Scaling Up Machine Learning

Parallel and Distributed Approaches

Ron Bekkerman, LinkedIn
Misha Bilenko, MSR
John Langford, Y!R

http://hunch.net/~large_scale_survey
<table>
<thead>
<tr>
<th>Section</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>Ron</td>
</tr>
<tr>
<td>Tree Induction</td>
<td>Misha</td>
</tr>
<tr>
<td>Break</td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td>Misha</td>
</tr>
<tr>
<td>Learning on GPUs</td>
<td>John</td>
</tr>
<tr>
<td>Linear Learning</td>
<td>John</td>
</tr>
<tr>
<td>Conclusion</td>
<td>John</td>
</tr>
</tbody>
</table>
The book

- Cambridge Uni Press
- Due in November 2011
- 21 chapters
- Covering
  - Platforms
  - Algorithms
  - Learning setups
  - Applications
Previous books

- *Mining Very Large Databases with Parallel Processing* by Alex A. Freitas and Simon H. Lavington, 1998
- *Large-Scale Parallel Data Mining* by Mohammed J. Zaki and Ching-Tien Ho (Eds.), 2000
- *Advances in Distributed and Parallel Knowledge Discovery* by Hillol Kargupta and Philip Chan, 2000
Data hypergrowth: an example

- Reuters-21578: about 10K docs (ModApte)
  *Bekkerman et al, SIGIR 2001*

- RCV1: about 807K docs
  *Bekkerman & Scholz, CIKM 2008*

- LinkedIn job title data: about 100M docs
  *Bekkerman & Gavish, KDD 2011*
New age of big data

• The world has gone mobile
  – 5 billion cellphones produce daily data

• Social networks have gone online
  – Twitter produces 200M tweets a day

• Crowdsourcing is the reality
  – Labeling of 100,000+ data instances is doable
    • Within a week 😊
Size matters

- One thousand data instances
- One million data instances
- One billion data instances
- One trillion data instances

Those are not different numbers, those are different mindsets 😊
One thousand data instances

• Will process it manually within a day (a week?)
  – No need in an automatic approach
• We shouldn’t publish main results on datasets of such size 😊
One million data instances

• Currently, the most active zone
• Can be crowdsourced
• Can be processed by a quadratic algorithm
  – Once parallelized
• 1M data collection cannot be too diverse
  – But can be too homogenous
• Preprocessing / data probing is crucial
Big dataset cannot be too sparse

• 1M data instances cannot belong to 1M classes
  – Simply because it’s not practical to have 1M classes 😊
• Here’s a statistical experiment, in text domain:
  – 1M documents
  – Each document is 100 words long
  – Randomly sampled from a unigram language model
    • No stopwords
  – 245M pairs have word overlap of 10% or more
• Real-world datasets are denser than random
Can big datasets be too dense?
Real-world example

Enron Email Dataset

This dataset was collected and prepared by the CALO Project (A Cognitive Assistant that Learns and Organizes). It contains data from about 150 users, mostly senior management of Enron, organized by folders. The corpus contains a total of about 0.5M messages. This data was originally made public and posted to the web by the Federal Energy Regulatory Commission during its investigation.

The email dataset was later purchased by Leslie Ka什ling at MIT, and turned out to have a number of integrity problems. A number of folks at SRI, notably Melinda Gervasio, worked hard to correct these problems, and it is thanks to them (not me) that the dataset is available. The dataset here does not include attachments, and some messages have been deleted "as part of a redaction effort due to privacy concerns from affected employees". Invalid email addresses were converted to something of the form user@enron.com whenever possible (i.e., recipient is specified in some parse-able format like "Doe, John" or "Mary K. Smith") and to no_address@enron.com when no recipient was specified.

I get a number of questions about this corpus each week, which I am unable to answer, mostly because they deal with preparation issues and such that I just don't know about. If you ask me a question I don't answer, please don't feel slighted.

I am distributing this dataset as a resource for researchers who are interested in improving current email tools, or understanding how email is currently used. This data is valuable to my knowledge about only substantial collection of "real" email that is public. The reason other datasets are not public is because of privacy concerns. In using this dataset, please be sensitive to the privacy of the people involved (and remember that many of these people were certainly not involved in any of the actions which precipitated the investigation).

