Scaling up: Naïve Bayes on Hadoop

What, How and Why?
Big ML c. 2001 (Banko & Brill, “Scaling to Very Very Large…”, ACL 2001)

Task: distinguish pairs of easily-confused words (“affect” vs “effect”) in context
Some other hot BigData topics

• Scaling up SGD
  – parallelizing the updates
  – asynchronous SGD and “parameter servers”

• Other models for parallelization
  – Graph-based machine learning

• Using less memory
  – Lossy compression: Bloom filters, …

• Fast nearest neighbors
  – Locality sensitive hash functions (LSH), ….
Some other hot BigData topics

• These are all covered in depth in my other course
  – 10-605 Machine Learning from Large Datasets, Tu-Th 1:30-2:50

• Today I’m going to talk about learning in Hadoop
The Naïve Bayes classifier – v2

- Dataset: each example has
  - A unique id $id$
    - Why? For debugging the feature extractor
  - $d$ attributes $X_1, \ldots, X_d$
    - Each $X_i$ takes a discrete value in $\text{dom}(X_i)$
  - One class label $Y$ in $\text{dom}(Y)$
- You have a train dataset and a test dataset
The Naïve Bayes classifier – v2

- You have a train dataset and a test dataset
- Initialize an “event counter” (hashtable) C
- For each example id, y, x_1, …, x_d in train:
  - C(“Y=ANY”) ++;  C(“Y=y”) ++
  - For j in 1..d:
    - C(“Y=y ^ X=x_j”) ++
- For each example id, y, x_1, …, x_d in test:
  - For each y’ in dom(Y):
    - Compute Pr(y’,x_1,…..,x_d) = \left( \prod_{j=1}^{d} \Pr(X = x_j | Y = y') \right) \Pr(Y = y')

= \left( \prod_{j=1}^{d} \frac{\Pr(X = x_j, Y = y')}{\Pr(X = \text{ANY}, Y = y')} \right) \Pr(Y = y')

  - Return the best y’
The Naïve Bayes classifier – v2

• You have a train dataset and a test dataset
• Initialize an “event counter” (hashtable) C
• For each example id, y, x₁,…..,xₐ in train:
  – C("Y=ANY") ++;  C("Y=y") ++
  – For j in 1..d:
    • C("Y=y ^ X=ANYₗ") ++; C("Y=y ^ X=xₗ") ++
• For each example id, y, x₁,…..,xₐ in test:
  – For each y’ in dom(Y):
    • Compute log Pr(y’,x₁,…..,xₐ) =

\[
= \left( \sum_j \log \frac{C(X = xₗ ^ Y = y') + mq_x}{C(X = ANY ^ Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}
\]

  – Return the best y’
Complexity of Naïve Bayes

- You have a *train* dataset and a *test* dataset
- Initialize an “event counter” (hashtable) C
- For each example id, y, x₁,…..,xₙ in *train*:
  - C(“Y=ANY”) ++; C(“Y=y”) ++
  - For j in 1..d:
    - C(“Y=y ^ X=ANY_j”) ++; C(“Y=y ^ X=x_j”) ++
- For each example id, y, x₁,…..,xₙ in *test*:
  - For each y’ in dom(Y):
    - Compute log Pr(y’,x₁,…..,xₙ) =
      \[
      \left( \sum_j \log \frac{C(X = x_j \land Y = y') + mq_x}{C(X = ANY \land Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}
      \]
  - Return the best y’

Sequential reads

Complexity: O(n), n=size of *train*

Complexity: O(|dom(Y)|*n’), n’=size of *test*
Naïve Bayes v2

• This is one example of a streaming classifier
  – Each example is only read only once
  – You can update a classifier after every example and can perform classifications at any point
  – Memory is minimal (<< O(n))
    • Ideally it would be constant
    • Traditionally less than O(sqrt(N))
  – Order doesn’t matter
    • Nice because we may not control the order of examples in real life
    • This is a hard one to get a learning system to have!
• There are few competitive learning methods that as stream-y as naïve Bayes...
Scaling up Naïve Bayes

Next....

