Introduction to Large Scale Data Mining

Based on ICDM 2009 Tutorial, from:
Spiros Papadimitriou, IBM Research
Jimeng Sun, IBM Research
Rong Yan, Facebook

Summarized version by Xiao Yu
Contents

- Introduction to MapReduce
- Large Scale Data Mining Algorithms
- Large Scale Data Mining Applications
- Summary
Data Everywhere

- Flickr (3 billion photos)
- YouTube (83M videos, 15 hrs/min)
- Web (10B videos watched / mo.)
- Digital photos (500 billion / year)
- All broadcast (70,000TB / year)
- Yahoo! Webmap (3 trillion links, 300TB compressed, 5PB disk)
- Human genome (2-30TB uncomp.)

So what?
Data Everywhere

Opportunities
- Real-time access to content
- Richer context from users and hyperlinks
- Abundant training examples
- Brute-force methods may suffice

Challenges
- Dirtier data
- Efficient algorithms
- Scalability (with reasonable cost)
Any solution?

- Many tasks: Process lots of data to produce other data
- Want to use hundreds or thousands of CPUs
  - ... but this needs to be easy

- MapReduce provides:
  - Automatic parallelization and distribution
  - Fault-tolerance
  - I/O scheduling
  - Status and monitoring

“All models are wrong, but some are useful” — George Box
Programming Model of MapReduce

- **map** \((\text{in\_key}, \text{in\_value}) \rightarrow \text{list(\text{out\_key}, \text{intermediate\_value})}\)**
  - Processes input key/value pair
  - Produces set of intermediate pairs

- **reduce** \((\text{out\_key}, \text{list(\text{intermediate\_value})}) \rightarrow \text{list(\text{out\_value})}\)**
  - Combines all intermediate values for a particular key
  - Produces a set of merged output values (usually just one)

- Inspired by similar primitives in LISP and other languages

Dean & Ghemawat, OSDI"04
Example on Programming Model

Q: “What is the frequency of each first name?”

```
employees.txt

<table>
<thead>
<tr>
<th>#</th>
<th>LAST</th>
<th>FIRST</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Smith</td>
<td>John</td>
<td>$90,000</td>
</tr>
<tr>
<td>2</td>
<td>Brown</td>
<td>David</td>
<td>$70,000</td>
</tr>
<tr>
<td>3</td>
<td>Johnson</td>
<td>George</td>
<td>$95,000</td>
</tr>
<tr>
<td>4</td>
<td>Yates</td>
<td>John</td>
<td>$80,000</td>
</tr>
<tr>
<td>5</td>
<td>Miller</td>
<td>Bill</td>
<td>$65,000</td>
</tr>
<tr>
<td>6</td>
<td>Moore</td>
<td>Jack</td>
<td>$85,000</td>
</tr>
<tr>
<td>7</td>
<td>Taylor</td>
<td>Fred</td>
<td>$75,000</td>
</tr>
<tr>
<td>8</td>
<td>Smith</td>
<td>David</td>
<td>$80,000</td>
</tr>
<tr>
<td>9</td>
<td>Harris</td>
<td>John</td>
<td>$90,000</td>
</tr>
</tbody>
</table>
```

**mapper**
```
def getName (line):
    return (line.split('\t')[1], 1)
```

**reducer**
```
def addCounts (hist, (name, c)):
    hist[name] = \
    hist.get(name, default=0) + c
    return hist
```

```
input = open('employees.txt', 'r')
intermediate = map(getName, input)
result = reduce(addCounts, \n    intermediate, { })
```

Key-value iterators
Recap

<table>
<thead>
<tr>
<th>Quick-n-dirty script</th>
<th>Hadoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>~5 lines of (non-boilerplate) code</td>
<td>Up to <em>thousands</em> of machines and drives</td>
</tr>
<tr>
<td>Single machine, local drive</td>
<td></td>
</tr>
</tbody>
</table>