- March 2, 2004 Version of dataset and the August 21, 2009 Version of dataset are no longer being distributed. If you are using this dataset for your work, you are requested to replace it with a newer version of the dataset below, or make the appropriate changes to your local copy. A total of four messages have been removed since the original version of the dataset.
- August 21, 2009 Version of dataset (about 423Mb, tarred and gripped).

There are also at least two online databases that allow you to search the data, at Enronemail.com and UCB.

Research uses of the dataset

This is a partial and poorly maintained list. If I've left your work out, don't take it personally, and feel free to send me a pointer and/or description.

- A paper describing the Enron data was presented at the 2004 CEAS conference.
- Some experiments associated with this data are described on Ron Bekkerman's home page.
- A social-network analysis of the data, including "useful mappings between the MD5 digest of the email bodies and such things as authors, recipients, etc", is available from Andres Corrado-
(Near) duplicate detection

Bekkerman et al, KDD 2009
One billion data instances

• Web-scale
• Guaranteed to contain data in different formats
  – ASCII text, pictures, javascript code, PDF documents...
• Guaranteed to contain (near) duplicates
• Likely to be badly preprocessed 😊
• Storage is an issue
One trillion data instances

• Beyond the reach of the modern technology
• Peer-to-peer paradigm is (arguably) the only way to process the data
• Data privacy / inconsistency / skewness issues
  – Can’t be kept in one location
  – Is intrinsically hard to sample
A solution to data privacy problem

- $n$ machines with $n$ private datasets
  - All datasets intersect
  - The intersection is shared
- Each machine learns a separate model
- Models get consistent over the data intersection
- Check out Chapter 16 to see this approach applied in a recommender system!
So what model will we learn?

- Supervised model?
- Unsupervised model?
- Semi-supervised model?

- Obviously, depending on the application 😊
  - But also on availability of labeled data
  - And its trustworthiness!
Size of training data

• Say you have 1K labeled and 1M unlabeled examples
  – Labeled/unlabeled ratio: 0.1%
  – Is 1K enough to train a supervised model?

• Now you have 1M labeled and 1B unlabeled examples
  – Labeled/unlabeled ratio: 0.1%
  – Is 1M enough to train a supervised model?
Skewness of training data

- Usually, training data comes from users
- **Explicit user feedback** might be misleading
  - Feedback providers may have various incentives
- Learning from **implicit feedback** is a better idea
  - E.g. clicking on Web search results

- In large-scale setups, skewness of training data is hard to detect
Real-world example

• Goal: find high-quality professionals on LinkedIn
• Idea: use recommendation data to train a model
  – Whoever has recommendations is a positive example

Recommendations For Sarah

Vice Presidential Nominee
John McCain 2008

“She is bright and gutsy and, guys, SHE is going to help McCain win.

Embrace her!

TraciGregory” September 13, 2008
Top qualities: Personable, High Integrity, Creative

3rd Traci Gregory,
hired Sarah as a Sincere, Moral & Ethical Service to the people of the US and the world in 2008

– Is it a good idea? 😊
Not enough (clean) training data?

• Use existing labels as a *guidance* rather than a directive
  – In a semi-supervised clustering framework
• Or label more data! 😊
  – With a little help from the crowd
Semi-supervised clustering

Bekkerman et al, ECML 2006

• Cluster unlabeled data $D$ while taking labeled data $D^*$ into account

• Construct clustering while maximizing Mutual Information $I(D; D^*)$
  – And keeping the number of clusters $k$ constant
  – $D^*$ is defined naturally over classes in $D^*$

• Results better than those of classification
Semi-supervised clustering (details)

\[ I(D, D^*) = \sum_{d \in D, d^* \in D^*} \frac{R(d, d^*)}{R(d) R(d^*)} \]

- Define an empirical joint distribution \( P(D, D^*) \)
  - \( P(d, d^*) \) is a normalized similarity between \( d \) and \( d^* \)
- Define the joint between clusterings \( R(D, D^*) \)
  - Where \( R(d, d^*) = \sum_{d \in \mathcal{I}, d^* \in \mathcal{I}^*} R(d, d^*) \)
- \( RD \) and \( RD^* \) are marginals
Crowdsourcing labeled data

• Crowdsourcing is a tough business 😊
  – People are not machines
• Any worker who can game the system will game the system
• Validation framework + qualification tests are a must
• Labeling a lot of data can be fairly expensive
How to label 1M instances

• Budget a month of work + about $50,000
How to label 1M instances

• Hire a data annotation contractor in your town
  – Presumably someone you know well enough
How to label 1M instances