What, How and Why?
Complexity of Naïve Bayes

- You have a *train* dataset and a *test* dataset
- Initialize an “event counter” (hashtable) C
- For each example id, y, x₁,…..,xₙ in *train*:
  - C(“Y=ANY”) ++;  C(“Y=y”) ++
  - For j in 1..d:
    • C(“Y=y ^ X=ANY”) ++; C(“Y=y ^ X=x_j”) ++
- For each example id, y, x₁,…..,xₙ in *test*:
  - For each y’ in dom(Y):
    • Compute log Pr(y’,x₁,…..,xₙ) =
      \[
      \sum_j \log \frac{C(X = x_j ^ Y = y') + mq_x}{C(X = ANY ^ Y = y') + m} + \log \frac{C(Y = y') + mq_y}{C(Y = ANY) + m}
      \]
  - Return the best y’
Big ML c. 2001 (Banko & Brill, “Scaling to Very Very Large…”, ACL 2001)

Task: distinguish pairs of easily-confused words (“affect” vs “effect”) in context
What’s next

• How to implement Naïve Bayes
  – Assuming the event counters do not fit in memory
  – This is a core problem in big data processing

• Large-scale text classification challenge: O(1,000,000) documents, O(100,000) categories.
• Or, classify web pages as product, nonProduct, product.review, product.offer, …

• Disk-based database
• Memory-based distributed database
• “Stream-and-sort” design pattern ➔ Hadoop
What’s next

• How to implement Naïve Bayes
  – Assuming the event counters do not fit in memory
• Possible approaches:
  – Use a database? (or at least a key-value store)
Who?

- Compilers don’t warn Jeff Dean. Jeff Dean warns compilers.
- Jeff Dean builds his code before committing it, but only to check for compiler and linker bugs.
- Jeff Dean writes directly in binary. He then writes the source code as a documentation for other developers.
- Jeff Dean once shifted a bit so hard, it ended up on another computer.
- When Jeff Dean has an ergonomic evaluation, it is for the protection of his keyboard.
- gcc -O4 emails your code to Jeff Dean for a rewrite.
- When he heard that Jeff Dean's autobiography would be exclusive to the platform, Richard Stallman bought a Kindle.
- Jeff Dean puts his pants on one leg at a time, but if he had more legs, you’d realize the algorithm is actually only $O(\log n)$.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>10,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>

100k sec/day, 100 hr/decade
A single large file can be spread out among many non-adjacent blocks/sectors...

and then you need to seek around to scan the contents of the file...

Question: What could you do to reduce this cost?
Distributed Counting

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

Counting logic

Hash table 1

Machine 1

Hash table 2

Machine 2

Machine K

New issues:
- Machines and memory cost $$!
- Routing increment requests to right machine
- Sending increment requests across the network
- Communication complexity
# Numbers (Jeff Dean says) Everyone Should Know

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>10,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>

- L1 cache reference is 10 times faster than L2 cache reference.
- Mutex lock/unlock is 15 times slower than a branch mispredict.
- Compress 1K bytes with Zippy is 40 times slower than a branch mispredict.
- Sending packets across the world is 100,000 times slower than a branch mispredict.
What’s next

• How to implement Naïve Bayes
  – Assuming the event counters do not fit in memory
• Possible approaches:
  – Use a memory-based distributed database?
    • Extra cost: Communication costs: $O(n)$ … but that’s “ok”
    • Extra complexity: routing requests correctly
  – Compress the counter hash table?
    • Use integers as keys instead of strings?
    • Use approximate counts?
    • Discard infrequent/unhelpful words?
  – Trade off time for space somehow?
    • Observation: if the counter updates were better-ordered we could avoid using disk

The complexity of using a memory-based distributed database is $O(n*\text{scan}) \rightarrow O(n*\text{scan}+n*\text{send})$.
Large-vocabulary Naïve Bayes Counting

• One way trade off time for space:
  – Assume you need $K$ times as much memory as you actually have
  – Method:
    • Construct a hash function $h(event)$
    • For $i=0,\ldots,K-1$:
      – Scan thru the train dataset
      – Increment counters for event only if $h(event) \mod K == i$
      – Save this counter set to disk at the end of the scan
    • After $K$ scans you have a complete counter set
  • Comment:
    – this works for any counting task, not just naïve Bayes
    – What we’re really doing here is organizing our “messages” to get more locality….
Large vocabulary counting

• **Another approach:**
  – Start with
    • **Q:** “what can we do for large sets quickly”?
    • **A:** sorting
      – It’s $O(n \log n)$, not much worse than linear
      – You can do it for very large datasets using a **merge sort**
        » sort $k$ subsets that fit in memory,
        » merge results, which can be done in linear time
Large-vocabulary Naïve Bayes