- What is hidden to achieve this:
  - Data partitioning, placement and replication
  - Computation placement (and replication)
  - Number of nodes (mappers / reducers)
Execution model: Flow

Input file

- Smith John $90,000
  - SPLIT 0
- Yates John $80,000
  - SPLIT 1
  - SPLIT 2
  - SPLIT 3

Mapper

- John 1
- John 2

Reducer

- John 1

Output file

- PART 0
- PART 1

Key/value iterators

Sort-merge

All-to-all, hash partitioning

Sequential scan
Execution model: Placement

HOST 0
- SPLIT 0 Replica 1/3
- SPLIT 1 Replica 2/3
- SPLIT 3 Replica 2/3
- Mapper
- Reducer

HOST 1
- SPLIT 0 Replica 2/3
- SPLIT 3 Replica 1/3
- Mapper

HOST 2
- SPLIT 2 Replica 2/3
- SPLIT 3 Replica 3/3
- Mapper

HOST 3
- SPLIT 2 Replica 3/3
- SPLIT 0 Replica 3/3
- Mapper

HOST 4

HOST 5

HOST 6

COMBINER
Hadoop's stated mission (Doug Cutting interview):
  ◦ Commoditize infrastructure for web-scale, data-intensive applications

Users:
  ◦ Yahoo!
  ◦ Facebook
  ◦ Last.fm
  ◦ Rackspace
  ◦ Digg
  ◦ Apache Nutch
Hadoop Architecture

Interact with apps and users

Distribution execution, storage and coordination service

Filesystems and IO. Cross language support
Contents

- Introduction to MapReduce and Hadoop
- Large Scale Data Mining Algorithms
- Large Scale Data Mining Applications
- Summary
Mining using MapReduce

- Information retrieval
- Graph algorithms: PageRank
- Clustering: Canopy clustering, KMeans
- Classification: kNN, Naive Bayes
IR: Distributed Grep

- Find the doc_id and line# of a matching pattern
- Map: (id, doc)→list(id, line#)
- Reduce: None
IR: URL Access Frequency

- Map: (null, log) \rightarrow (URL, 1)
- Reduce: (URL, 1) \rightarrow (URL, total_count)
**IR: Inverted Index**

- **Map**: (id, doc) → list(word, id)
- **Reduce**: (word, list(id)) → (word, list(id))

![Diagram showing the process of Map and Reduce operations in IR: Inverted Index](image)

1. **Doc**: Represents the input documents.
2. **Map1**: For the first document, maps to <w1, 1>.
3. **Map2**: For the second document, maps to <w2, 2> and <w3, 3>.
4. **Map3**: For the third document, maps to <w1, 5>.

This diagram illustrates the flow of data through the Map and Reduce processes, demonstrating how each document's words are processed and aggregated at the Reduce stage.
Mining using MapReduce

- Information retrieval
- Graph algorithms: PageRank
- Clustering: Canopy clustering, KMeans
- Classification: kNN, Naive Bayes
PageRank

- PageRank vector $\mathbf{q}$ is defined as
  \[ \mathbf{q} = c\mathbf{A}^T\mathbf{q} + \frac{1-c}{N}\mathbf{e} \]
  - $\mathbf{A}$ is the source-by-destination adjacency matrix,
  - $\mathbf{e}$ is all one vector.
  - $N$ is the number of nodes
  - $c$ is the weight between 0 and 1 (e.g. 0.85)

- PageRank indicates the importance of a page.
- Algorithm: Iterative powering for finding the first eigen-vector

\[
\mathbf{A} = \begin{pmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
\end{pmatrix}
\]
MapReduce: PageRank

Map: distribute PageRank $q_i$

Reduce: update new PageRank

PageRank Map()
- Input: key = page $x$, value = (PageRank $q_x$, links[$y_1 ... y_m$])
- Output: key = page $x$, value = $partial_x$
  1. Emit($x$, 0) //guarantee all pages will be emitted
  2. For each outgoing link $y_i$:
     - Emit($y_i$, $q_x$/m)