• Offer the guy $10,000 for one month of work
How to label 1M instances

- Construct a qualification test for your job
- Hire 100 workers who pass it
  - Keep their worker IDs
How to label 1M instances

• Explain to them the task, make sure they get it
How to label 1M instances

• Offer them 4¢ per data instance if they do it right
How to label 1M instances

- Your contractor will label 500 data instances a day
- This data will be used to validate worker results
How to label 1M instances

- You’ll need to spot-check the results
How to label 1M instances

• You’ll need to spot-check the results
How to label 1M instances

• Each worker gets a daily task of 1000 data instances
How to label 1M instances

• Some of which are already labeled by the contractor
How to label 1M instances

- Check every worker’s result on that validation set
How to label 1M instances

• Check every worker’s result on that validation set
How to label 1M instances

• Fire the worst 50 workers
  • Disregard their results
How to label 1M instances

• Hire 50 new ones
How to label 1M instances

• Repeat for a month (20 working days)
  • 50 workers × 20 days × 1000 data points a day × 4¢
Got 1M labeled instances, now what?

• Now go train your model 😊
• Rule of the thumb: heavier algorithms produce better results
• Rule of the other thumb: forget about super-quadratic algorithms

• Parallelization looks unavoidable
## Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Example: $k$-means clustering

- An EM-like algorithm:
- Initialize $k$ cluster centroids
- E-step: associate each data instance with the closest centroid
  - Find expected values of cluster assignments given the data and centroids
- M-step: recalculate centroids as an average of the associated data instances
  - Find new centroids that maximize that expectation
Parallelizing $k$-means
Parallelizing $k$-means
Parallelizing $k$-means
## Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Peer-to-peer (P2P) systems

• Millions of machines connected in a network
  – Each machine can only contact its neighbors
• Each machine storing millions of data instances
  – Practically unlimited scale 😊
• Communication is the bottleneck
  – Aggregation is costly, broadcast is cheaper
• Messages are sent over a spanning tree
  – With an arbitrary node being the root
**k-means in P2P**

*Datta et al, TKDE 2009*

- Uniformly sample $k$ centroids over P2P
  - Using a random walk method
- Broadcast the centroids
- Run local $k$-means on each machine
- Sample $n$ nodes
- Aggregate local centroids of those $n$ nodes
### Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Virtual clusters

• Datacenter-scale clusters
  – Hundreds of thousands of machines
• Distributed file system
  – Data redundancy
• Cloud computing paradigm
  – Virtualization, full fault tolerance, pay-as-you-go
• MapReduce is #1 data processing scheme
MapReduce

• Process in parallel $\rightarrow$ shuffle $\rightarrow$ process in parallel

• Mappers output (key, value) records
  – Records with the same key are sent to the same reducer
**k-means on MapReduce**

*Panda et al, Chapter 2*

- Mappers read data portions and centroids
- Mappers **assign data instances** to clusters
- Mappers **compute new local centroids** and local cluster sizes
-Reducers **aggregate local centroids** (weighted by local cluster sizes) into new global centroids
-Reducers **write the new centroids**
Discussion on MapReduce

• MapReduce is not designed for iterative processing
  – Mappers read the same data again and again
• MapReduce looks too low-level to some people
  – Data analysts are traditionally SQL folks 😊
• MapReduce looks too high-level to others
  – A lot of MapReduce logic is hard to adapt
    • Example: grouping documents by words
MapReduce wrappers

• Many of them are available
  – At different levels of stability 😊

• Apache Pig is an SQL-like environment
  – Group, Join, Filter rows, Filter columns (Foreach)
  – Developed at Yahoo! Research

  Olston et al, SIGMOD 2008

• DryadLINQ is a C#-like environment
  – Developed at Microsoft Research

  Yu et al, OSDI 2008
**k-means in Apache Pig: input data**

- Assume we need to cluster documents
  - Stored in a 3-column table $D$:
    
    | Document | Word | Count |
    |----------|------|-------|
    | doc1     | new  | 2     |
    | doc1     | york | 2     |