- Create a hashtable $C$
- For each example $id, y, x_1, \ldots, x_d$ in $train$:
  - $C(\text{"Y=ANY"})$ $$; C(\text{"Y=y"})$$ $$
  - For $j$ in $1..d$:
    - $C(\text{"Y=y ^ X=x_j"})$ $$
Large-vocabulary Naïve Bayes

- Create a hashtable C
- For each example \( id, y, x_1, \ldots, x_d \) in train:
  - \( C("Y=ANY") += 1; \quad C("Y=y") += 1 \)
  - Print “Y=ANY += 1”
  - Print “Y=y += 1”
  - For \( j \) in 1..d:
    - \( C("Y=y \land X=x_j") += 1 \)
    - Print “Y=y \land X=x_j += 1”
- Sort the event-counter update “messages”
- Scan the sorted messages and compute and output the final counter values

```
java MyTrainer train | sort | java MyCountAdder > model
```
Large-vocabulary Naïve Bayes

• Create a hashtable $C$

• For each example $id, y, x_1, \ldots, x_d$ in train:
  - $C(\text{"Y=ANY"})$ += 1; $C(\text{"Y=y"})$ +=
  - Print “Y=ANY += 1”
  - Print “Y=y += 1”
  - For $j$ in 1..$d$:
    - $C(\text{"Y=y ^ X=x_j"})$ +=
    - Print “Y=y ^ X=x_j += 1”

• Sort the event-counter update “messages”
  - We’re collecting together messages about the same counter

• Scan and add the sorted messages and output the final counter values

Y=business    += 1
Y=business    += 1
...            
Y=business ^ X=aaa += 1
...            
Y=business ^ X=zynga += 1
Y=sports ^ X=hat  += 1
Y=sports ^ X=hockey += 1
Y=sports ^ X=hockey += 1
...            
Y=sports ^ X=hoe  += 1
...            
Y=sports      += 1
...            
Y=sports      += 1
...
Large-vocabulary Naïve Bayes

Scan-and-add:

\[
\begin{align*}
Y &= \text{business} & + &= 1 \\
Y &= \text{business} & + &= 1 \\
\cdots \\
Y &= \text{business} \land X &= \text{aaa} & + &= 1 \\
\cdots \\
Y &= \text{business} \land X &= \text{zynga} & + &= 1 \\
Y &= \text{sports} \land X &= \text{hat} & + &= 1 \\
Y &= \text{sports} \land X &= \text{hockey} & + &= 1 \\
Y &= \text{sports} \land X &= \text{hockey} & + &= 1 \\
\cdots \\
Y &= \text{sports} \land X &= \text{hoe} & + &= 1 \\
\cdots \\
Y &= \text{sports} & + &= 1 \\
\cdots 
\end{align*}
\]

\[
\begin{align*}
\text{• previousKey} &= \text{Null} \\
\text{• sumForPreviousKey} &= 0 \\
\text{• For each (event, delta) in input:} \\
\quad \text{• If event} &= \text{previousKey} \\
\quad \quad \text{• sumForPreviousKey} &= \text{+ delta} \\
\quad \text{• Else} \\
\quad \quad \text{• OutputPreviousKey()} \\
\quad \quad \text{• previousKey} &= \text{event} \\
\quad \quad \text{• sumForPreviousKey} &= \text{delta} \\
\text{• OutputPreviousKey()} \\
\end{align*}
\]

\[
\text{define OutputPreviousKey():} \\
\begin{align*}
\text{• If PreviousKey}!&=\text{Null} \\
\quad \text{• print PreviousKey,sumForPreviousKey} \\
\end{align*}
\]

Accumulating the event counts requires constant storage … as long as the input is sorted.
Distributed Counting → Stream and Sort Counting

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

Counting logic

Message-routing logic

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

Machine 0

Machine 1

Machine 2

... 

Machine K
Distributed Counting → Stream and Sort Counting

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

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

Machine A

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

Logic to combine counter updates

Machine C

Machine B
Stream and Sort Counting $\Rightarrow$ Distributed Counting

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

Counting logic

Machines A1, ...

"C[x] += D"

Sort

Standardized message routing logic

Machines B1, ...

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

Logic to combine counter updates

Machines C1, ...

Trivial to parallelize!