PageRank Reduce()
- Input: key = page $x$, value = the list of [$partial_x$]
- Output: key = page $x$, value = PageRank $q_x$
  1. $q_x = 0$
  2. For each partial value $d$ in the list:
     - $q_x += d$
  3. $q_x = cq_x + (1-c)/N$
  4. Emit($x$, $q_x$)

Check out Kang et al ICDM’09
Mining using MapReduce

- Information retrieval
- Graph algorithms: PageRank
- Clustering: Canopy clustering, KMeans
- Classification: kNN, Naive Bayes
Canopy creation

- Construct overlapping clusters – canopies
- Make sure no two canopies with too much overlaps
- Key: no canopy centers are too close to each other

McCallum, Nigam and Ungar, "Efficient Clustering of High Dimensional Data Sets with Application to Reference Matching", KDD’00
Input: 1) points 2) threshold T1, T2 where T1 > T2
Output: cluster centroids
Put all points into a queue Q
While (Q is not empty)
  p = dequeue(Q)
  For each canopy c:
    if dist(p, c) < T1: c.add(p)
    if dist(p, c) < T2: strongBound = true
    If not strongBound: create canopy at p
  For all canopy c:
    Set centroid to the mean of all points in c
MapReduce – Canopy Map()

**Canopy creation Map()**
- **Input**: A set of points $P$, threshold $T_1$, $T_2$
- **Output**: key = null; value = a list of local canopies (total, count)
- **For each** $p$ in $P$:
  - **For each canopy** $c$:
    - if $\text{dist}(p,c) < T_1$ then $c$.total+=p, $c$.count++;
    - if $\text{dist}(p,c) < T_2$ then strongBound = true
  - If not strongBound then create canopy at $p$

**Close()**
- **For each canopy** $c$:
  - Emit(null, (total, count))
MapReduce – Canopy Reduce()

Reduce()

- Input: key = null; input values (total, count)
- Output: key = null; value = cluster centroids
- For each intermediate values
  - $p = \text{total}/\text{count}$
  - For each canopy $c$:
    - if $\text{dist}(p,c) < T1$ then $c.\text{total}+=p$, $c.\text{count}++$
    - if $\text{dist}(p,c) < T2$ then $\text{strongBound} = \text{true}$
  - If not $\text{strongBound}$ then create canopy at $p$

Close()

- For each canopy $c$: emit(null, $c.\text{total}/c.\text{count}$)

For simplicity we assume only one reducer.
KMeans: Multi-pass clustering

AssignCluster():
- For each point p
  Assign p the closest c

UpdateCentroids():
- For each cluster
  Update cluster center

Kmeans()
- While not converge:
  - AssignCluster()
  - UpdateCentroids()
MapReduce – KMeans

Map: assign each \( p \) to closest centroids

Reduce: update each centroid with its new location (total, count)

KmeansIter()

Map(\( p \)) // Assign Cluster
- For \( c \) in clusters:
  - If \( \text{dist}(p,c) < \text{minDist} \),
    then \( \text{minC}=c, \text{minDist} = \text{dist}(p,c) \)
  - Emit(\( \text{minC}.id, (p, 1) \))

Reduce() //Update Centroids
- For all values \( (p, c) \):
  - total += \( p \); count += \( c \);
- Emit(key, (total, count))
Mining using MapReduce

- Information retrieval
- Graph algorithms: PageRank
- Clustering: Canopy clustering, KMeans
- Classification: kNN, Naive Bayes
MapReduce kNN

K=3

Map()
- Input:
  - All points
  - query point \( p \)
- Output: k nearest neighbors (local)
- Emit the k closest points to \( p \)

Reduce()
- Input:
  - Key: null; values: local neighbors
  - query point \( p \)
- Output: k nearest neighbors (global)
- Emit the k closest points to \( p \) among all local neighbors
Naïve Bayes