- Initial centroids are $k$ randomly chosen docs
  - Stored in table $C$ in the same format as above
\[ c_d = \arg \max_c \sum_{i \in \mathcal{C}} (i^w)^2 \cdot (d^w)^2 \]

\[ \text{DOT}_\text{LEN} = \text{FOREACH } \text{DOT\_LEN} \text{ GENERATE } d, c, dXc / \text{len}_c; \]

\[ \text{SIM}_g = \text{GROUP } \text{SIM} \text{ BY } d; \]

\[ \text{CLUSTERS} = \text{FOREACH } \text{SIM}_g \text{ GENERATE TOP}(1, 2, \text{SIM}); \]
### k-means in Apache Pig: E-step

\[
D_C = \text{JOIN } C \text{ BY } w, D \text{ BY } w;
\]

\[
\text{PROD} = \text{FOREACH } D_C \text{ GENERATE } d, c, i_d \times i_c \text{ AS } i_{d,c};
\]

\[
\text{PROD}_q = \text{GROUP } \text{PROD} \text{ BY } (d, c);
\]

\[
\text{DOT}_C = \text{FOREACH } \text{PROD}_q \text{ GENERATE } d, c, \sum (i_{d,c}^w) \text{ AS } d_{Xc};
\]

\[
\text{SQR}_C = \text{FOREACH } C \text{ GENERATE } c, i_c^2 \text{ AS } i_{c,2};
\]

\[
\text{SQR}_g = \text{GROUP } \text{SQR} \text{ BY } c;
\]

\[
\text{LEN}_C = \text{FOREACH } \text{SQR}_g \text{ GENERATE } c, \sqrt{\sum (i_{c,2})} \text{ AS } \text{len}_c;
\]

\[
\text{DOT}_C = \text{JOIN } \text{LEN}_C \text{ BY } c, \text{DOT}_C \text{ BY } c;
\]

\[
\text{SIM} = \text{FOREACH } \text{DOT}_C \text{ GENERATE } d, c, d_{Xc} / \text{len}_c;
\]

\[
\text{SIM}_g = \text{GROUP } \text{SIM} \text{ BY } d;
\]

\[
\text{CLUSTERS} = \text{FOREACH } \text{SIM}_g \text{ GENERATE } \text{TOP}(1, 2, \text{SIM});
\]
**k-means in Apache Pig: E-step**

\[
D_C = \text{JOIN } C \text{ BY } w, \ D \text{ BY } w;
\]

\[
\text{PROD} = \text{FOREACH } D_C \text{ GENERATE } d, c, i_d * i_c \text{ AS } i_di_c;
\]

\[
\text{PROD}_g = \text{GROUP } \text{PROD} \text{ BY } (d, c);
\]

\[
\text{DOT}_{\text{prod}} = \text{FOREACH } \text{PROD}_g \text{ GENERATE } d, c, \sum(i_d * i_c) \text{ AS } dXc;
\]

\[
\text{SQR} = \text{FOREACH } C \text{ GENERATE } c, i_c * i_c \text{ AS } i_c^2;
\]

\[
\text{SQR}_g = \text{GROUP } \text{SQR} \text{ BY } c;
\]

\[
\text{LEN}_C = \text{FOREACH } \text{SQR}_g \text{ GENERATE } c, \sqrt{\sum(i_c^2)} \text{ AS } lenC;
\]

\[
\text{DOT}_{\text{prod}} = \text{JOIN } \text{LEN}_C \text{ BY } c, \text{DOT}_{\text{prod}} \text{ BY } c;
\]

\[
\text{SIM} = \text{FOREACH } \text{DOT}_{\text{prod}} \text{ GENERATE } d, c, dXc / lenC;
\]

\[
\text{SIM}_g = \text{GROUP } \text{SIM} \text{ BY } d;
\]

\[
\text{CLUSTERS} = \text{FOREACH } \text{SIM}_g \text{ GENERATE } \text{TOP}(1, 2, \text{SIM});
\]
$k$-means in Apache Pig: E-step

$$D\_C = \text{JOIN } C \text{ BY } w, D \text{ BY } w;$$

$$\text{PROD} = \text{FOREACH } D\_C \text{ GENERATE } d, c, i_d \ast i_c \text{ AS } i_di_c;$$

$$\text{PROD}_g = \text{GROUP } \text{PROD} \text{ BY } (d, c);$$

$$\text{DOT} = \text{FOREACH } \text{PROD}_g \text{ GENERATE } d, c, \sqrt{\sum_{i\in c} i^W_j \cdot i^W_j};$$