Easy to parallelize!
Locality is good

| Micro: 0.6G memory | Standard: S: 1.7Gb L: 7.5Gb XL: 15Mb | Hi Memory: XXL: 34.2 XXXXL: 68.4 |

<table>
<thead>
<tr>
<th>Region: US East (Virginia)</th>
<th>Linux/UNIX Usage</th>
<th>Windows Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard On-Demand Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small (Default)</td>
<td>$0.085 per hour</td>
<td>$0.12 per hour</td>
</tr>
<tr>
<td>Large</td>
<td>$0.34 per hour</td>
<td>$0.48 per hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.68 per hour</td>
<td>$0.96 per hour</td>
</tr>
<tr>
<td><strong>Micro On-Demand Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro</td>
<td>$0.02 per hour</td>
<td>$0.03 per hour</td>
</tr>
<tr>
<td><strong>Hi-Memory On-Demand Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.50 per hour</td>
<td>$0.62 per hour</td>
</tr>
<tr>
<td>Double Extra Large</td>
<td>$1.00 per hour</td>
<td>$1.24 per hour</td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$2.00 per hour</td>
<td>$2.48 per hour</td>
</tr>
<tr>
<td><strong>Hi-CPU On-Demand Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>$0.17 per hour</td>
<td>$0.29 per hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.68 per hour</td>
<td>$1.16 per hour</td>
</tr>
<tr>
<td><strong>Cluster Compute Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$1.30 per hour</td>
<td>$1.61 per hour</td>
</tr>
<tr>
<td>Eight Extra Large</td>
<td>$2.40 per hour</td>
<td>$2.97 per hour</td>
</tr>
<tr>
<td><strong>Cluster GPU Instances</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$2.10 per hour</td>
<td>$2.60 per hour</td>
</tr>
</tbody>
</table>
Large-vocabulary Naïve Bayes

• For each example id, y, x₁, …., x₉ in train:
  – Print Y=ANY += 1
  – Print Y=y += 1
  – For j in 1..d:
    • Print \( Y=y \land X=x_j += 1 \)
• Sort the event-counter update “messages”
• Scan and add the sorted messages and output the final counter values

java MyTrainer train | sort | java MyCountAdder > model

Model size: max \( O(n) \), \( O(|V| \vert \text{dom}(Y) \vert) \)
Optimizations

java MyTrainer > train | sort | java MyCountAdder > model

O(n)
Input size=n
Output size=n

O(nlogn)
Input size=n
Output size=n

O(n)
Input size=n
Output size=m

m<<n ... say O(sqrt(n))

A useful optimization:

decrease the size of
the input to the sort

Reduces the size from
O(n) to O(m)

1. Compress the output by using simpler messages
   ("C[event] += 1") → "event 1"

2. Compress the output more – e.g. string → integer code
   Tradeoff – ease of debugging vs efficiency – are messages
   meaningful or meaningful in context?
Optimization...

• For each example id, y, x₁,…..,xₐ in \textit{train}:
  – Print ”Y=\text{ANY} += 1”
  – Print ”Y=y += 1”
  – For j in 1..d:
    • Print ”Y=y \land X=x_j += 1”
• Sort the event-counter update “messages”
• Scan and add the sorted messages and output the final counter values

• Initialize hashtable C
• For each example id, y, x₁,…..,xₐ in \textit{train}:
  – C[Y=\text{ANY}] += 1
  – C[Y=y] += 1
  – For j in 1..d:
    • C[Y=y \land X=x_j] += 1
• If memory is getting full: output all values from C as messages and re-initialize C
• Sort the event-counter update “messages”
• Scan and add the sorted messages

\texttt{java MyTrainer train | sort | java MyCountAdder > model}
Scaling up Naïve Bayes

Sort of….

DONE!

What, How and Why?

This “scales” but it’s pretty slow

Can we parallelize this process?
Optimizations

java MyTrainer | sort | java MyCountAdder > model

A useful optimization:

decrease the size of the input to the sort

O(n)  
Input size=n  
Output size=n

O(nlogn)  
Input size=n  
Output size=n

O(n)  
Input size=n  
Output size=m

m<<n … say O(sqrt(n))

Reduces the size from O(n) to O(m)

1. Compress the output by using simpler messages
   (“C[event] += 1”) ➞ “event 1”

2. Compress the output more – e.g. string ➞ integer code
   Tradeoff – ease of debugging vs efficiency – are messages meaningful or meaningful in context?
INADVISABLE SCIENCE

THE ASSIGNMENT THAT NEVER EXISTED

A HORROR that could ONLY be IMAGINED!