- **Formulation:**
  \[ P(c|d) \propto P(c) \prod_{w \in d} P(w|c) \]

- **Parameter estimation**  
  - Class prior: \( \hat{P}(c) = \frac{N_c}{N} \) where \( N_c \) is #docs in \( c \), \( N \) is #docs
  - Conditional probability:
    \[ \hat{P}(w|c) = \frac{T_{cw}}{\sum_{w'} T_{cw'}} \]
    \( T_{cw} \) is # of occurrences of \( w \) in class \( c \)

- **Goals:**
  1. total number of docs \( N \)
  2. number of docs in \( c \): \( N_c \)
  3. word count histogram in \( c \): \( T_{cw} \)
  4. total word count in \( c \): \( \sum T_{cw'} \)
MapReduce: Naïve Bayes

Goals:
1. total number of docs $N$
2. number of docs in $c$: $N_c$
3. word count histogram in $c$: $T_{cw}$
4. total word count in $c$: $\sum T_{cw}$

Naïve Bayes can be implemented using MapReduce jobs of histogram computation

ClassPrior()
Map(doc):
   Emit(class_id, (doc_id, doc_length))
Combine()/Reduce()
   - $N_c = 0$; $sT_{cw} = 0$
   - For each doc_id:
     - $N_c += 1$; $sT_{cw} += doc\_length$
     - Emit($c, N_c$)

ConditionalProbability()
Map(doc):
   - For each word $w$ in doc:
     - Emit(pair($c, w$), 1)
Combine()/Reduce()
   - $T_{cw} = 0$
   - For each value $v$: $T_{cw} += v$
   - Emit(pair($c, w$), $T_{cw}$)
MapReduce Mining Summary
# Taxonomy of MapReduce algorithms

<table>
<thead>
<tr>
<th></th>
<th>One Iteration</th>
<th>Multiple Iterations</th>
<th>Not good for MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clustering</strong></td>
<td>Canopy</td>
<td>KMeans</td>
<td></td>
</tr>
<tr>
<td><strong>Classification</strong></td>
<td>Naïve Bayes, kNN</td>
<td>Gaussian Mixture</td>
<td>SVM</td>
</tr>
<tr>
<td><strong>Graphs</strong></td>
<td></td>
<td>PageRank</td>
<td></td>
</tr>
<tr>
<td><strong>Information Retrieval</strong></td>
<td>Inverted Index</td>
<td>Topic modeling (PLSI, LDA)</td>
<td></td>
</tr>
</tbody>
</table>

- One-iteration algorithms are perfect fits
- Multiple-iteration algorithms are OK fits
  - but **small shared info** have to be synchronized across iterations (typically through filesystem)
- Some algorithms are not good for MapReduce framework
  - Those algorithms typically require **large shared info** with a lot of synchronization.
  - Traditional parallel framework like MPI is better suited for those.
MapReduce for machine learning

The key is to convert into summation form (Statistical Query model [Kearns’94])

\[ y = \sum f(x) \] where \( f(x) \) corresponds to \texttt{map}(), \( \sum \) corresponds to \texttt{reduce}().

Naïve Bayes, Kmeans, Linear Regression, Locally Weighted LR, Logistic Regression, Neural Network, PCA, ICA, EM for Gaussian Mixture Model
Contents

- Introduction to MapReduce
- Large Scale Data Mining Algorithms
- Large Scale Data Mining Applications
- Summary
# MapReduce Applications in the Real World

