the K boxes
- For each key,val pair in input.txt:
  - Send key,val pair to boxFor (key)
- Run K processes: rsh boxk ‘MAP < mapin.txt > mapout.txt’

What infrastructure would you need?

How could you run a generic “stream-and-sort” algorithm in parallel?

- cat input.txt | MAP | sort | REDUCE > output.txt
• Open sockets to receive data to boxk:/kludge/mapin.txt on each of the K boxes
• For each key,val pair in input.txt:
  • Send key,val pair to socket[boxFor (key)]
• Run K processes: rsh … ‘MAP < ….> …’ to completion
• On each box:
  • Open sockets to receive and sort data to boxk:/kludge/redin.txt on each of the K boxes
  • For each key,val pair in mapout.txt:
    • Send key,val pair to socket[boxFor (key)]
• Open sockets to receive data to boxk:/kludge/mapin.txt on each of the K boxes
• For each key,val pair in input.txt:
  • Send key,val pair to socket[boxFor (key)]
• Run K processes: rsh MAP ...
• On each box:
  • Open sockets to receive and sort data to boxk:/kludge/redin.txt on each of the K boxes
  • For each key,val pair in mapout.txt:
    • Send key,val pair to socket[boxFor (key)]

Step 3: redistribute the map output
How would you run assignment 1B in parallel?

- Open sockets to receive data to boxk:/kludge/mapin.txt on each of the K boxes
- For each key,val pair in input.txt:
  - Send key,val pair to socket[boxFor (key)]
- Run K processes: rsh MAP < mapin.txt > mapout.txt
- Shuffle the data back to the right box
- Do the same steps for the reduce processes

\[
\text{cat input.txt } | \text{ MAP } | \text{ sort } | \text{ REDUCE } > \text{ output.txt}
\]
• Open sockets to receive data to boxk:/kludge/mapin.txt on each of the K boxes
• For each key, val pair in input.txt:
  • Send key, val pair to socket[boxFor (key)]
• Run K processes: rsh MAP < mapin.txt > mapout.txt
• Shuffle the data back to the right box
• Do the same steps for the reduce process
  • (If the keys for reduce process don’t change, you don’t need to reshuffle them)
1. This would be pretty systems-y (remote copy files, waiting for remote processes, …)

2. It would take work to make it useful….
Motivating Example*

* not to scale

INTERNET

Your dataset
1. This would be pretty systems-y (remote copy files, waiting for remote processes, ...)

2. It would take work to make run for 1000 jobs

- Reliability: Replication, restarts, monitoring jobs, ...
- Efficiency: load-balancing, reducing file/network i/o, optimizing file/network i/o, ...
- Useability: stream defined datatypes, simple reduce functions, ....
Event Counting on Subsets of Documents

Summing Counts
1. This would be pretty systems-y (remote copy files, waiting for remote processes, ...)
2. It would take work to make run for 1000 jobs
   - **Reliability**: Replication, restarts, monitoring jobs,...
   - **Efficiency**: load-balancing, reducing file/network i/o, optimizing file/network i/o, ...
   - **Useability**: stream defined datatypes, simple reduce functions, ....
Event Counting on Subsets of Documents
Hadoop: Intro

• pilfered from: Alona Fyshe, Jimmy Lin, Google, Cloudera

  • http://code.google.com/edu/submissions/mapreduce-minilecture/listing.html
  • http://code.google.com/edu/submissions/mapreduce/listing.html
Surprise, you mapreduced!