CREATED WITH PULP-O-MIZER COVER MAKER
Write code to run assignment 2 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 2 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 2 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 2 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

- 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’

---

Step 2: run the maps 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 < ....> ...’ 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)]

Step 3: redistribute the map output

• cat input.txt | MAP | sort | REDUCE > output.txt
How would you run assignment 2 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 …
- 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)]

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

Step 3: redistribute the map output
How would you run assignment 2 in parallel?

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

Step 4: run the reduce processes in parallel
How would you run assignment 2 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
  - 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 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, ...
Also: it’s been done.

It’s Hadoop.
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 arbitrated cargoes from the West has put the local market under pressure…
M14,M143,MCAT The Brent crude market on the Singapore International …
MR Overview

The diagram illustrates the workflow of a MapReduce (MR) job, which consists of two main phases: Map and Reduce.

1. **User Program**: The user program initiates the process by creating a Master process.
2. **Master**: The Master assigns tasks to the Worker processes.
3. **Worker**: Each Worker receives tasks and executes them. Tasks can include reading from input files, processing data, and writing intermediate results.

**Map Phase**:
- **Split 0**, **Split 1**, **Split 2**, **Split 3**, **Split 4** are input files.
- Worker processes read data from input files.

**Intermediate Files**: Intermediate results are written to local disks.

**Reduce Phase**:
- Workers perform Reduce operations on intermediate files.
- Results are written to output files.

The diagram highlights the steps and the flow of data through the MR framework.
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>0%</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>
</table>
| task_201301231150_0778_m_000000 | 0.00% |        | 30-Jan-2013 11:47:01 | 30-Jan-2013 11:47:25 (24sec) | java.lang.RuntimeException: PipeMapReduceException
at org.apache.hadoop.streaming.StreamJob.runPipeline
at org.apache.hadoop.streaming.StreamJob.runJob
at org.apache.hadoop.streaming.StreamJob.runPipeline
at org.apache.hadoop.streaming.BlockConsumer.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
at org.apache.hadoop.streaming.PipeMapReduce.runPipeline
at org.apache.hadoop.streaming.PipeMapReduce.runJob
<p>|</p>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:311)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:540)</td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:57)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.streaming.PipeMapRunner.run(PipeMapRunner.java:36)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:358)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.mapred.MapTask.run(MapTask.java:307)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at org.apache.hadoop.mapred.Child.main(Child.java:170)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 work 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, ….
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);
}
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));
    }
}
Is any part of this 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
Optimizations

java MyTrainer | sort | java MyCountAdder > model

- O(n)
  - Input size=n
  - Output size=n

- O(nlogn)
  - Input size=n
  - Output size=n

- O(n)
  - Input size=n
  - Output size=m

  \[m \ll n \ldots \text{say } O(\sqrt{n})\]

A useful optimization:
decrease the size of the input to the sort

Reduces the size from O(n) to O(m)

1. Compress the output by using simpler messages
   ("C[event] += 1") \(\Rightarrow\) "event 1"

2. Compress the output more – e.g. string \(\Rightarrow\) integer code
   Tradeoff – ease of debugging vs efficiency – are messages meaningful or meaningful in context?
Optimization...

- For each example \( id, y, x_1, \ldots, x_d \) in \( train \):
  - Print “\( Y=ANY \) += 1”
  - Print “\( Y=y \) += 1”
  - For \( j \) in \( 1..d \):
    - Print “\( Y=y ^ X=x_j \) += 1”
- Sort the event-counter update “messages”
- Scan and add the sorted messages and output the final counter values

- Initialize hashtable \( C \)
- For each example \( id, y, x_1, \ldots, x_d \) in \( train \):
  - \( C[Y=ANY] += 1 \)
  - \( C[Y=y] += 1 \)
  - For \( j \) in \( 1..d \):
    - \( C[Y=y ^ X=x_j] += 1 \)
- If memory is getting full: output all values from \( C \) as messages and re-initialize \( C \)
- Sort the event-counter update “messages”
- Scan and add the sorted messages

```
java MyTrainer train | sort | java MyCountAdder > model
```
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
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
  
  • More on this later

• 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
What’s so tricky about MapReduce?