<table>
<thead>
<tr>
<th>Organizations</th>
<th>Application of MapReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>Wide-range applications, grep / sorting, machine learning, clustering, report extraction, graph computation</td>
</tr>
<tr>
<td>Yahoo</td>
<td>Data model training, Web map construction, Web log processing using Pig, and much, much more</td>
</tr>
<tr>
<td>Amazon</td>
<td>Build product search indices</td>
</tr>
<tr>
<td>Facebook</td>
<td>Web log processing via both MapReduce and Hive</td>
</tr>
<tr>
<td>PowerSet (Microsoft)</td>
<td>HBase for natural language search</td>
</tr>
<tr>
<td>Twitter</td>
<td>Web log processing using Pig</td>
</tr>
<tr>
<td>New York Times</td>
<td>Large-scale image conversion</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Others (&gt;74)</td>
<td>Details in <a href="http://wiki.apache.org/hadoop/PoweredBy">http://wiki.apache.org/hadoop/PoweredBy</a> (so far, the longest list of applications for MapReduce)</td>
</tr>
</tbody>
</table>
Growth of MapReduce Applications in Google

[Dean, PACT’06 Keynote]

Example Use
- Distributed grep
- Distributed sort
- Term-vector per host
- Document clustering
- Web access log stat
- Web link reversal
- Inverted index
- Statistical translation
Data Mining Goes Big

- **Google**: >100,000 jobs submitted, 20PB data processed per day
  - Anyone can process tera-bytes of data w/o difficulties
- **Yahoo**: >100,000 CPUs in >25,000 computers running Hadoop
  - Biggest cluster: 4000 nodes (2*4 CPUs with 4*1TB disk)
  - Support research for Ad system and web search
- **Facebook**: 600 nodes with 4800 cores and ~2PB storage
  - Store internal logs and dimension user data
User Experience on MapReduce

Google: “completely rewrote the production indexing system using MapReduce in 2004” [Dean, OSDI’ 2004]

- Simpler code (Reduce 3800 C++ lines to 700)
- MapReduce handles failures and slow machines
- Easy to speedup indexing by adding more machines

Nutch: “convert major algorithms to MapReduce implementation in 2 weeks” [Cutting, Yahoo!, 2005]

- Before: several undistributed scalability bottlenecks, impractical to manage collections >100M pages
- After: the system becomes scalable, distributed, easy to operate; it permits multi-billion page collections
MapReduce in Academic Papers

- 981 papers cite the first MapReduce paper [Dean & Ghemawat, OSDI’04]
  - Category: **Algorithmic**, cloud overview, infrastructure, future work
  - Company: Internet (Google, Microsoft, Yahoo ..), IT (HP, IBM, Intel)
  - University: CMU, U. Penn, UC. Berkeley, UCF, U. of Missouri, ...

- >10 research areas covered by algorithmic papers
  - Indexing & Parsing, Machine Translation
  - Information Extraction, Spam & Malware Detection
  - Ads analysis, Search Query Analysis
  - Image & Video Processing, Networking
  - Simulation, Graphs, Statistics, ...

- 3 categories for MapReduce applications
  - Text processing: tokenization and indexing
  - Data warehousing: managing and querying structured data
  - Machine learning: learning and predicting data patterns
Contents

- Introduction to MapReduce
- Large Scale Data Mining Algorithms
- Large Scale Data Mining Applications
- Summary
Summary

- **MapReduce**: simplified parallel programming model
  - Build ground-up from scalability, simplicity, fault-tolerance
  - Hadoop: open-source platform on commodity machines
  - Growing collections of components & extensions

- **Data Mining Algorithms with MapReduce**
  - MapReduce-compatible for summation-form algorithms
  - Need task-specific algorithm design and tuning

- **MapReduce has been widely used in a broad range of applications and by many organizations**
  - Growing tractions from both academia and industry
  - Three application categories: text processing, data warehousing and machine learning
Future Research Opportunities

- **Algorithm** perspective
  - Convert known algorithms to their MapReduce version
  - Design descriptive language for MapReduce mining
  - Extend MapReduce primitives for data mining, such as multi-iteration MapReduce with data sharing

- **System** perspective
  - Improve MapReduce scalability for mining algorithms

- **Application** perspective
  - Discover novel applications by learning and processing such an unprecedented scale of data