- Mapreduce has three main phases
  - Map (send each input record to a key)
  - Sort (put all of one key in the same place)
    - handled behind the scenes
  - Reduce (operate on each key and its set of values)
    - Terms come from functional programming:
      - map(lambda x:x.upper(),
        ['william', 'w', 'cohen']) ➞ ['WILLIAM', 'W', 'COHEN']
      - reduce(lambda x,y:x+"-"+y,
        ['william', 'w', 'cohen']) ➞ "william-w-cohen"
Mapreduce overview
Mapreduce: slow motion

- The canonical mapreduce example is word count
- Example corpus:
  - Joe likes toast
  - Jane likes toast with jam
  - Joe burnt the toast
MR: slow motion: Map

Input

Joe likes toast

Jane likes toast with jam

Joe burnt the toast

Output

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
<tr>
<td>Jane</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
</tr>
<tr>
<td>jam</td>
<td>1</td>
</tr>
<tr>
<td>burnt</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
</tbody>
</table>
**MR: slow motion: Sort**

**Input**
- Joe 1
- likes 1
- toast 1

- Jane 1
- likes 1
- toast 1
- with 1
- jam 1

- Joe 1
- burnt 1
- the 1
- toast 1

**Output**
- Joe 1
- Jane 1
- likes 1
- likes 1
- toast 1
- toast 1
- toast 1
- with 1
- jam 1
- burnt 1
- the 1
MR: slow mo: Reduce

**Input**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>1</td>
</tr>
<tr>
<td>Jane</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
<tr>
<td>toast</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
</tr>
<tr>
<td>jam</td>
<td>1</td>
</tr>
<tr>
<td>burnt</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>Reduce 1</td>
<td></td>
</tr>
<tr>
<td>Reduce 2</td>
<td></td>
</tr>
<tr>
<td>Reduce 3</td>
<td></td>
</tr>
<tr>
<td>Reduce 4</td>
<td></td>
</tr>
<tr>
<td>Reduce 5</td>
<td></td>
</tr>
<tr>
<td>Reduce 6</td>
<td></td>
</tr>
<tr>
<td>Reduce 7</td>
<td></td>
</tr>
<tr>
<td>Reduce 8</td>
<td></td>
</tr>
</tbody>
</table>

**Output**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>2</td>
</tr>
<tr>
<td>Jane</td>
<td>1</td>
</tr>
<tr>
<td>likes</td>
<td>2</td>
</tr>
<tr>
<td>toast</td>
<td>3</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
</tr>
<tr>
<td>jam</td>
<td>1</td>
</tr>
<tr>
<td>burnt</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
</tbody>
</table>
Issue: reliability

• Questions:
  – How will you know when each machine is done?
    • Communication overhead
  – How will you know if a machine is dead?
    • Remember: we can use a huge pile of cheap machines, so failures will be common!
  – Is it dead or just really slow?
Issue: reliability

• What’s the difference between slow and dead?
  – Who cares? Start a backup process.
    • If the process is slow because of machine issues, the backup may finish first
    • If it’s slow because you poorly partitioned your data... waiting is your punishment
Issue: reliability

• If a disk fails you can lose some intermediate output
  • Ignoring the missing data could give you wrong answers

• Who cares? if I’m going to run backup processes I might as well have backup copies of the intermediate data also
HDFS: The Hadoop File System

• Distributes data across the cluster
  • distributed file *looks like* a directory with shards as files inside it
  • makes an effort to run processes *locally* with the data
• Replicates data
  • default 3 copies of each file
• Optimized for streaming
  • really really big “blocks”
$ hadoop fs -ls rcv1/small/sharded
Found 10 items
-rw-r--r-- 3 ...  606405 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00000
-rw-r--r-- 3 ...  1347611 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00001
-rw-r--r-- 3 ...  939307 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00002
-rw-r--r-- 3 ...  1284062 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00003
-rw-r--r-- 3 ...  1009890 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00004
-rw-r--r-- 3 ...  1206196 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00005
-rw-r--r-- 3 ...  1384658 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00006
-rw-r--r-- 3 ...  1299698 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00007
-rw-r--r-- 3 ...  928752 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00008
-rw-r--r-- 3 ...  806030 2013-01-22 16:28 /user/wcohen/rcv1/small/sharded/part-00009