• Really, nothing. It’s easy.
• Some things are tricky:
  • Dealing with the one-in-a-million weirdo problems
  • Debugging on a cluster
• 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?
A More Complex Algorithm in Map-Reduce

(almost)
Flaw: Large-vocabulary Naïve Bayes is Expensive to Use

• For each example \( id, y, x_1, \ldots, x_d \) in test:
• Sort the event-counter update “messages”
• Scan and add the sorted messages and output the final counter values

\[
\text{Model size: } \max O(n), O(|V| \cdot |\text{dom}(Y)|)
\]

• For each example \( id, y, x_1, \ldots, x_d \) in test:
  – For each \( y' \) in \( \text{dom}(Y) \):
    • Compute \( \log \Pr(y',x_1,\ldots,x_d) = \)

\[
= \left( \sum_j \log \frac{C(X = x_j \land Y = y') + mq_x}{C(X = \text{ANY} \land Y = y') + m} \right) + \log \frac{C(Y = y') + mq_y}{C(Y = \text{ANY}) + m}
\]
## Can we do better?

### Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w₁,₁ w₁,₂ w₁,₃ ⋯ w₁,k₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>id₂</td>
<td>w₂,₁ w₂,₂ w₂,₃ ⋯</td>
</tr>
<tr>
<td>id₃</td>
<td>w₃,₁ w₃,₂ ⋯</td>
</tr>
<tr>
<td>id₄</td>
<td>w₄,₁ w₄,₂ ⋯</td>
</tr>
<tr>
<td>id₅</td>
<td>w₅,₁ w₅,₂ ⋯</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### Event counts

<table>
<thead>
<tr>
<th>X=w₁,Y=sports</th>
<th>5245</th>
</tr>
</thead>
<tbody>
<tr>
<td>X=w₁,Y=worldNews</td>
<td>1054</td>
</tr>
<tr>
<td>X=..</td>
<td>2120</td>
</tr>
<tr>
<td>X=w₂,Y=...</td>
<td>37</td>
</tr>
<tr>
<td>X=...</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

### What we’d like

<table>
<thead>
<tr>
<th>id₁</th>
<th>w₁,₁ w₁,₂ w₁,₃ ⋯ w₁,k₁</th>
<th>C[X=w₁,₁,Y=sports]=5245, C[X=w₁,₁,Y=..], C[X=w₁,₂,...]</th>
</tr>
</thead>
<tbody>
<tr>
<td>id₂</td>
<td>w₂,₁ w₂,₂ w₂,₃ ⋯</td>
<td>C[X=w₂,₁,Y=...]=1054,..., C[X=w₂,₂,...]</td>
</tr>
<tr>
<td>id₃</td>
<td>w₃,₁ w₃,₂ ⋯</td>
<td>C[X=w₃,₁,Y=...]=...</td>
</tr>
<tr>
<td>id₄</td>
<td>w₄,₁ w₄,₂ ⋯</td>
<td>...</td>
</tr>
</tbody>
</table>
Can we do better?

Step 1: group counters by word $w$

How:
- Stream and sort:
  - for each $C[X=w^Y=y]=n$
  - print “$w$ $C[Y=y]=n$”
- sort and build a list of values associated with each key $w$

Event counts

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>sports</td>
<td>5245</td>
</tr>
<tr>
<td>$w_1$</td>
<td>worldNews</td>
<td>1054</td>
</tr>
<tr>
<td>$.$</td>
<td></td>
<td>2120</td>
</tr>
<tr>
<td>$w_2$</td>
<td>$\ldots$</td>
<td>37</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>$\ldots$</td>
<td></td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$w$</th>
<th>Counts associated with $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>$C[w^Y=sports]=2$</td>
</tr>
<tr>
<td>agent</td>
<td>$C[w^Y=sports]=1027, C[w^Y=worldNews]=564$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>zynga</td>
<td>$C[w^Y=sports]=21, C[w^Y=worldNews]=4464$</td>
</tr>
</tbody>
</table>
If these records were in a key-value DB we would know what to do….

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_1,1</th>
<th>w_1,2</th>
<th>w_1,3</th>
<th>…</th>
<th>w_1,k_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_2</td>
<td>w_2,1</td>
<td>w_2,2</td>
<td>w_2,3</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>id_3</td>
<td>w_3,1</td>
<td>w_3,2</td>
<td>…</td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>id_4</td>
<td>w_4,1</td>
<td>w_4,2</td>
<td>…</td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>id_5</td>
<td>w_5,1</td>
<td>w_5,2</td>
<td>…</td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027, C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21, C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Step 2: stream through and for each test case

\[ id_i \ w_{i,1} \ w_{i,2} \ w_{i,3} \ \ldots \ \ w_{i,k_i} \]

request the event counters needed to classify \( id_i \) from the event-count DB, then classify using the answers
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1} w_{1,2} w_{1,3} \ldots w_{1,k_1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>id_2</td>
<td>w_{2,1} w_{2,2} w_{2,3} \ldots</td>
</tr>
<tr>
<td>id_3</td>
<td>w_{3,1} w_{3,2} \ldots</td>
</tr>
<tr>
<td>id_4</td>
<td>w_{4,1} w_{4,2} \ldots</td>
</tr>
<tr>
<td>id_5</td>
<td>w_{5,1} w_{5,2} \ldots</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027,C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21,C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Step 2: stream through and for each test case