$ hadoop fs -tail rcv1/small/sharded/part-00005
weak as the arrival of arbitraged cargoes from the West has put the local market under pressure...
M14,M143,MCAT  The Brent crude market on the Singapore International ...
MR Overview
Hadoop job_201301231150_0778 on hadoopjt

User: wcohen
Job Name: streamjob6055532903853567038.jar
Job Setup: Successful
Status: Failed
Started at: Wed Jan 30 11:46:47 EST 2013
Failed in: 41sec
Job Cleanup: Successful
Black-listed TaskTrackers: 2
Job Scheduling information: 5 running map tasks using 5 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>100.00%</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>35 / 5</td>
</tr>
<tr>
<td>reduce</td>
<td>00%</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</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>38</td>
</tr>
<tr>
<td>Launched map tasks</td>
<td>0</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>Data-local map tasks</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Failed map tasks</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Map Completion Graph - close
<table>
<thead>
<tr>
<th>Task</th>
<th>Complete</th>
<th>Status</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_201301231150_0778_m_000000</td>
<td>0.00%</td>
<td></td>
<td>30-Jan-2013 11:47:01</td>
<td>30-Jan-2013 11:47:25 (24sec)</td>
<td>java.lang.RuntimeException: PipeMa... at org.apache.hadoop.mapreduce... Java... at org.apache.hadoop.mapreduce...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## All Task Attempts

<table>
<thead>
<tr>
<th>Task Attempts</th>
<th>Machine</th>
<th>Status</th>
<th>Progress</th>
<th>Start Time</th>
<th>Finish Time</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>attempt_201301231150_0778_m_000000_0</td>
<td>/default-rack/cloud3u12.opencloud</td>
<td>FAILED</td>
<td>0.00%</td>
<td>30-Jan-2013 11:47:01</td>
<td>30-Jan-2013 11:47:06 (4sec)</td>
<td>java.</td>
</tr>
<tr>
<td>attempt_201301231150_0778_m_000000_1</td>
<td>/default-rack/cloud2u28.opencloud</td>
<td>FAILED</td>
<td>0.00%</td>
<td>30-Jan-2013 11:47:07</td>
<td>30-Jan-2013 11:47:11 (4sec)</td>
<td>java.</td>
</tr>
<tr>
<td>Name</td>
<td>Errors</td>
<td>Task Logs</td>
<td>Counters</td>
<td>Actions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------</td>
<td>----------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1</td>
<td>Last 4KB</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311)</td>
<td>Last 8KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132)</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1</td>
<td>Last 4KB</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311)</td>
<td>Last 8KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132)</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1</td>
<td>Last 4KB</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311)</td>
<td>Last 8KB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapper.close(PipeMapper.java:132)</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task Logs: 'attempt_201301231150_0778_m_000000_0'

stdout logs

stderr logs

Exception in thread "main" java.lang.NoClassDefFoundError: com/wcohen/StreamNB
Caused by: java.lang.ClassNotFoundException: com.wcohen.StreamNB
    at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
    at java.security.AccessController.doPrivileged(Native Method)
    at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
Could not find the main class: com.wcohen.StreamNB. Program will exit.
1. This would be pretty systems-y (remote copy files, waiting for remote processes, …)

2. It would take work to make run for 500 jobs

   • Reliability: Replication, restarts, monitoring jobs,…
   • Efficiency: load-balancing, reducing file/network i/o, optimizing file/network i/o, …
   • Useability: stream defined datatypes, simple reduce functions, …
Map reduce with Hadoop streaming
Breaking this down...