\textit{id}_i \ w_{i,1} w_{i,2} w_{i,3} \ldots w_{i,k_i}

\textbf{request} the event counters needed to classify \textit{id}_i from the event-count DB, then classify using the \textbf{answers}
Recall: Stream and Sort Counting: sort messages so the recipient can stream through them

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

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

Machine A

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

Logic to combine counter updates

Machine C

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

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

Machine B
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

<table>
<thead>
<tr>
<th>id</th>
<th>w_1,1</th>
<th>w_1,2</th>
<th>w_1,3</th>
<th>...</th>
<th>w_1,k1</th>
</tr>
</thead>
<tbody>
<tr>
<td>id2</td>
<td>w_2,1</td>
<td>w_2,2</td>
<td>w_2,3</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>id3</td>
<td>w_3,1</td>
<td>w_3,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id4</td>
<td>w_4,1</td>
<td>w_4,2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id5</td>
<td>w_5,1</td>
<td>w_5,2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=1027, C[w^Y=worldNews]=564</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>C[w^Y=sports]=21, C[w^Y=worldNews]=4464</td>
</tr>
</tbody>
</table>

Classification logic

W_{1,1} counters to id_1
W_{1,2} counters to id_2
...
W_{i,j} counters to id_i
...
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

$id_1$ found an aardvark in zynga’s farmville today!
$id_2$ ...
$id_3$ ....
$id_4$ ...
$id_5$ ...
...

Record of all event counts for each word

<table>
<thead>
<tr>
<th>$w$</th>
<th>Counts associated with $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>$C[w^Y=\text{sports}]=2$</td>
</tr>
<tr>
<td>agent</td>
<td>$C[w^Y=\text{sports}]=1027,C[w^Y=\text{worldNews}]=564$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>$C[w^Y=\text{sports}]=21,C[w^Y=\text{worldNews}]=4464$</td>
</tr>
</tbody>
</table>

Classification logic

found ctrs to $id_1$
aardvark ctrs to $id_1$
... ctrs to $id_1$
today ctrs to $id_1$
...
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data

$id_1$ found an aardvark in zynga’s farmville today!
$id_2$ ...
$id_3$ ....
$id_4$ ...
$id_5$ ...

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts associated with W</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>$C[w^Y=sports]=2$</td>
</tr>
<tr>
<td>agent</td>
<td>$C[w^Y=sports]=1027, C[w^Y=worldNews]=564$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>$C[w^Y=sports]=21, C[w^Y=worldNews]=4464$</td>
</tr>
</tbody>
</table>

~ is the last ascii character means that it will sort after anything else with unix sort
Is there a stream-and-sort analog of this request-and-answer pattern?

Test data:

\[ id_1 \text{ found an aardvark in zynga’s farmville today!} \]
\[ id_2 \ldots \]
\[ id_3 \ldots \]
\[ id_4 \ldots \]
\[ id_5 \ldots \]

Record of all event counts for each word:

<table>
<thead>
<tr>
<th>( w )</th>
<th>Counts associated with ( W )</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>( C[w^Y=\text{sports}]=2 )</td>
</tr>
<tr>
<td>agent</td>
<td>( C[w^Y=\text{sports}]=1027, C[w^Y=\text{worldNews}]=564 )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>( C[w^Y=\text{sports}]=21, C[w^Y=\text{worldNews}]=4464 )</td>
</tr>
</tbody>
</table>

Classification logic:

- found \( \sim \text{ctr to } id_1 \)
- aardvark \( \sim \text{ctr to } id_2 \)
- ... \( \sim \text{ctr to } id_i \)
- today \( \sim \text{ctr to } id_i \)

Counter records:

Combine and sort.
A stream-and-sort analog of the request-and-answer pattern…

Record of all event counts for each word

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Counter records

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id34742</td>
</tr>
</tbody>
</table>

Combine and sort requests

Request-handling logic
A stream-and-sort analog of the request-and-answer pattern...