• Our imaginary assignment uses key-value pairs. What’s the data structure for that? How do you interface with Hadoop?
• One very simple way: Hadoop’s streaming interface.
  – Mapper outputs key-value pairs as:
    • One pair per line, key and value tab-separated
  – Reduced reads in data in the same format
    • Lines are sorted so lines with the same key are adjacent.
An example:

- **SmallStreamNB.java** and **StreamSumReducer.java**:
To run locally:

test-small: small-events.txt nb.jar
    time java -cp nb.jar com.wcohen.SmallStreamNB \ 
    RCV1.small_test.txt MCAT,CCAT,GCAT,ECAT 2000 < small-events.txt \ 
    | cut -f3 | sort | uniq -c

small-events.txt: nb.jar
    time java -cp nb.jar com.wcohen.SmallStreamNB \ 
    < RCV1.small_train.txt | sort -k1,1 \ 
    | java -cp nb.jar com.wcohen.StreamSumReducer> small-events.txt
To train with streaming Hadoop you do this:

```bash
STRJAR = /usr/lib/hadoop/contrib/streaming/hadoop-streaming-1.2.0.1.3.0.0-107.jar

small-events-hs:
    hadoop fs -rmr rcv1/small/events
    time hadoop jar $STRJAR \
        -input rcv1/small/sharded -output rcv1/small/events \
        -mapper 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamNB' \
        -reducer 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamSumReducer' \
        -file nb.jar -numReduceTasks 10
```

But first you need to get your code and data to the “Hadoop file system”
hadoop fs -rmr rcv1/small/events
Moved to trash: hdfs://t1disc-pnn:8020/user/wcohen/rcv1/small/events
time hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-1.2.0.1.3.0.0-107.jar \ 
  -input rcv1/small/sharded -output rcv1/small/events \ 
  -mapper 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamNB' \ 
  -reducer 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamSumReducer' \ 
  -file nb.jar -numReduceTasks 10
packageJobJar: [nb.jar, /tmp/hadoop-wcohen/hadoop-unjar8935719391413249732/] [] /tmp/streamjob

14/02/05 11:24:56 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
14/02/05 11:24:56 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [has 08101c2729dc0c9ff3]
14/02/05 11:24:56 WARN snappy.LoadSnappy: Snappy native library is available
14/02/05 11:24:56 INFO util.NativeCodeLoader: Loaded the native-hadoop library
14/02/05 11:24:56 INFO snappy.LoadSnappy: Snappy native library loaded
14/02/05 11:24:56 INFO mapred.FileInputFormat: Total input paths to process : 10
14/02/05 11:24:57 INFO streaming.StreamJob: getLocalDirs(): [/a/hadoop/mapred, /b/hadoop/\ndoop/mapred]
14/02/05 11:24:57 INFO streaming.StreamJob: Running job: job_201312100900_1189
14/02/05 11:24:57 INFO streaming.StreamJob: To kill this job, run:
14/02/05 11:24:57 INFO streaming.StreamJob: /usr/lib/hadoop/libexec/../bin/hadoop job -Dmapre\nmu.local:50300 -kill job_201312100900_1189
14/02/05 11:24:57 INFO streaming.StreamJob: Tracking URL: http://t1disc-jt.disc.pdl.cmu.local:312100900_1189
14/02/05 11:24:58 INFO streaming.StreamJob: map 0% reduce 0%
14/02/05 11:25:09 INFO streaming.StreamJob: map 20% reduce 0%
14/02/05 11:25:14 INFO streaming.StreamJob: map 57% reduce 0%
14/02/05 11:25:15 INFO streaming.StreamJob: map 77% reduce 0%
14/02/05 11:25:16 INFO streaming.StreamJob: map 79% reduce 0%
14/02/05 11:25:17 INFO streaming.StreamJob: map 79% reduce 4%
14/02/05 11:25:18 INFO streaming.StreamJob: map 80% reduce 6%
14/02/05 11:25:19 INFO streaming.StreamJob: map 79% reduce 7%
14/02/05 11:25:24 INFO streaming.StreamJob: map 80% reduce 7%
14/02/05 11:25:25 INFO streaming.StreamJob: map 80% reduce 10%
14/02/05 11:25:26 INFO streaming.StreamJob: map 90% reduce 17%
14/02/05 11:25:27 INFO streaming.StreamJob: map 100% reduce 20%
14/02/05 11:25:28 INFO streaming.StreamJob: map 100% reduce 22%
14/02/05 11:25:31 INFO streaming.StreamJob: map 100% reduce 34%
14/02/05 11:25:32 INFO streaming.StreamJob: map 100% reduce 38%
14/02/05 11:25:33 INFO streaming.StreamJob: map 100% reduce 40%
14/02/05 11:25:34 INFO streaming.StreamJob: map 100% reduce 60%
14/02/05 11:25:35 INFO streaming.StreamJob: map 100% reduce 87%
14/02/05 11:25:36 INFO streaming.StreamJob: map 100% reduce 100%
14/02/05 11:25:37 INFO streaming.StreamJob: Job complete: job_201312100900_1189
14/02/05 11:25:37 INFO streaming.StreamJob: Output: rcv1/small/events
2.59User 0.13system 0:42.14elapsed 6%CPU (0avgtext+0avgdata 314512maxresident)k
0inputs+1080outputs (0major+2550minor)pagefaults 0swaps
To train with streaming Hadoop:

• First, you need to prepare the corpus by splitting it into *shards*
• … and distributing the shards to different machines:

```
$ hadoop fs --help
-help: Unknown command
Usage: java FsShell
    [-ls <path>]
    [-lsr <path>]
    [-du <path>]
    [-dus <path>]
    [-count[-q] <path>]
    [-mv <src> <dst>]
    [-cp <src> <dst>]
    [-rm [-skipTrash] <path>]
    [-rmr [-skipTrash] <path>]
    [-expunge]
    [-put <localsrc> ... <dst>]
    [-copyFromLocal <localsrc> ... <dst>]
    [-moveFromLocal <localsrc> ... <dst>]
    [-get [-ignoreCrc] [-crc] <src> <localdst>]
    [-getmerge <src> <localdst> [addnl]]
```
weak as the arrival of arbitrated cargoes from the West has put the local market under pressure. In Singapore, May swaps fell to $21.30/$21.50 per barrel in late trade on Thursday from $21.70/$21.90 on Wednesday. While in Tokyo, first-half June open-spec naphtha was assessed at $207.00/$208.00 per tonne, compared with late Wednesday's $213.00/$214.00. Added to these factors, traders said that a few petrochemicals were due to go into turnaround in the next few weeks which would further dampen demand. -- Melanie Goodfellow, London Newsroom, +44 171 542 7714.

M14, M143, MCAT  The Brent crude market on the Singapore International Monetary Exchange (SIMEX) will be closed on Friday and Monday, SIMEX officials said on Tuesday. The closure will mark the corresponding closure on Friday and Monday of the Brent market on the International Petroleum Exchange (IPE) in London. SIMEX Brent operates a mutual offset system with the IPE in London, so SIMEX tends to close in line with the U.K. -- Singapore Newsroom (+65 870 3081)
To train with streaming Hadoop:

• One way to shard text:
  – hadoop fs -put LocalFileName HDFSName
  – then run a streaming job with ‘cat’ as mapper and reducer
  – and specify the number of shards you want with option
    -numReduceTasks
To train with streaming Hadoop:

• Next, prepare your code for upload and distribution to the machines cluster

```shell
nb.jar: StreamSumReducer.java StreamNB.java SmallStreamNB.java
javac -d classes StreamSumReducer.java StreamNB.java SmallStreamNB.java
jar -cvf nb.jar -C classes .
```
To train with streaming Hadoop:

• Next, prepare your code for upload and distribution to the machines cluster

```bash
$ javac -d classes StreamSumReducer.java StreamNB.java SmallStreamNB.java
$ jar -cvf nb.jar -C classes .
```
Now you can run streaming Hadoop:

```
STRJAR = /usr/lib/hadoop/contrib/streaming/hadoop-streaming-1.2.0.1.3.0.0-107.jar

small-events-hs:
    hadoop fs -rmdir rcv1/small/events
    time hadoop jar $(STRJAR) \
        -input rcv1/small/sharded -output rcv1/small/events \
        -mapper 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamNB' \
        -reducer 'java -Xmx512m -cp ./lib/nb.jar com.wcohen.StreamSumReducer' \
        -file nb.jar -numReduceTasks 10
```
Map reduce without Hadoop streaming
“Real” Hadoop

• Streaming is simple but
  – There’s no typechecking of inputs/outputs
  – You need to parse strings a lot
  – You can’t use compact binary encodings
  – …
  – basically you have limited control over the messages you’re sending
    • i/o costs = O(message size) often dominates
public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = new Job(conf, "wordcount");

job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);
}
MR Code: Word Count Map

```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws <stuff> {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}
```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
Debugging Map-Reduce
Some common pitfalls

• You have no control over the order in which reduces are performed
• You have no* control over the order in which you encounter reduce values
  • *by default anyway
• The only ordering you should assume is that Reducers always start after Mappers
Some common pitfalls

• You should assume your Maps and Reduces will be taking place on different machines with different memory spaces

• Don’t make a static variable and assume that other processes can read it
  – They can’t.
  – It appear that they can when run locally, but they can’t
  – No really, don’t do this.
Some common pitfalls

• Do not communicate between mappers or between reducers
  • overhead is high
  • you don’t know which mappers/reducers are actually running at any given point
  • there’s no easy way to find out what machine they’re running on
    – because you shouldn’t be looking for them anyway
When mapreduce doesn’t fit

- The beauty of mapreduce is its separability and independence
- If you find yourself trying to communicate between processes
  - you’re doing it wrong
    » or
  - what you’re doing is not a mapreduce
When mapreduce doesn’t fit

• Not everything is a mapreduce
• Sometimes you need more communication
  – We’ll talk about other programming paradigms later
What’s so tricky about MapReduce?

• Really, nothing. It’s easy.
• What’s often tricky is figuring out how to write an algorithm as a series of map-reduce substeps.
  – How and when do you parallelize?
  – When should you even try to do this? when should you use a different model?
Performance

• IMPORTANT
  – You may not have room for all reduce values in memory
    • In fact you should PLAN not to have memory for all values
    • Remember, small machines are much cheaper
      – you have a limited budget
Combiners in Hadoop
Some of this is wasteful

• Remember - moving data around and writing to/reading from disk are very expensive operations

• No reducer can start until:
  • all mappers are done
  • data in its partition has been sorted
# How much does buffering help?

<table>
<thead>
<tr>
<th>BUFFER_SIZE</th>
<th>Time</th>
<th>Message Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
<td>1.7M words</td>
</tr>
<tr>
<td>100</td>
<td>47s</td>
<td>1.2M</td>
</tr>
<tr>
<td>1,000</td>
<td>42s</td>
<td>1.0M</td>
</tr>
<tr>
<td>10,000</td>
<td>30s</td>
<td>0.7M</td>
</tr>
<tr>
<td>100,000</td>
<td>16s</td>
<td>0.24M</td>
</tr>
<tr>
<td>1,000,000</td>
<td>13s</td>
<td>0.16M</td>
</tr>
<tr>
<td>limit</td>
<td></td>
<td>0.05M</td>
</tr>
</tbody>
</table>
Combiners

• Sits between the map and the shuffle
  – Do some of the reducing while you’re waiting for other stuff to happen
  – Avoid moving all of that data over the network

• Only applicable when
  – order of reduce values doesn’t matter
  – effect is cumulative
public static class Combiner extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
Deja vu: Combiner = Reducer

• Often the combiner is the reducer.
  – like for word count
  – but not always

  – remember you have no control over when/whether the combiner is applied