- `previousKey = somethingImpossible`
- For each `(key,val)` in input:
  - If `key==previousKey`
    - `Answer(recordForPrevKey,val)`
  - Else
    - `previousKey = key`
    - `recordForPrevKey = val`

Define `Answer(record,request)`:  
- `find id` where “`request = ~ctr to id`”
- `print “id ~ctr for request is record”`

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Combined and sorted requests

Request-handling logic
A stream-and-sort analog of the request-and-answer pattern…

• previousKey = somethingImpossible
• For each (key,val) in input:
  • If key==previousKey
    • Answer(recordForPrevKey,val)
  • Else
    • previousKey = key
    • recordForPrevKey = val

Define Answer(record,request):
• find id where “request = ~ctr to id”
• print “id ~ctr for request is record”

Output:
id1 ~ctr for aardvark is C[w^Y=sports]=2
...
id1 ~ctr for zynga is ...
...

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>aardvark</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Combine and sort requests

Request-handling logic
A stream-and-sort analog of the request-and-answer pattern...

<table>
<thead>
<tr>
<th>w</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>aardvark</td>
<td>C[w^Y=sports]=2</td>
</tr>
<tr>
<td>aardvark</td>
<td>~ctr to id1</td>
</tr>
<tr>
<td>agent</td>
<td>C[w^Y=sports]=...</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id9854</td>
</tr>
<tr>
<td>...</td>
<td>~ctr to id345</td>
</tr>
<tr>
<td>agent</td>
<td>~ctr to id34742</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>zynga</td>
<td>C[...]</td>
</tr>
<tr>
<td>zynga</td>
<td>~ctr to id1</td>
</tr>
</tbody>
</table>

Output:

id1 ~ctr for aardvark is C[w^Y=sports]=2
...
id1 ~ctr for zynga is ....
...

id1 found an aardvark in zynga’s farmville today!
id2 ...
id3 ....
id4 ...
id5 ...
...

Request-handling logic → Combine and sort → ????
### What we’d wanted

<table>
<thead>
<tr>
<th>id</th>
<th>w_{1,1} w_{1,2} w_{1,3} .... w_{1,k1}</th>
<th>(C[X=w_{1,1}^Y=\text{sports}]=5245, C[X=w_{1,1}^Y=..], C[X=w_{1,2}^Y=..])</th>
</tr>
</thead>
<tbody>
<tr>
<td>id2</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
<td>(C[X=w_{2,1}^Y=..]=1054, ..., C[X=w_{2,k2}^Y=..])</td>
</tr>
<tr>
<td>id3</td>
<td>w_{3,1} w_{3,2} ....</td>
<td>(C[X=w_{3,1}^Y=..]=..)</td>
</tr>
<tr>
<td>id4</td>
<td>w_{4,1} w_{4,2} ...</td>
<td>...</td>
</tr>
</tbody>
</table>

### What we ended up with

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>found aardvark zynga farmville today</td>
</tr>
<tr>
<td></td>
<td>(~\text{ctr for aardvark is } C[w^Y=\text{sports}]=2)</td>
</tr>
<tr>
<td></td>
<td>(~\text{ctr for found is } C[w^Y=\text{sports}]=1027, C[w^Y=\text{worldNews}]=564)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>id2</td>
<td>w_{2,1} w_{2,2} w_{2,3} ....</td>
</tr>
<tr>
<td></td>
<td>(~\text{ctr for } w_{2,1} \text{ is } \ldots)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Wrapup: (1) Hadoop

• Parallelizes the process of streaming through data
• It’s great for:
  – counting and indexing
  – building datasets from: transaction logs, text corpora, …
  – performing feature extraction, feature transformation, combining different feature sets, …
• It’s not great for looping over data
  – Extensions: Spark, HaLoop, …
Wrapup: (2) Hadoop != Big Data

• Scaling up SGD
  – parallelizing the updates
  – asynchronous SGD and “parameter servers”
• Other models for parallelization
  – Graph-based machine learning
• Using less memory
  – Lossy compression: Bloom filters, …
• Fast nearest neighbors
  – Locality sensitive hash functions (LSH), …
Wrapup: (3) MapReduce != Hadoop

• Lots of higher-level functionality available to simplify your interaction with MapReduce
  – PIG query language
  – HiveQL, Pipes, dumbo, …

• My opinion:
  – these are useful *only if* you understand the full software stack pretty well