$$\text{SQR} = \text{FOREACH } C \text{ GENERATE } c, i_c^2 \text{ AS } i_c^2;$$

$$\text{SQR}_g = \text{GROUP } \text{SQR} \text{ BY } c;$$

$$\text{LEN} = \text{FOREACH } \text{SQR}_g \text{ GENERATE } c, \sqrt{\sum_{i\in c} i^2} \text{ AS } \text{len}_c;$$

$$\text{DOT}_g = \text{GROUP } \text{DOT} \text{ BY } c;$$

$$\text{SIM} = \text{FOREACH } \text{DOT}_g \text{ GENERATE } d, c, d^W_d / \text{len}_c;$$

$$\text{SIM}_g = \text{GROUP } \text{SIM} \text{ BY } d;$$

$$\text{CLUSTERS} = \text{FOREACH } \text{SIM}_g \text{ GENERATE } \text{TOP}(1, 2, \text{SIM});$$
$k$-means in Apache Pig: E-step

\[
D_\_C = \text{JOIN } C \text{ BY } w, D \text{ BY } w;
\]

\[
\text{PROD} = \text{FOREACH } D_\_C \text{ GENERATE } d, c, i_d * i_c \text{ AS } i_d i_c;
\]

\[
\text{PROD}_q = \text{GROUP } \text{PROD} \text{ BY } (d, c);
\]

\[
\text{DOT}_g = \text{FOREACH } \text{PROD}_q \text{ GENERATE } d, c, \sum i_d i_c \text{ AS } dXc;
\]

\[
\text{SQR}_g = \text{GROUP } \text{SQR} \text{ BY } c;
\]

\[
\text{LEN}_c = \text{FOREACH } \text{SQR}_g \text{ GENERATE } c, \sqrt{\sum (i_c^2)} \text{ AS } len_c;
\]

\[
\text{DOT}_\_LEN = \text{JOIN } \text{LEN}_c \text{ BY } c, \text{DOT}_g \text{ BY } c;
\]

\[
\text{SIM} = \text{FOREACH } \text{DOT}_\_LEN \text{ GENERATE } d, c, dXc / len_c;
\]

\[
\text{SIM}_g = \text{GROUP } \text{SIM} \text{ BY } d;
\]

\[
\text{CLUSTERS} = \text{FOREACH } \text{SIM}_g \text{ GENERATE } \text{TOP}(1, 2, \text{SIM});
\]
\[ k\text{-}means \text{ in Apache Pig: E-step} \]

\[
D_C = \text{JOIN} \ C \text{ BY } w, \ D \text{ BY } w;
\]

\[
\text{PROD} = \text{FOREACH} \ D_C \text{ GENERATE } d, c, i_d * i_c \text{ AS } i_di_c;
\]

\[
\text{PROD}_g = \text{GROUP} \ PROD \text{ BY } (d, c);
\]

\[
\text{DOT_PROD} = \text{FOREACH} \ PROD_g \text{ GENERATE } d, c, \text{SUM}(i_di_c) \text{ AS } dXc;
\]

\[
\text{SQR} = \text{FOREACH} \ C \text{ GENERATE } c, i_c * i_c \text{ AS } i_c^2;
\]

\[
\text{SQR}_g = \text{GROUP} \ C \text{ GENERATE } c, \text{SQRT(SUM}(i_c^2)) \text{ AS } len_c;
\]

\[
\text{DOT_LEN} = \text{JOIN} \ len_C \text{ BY } c, \text{DOT_PROD} \text{ BY } c;
\]

\[
\text{SIM} = \text{FOREACH} \ DOT_LEN \text{ GENERATE } d, c, \frac{dXc}{len_c};
\]

\[
\text{SIM}_g = \text{GROUP} \ SIM \text{ BY } d;
\]

\[
\text{CLUSTERS} = \text{FOREACH} \ SIM_g \text{ GENERATE } \text{TOP}(1, 2, \ SIM);
\]
**k-means in Apache Pig: M-step**

\[ D_{C\_W} = JOIN CLUSTERS BY d, D BY d; \]

\[ D_{C\_W_g} = GROUP D_{C\_W} BY (c, w); \]
\[ SUMS = FOREACH D_{C\_W_g} \text{ GENERATE } c, w, \text{SUM}(i_d) \text{ AS sum}; \]

\[ D_{C\_W_{gg}} = GROUP D_{C\_W} BY c; \]
\[ SIZES = FOREACH D_{C\_W_{gg}} \text{ GENERATE } c, \text{COUNT}(D_{C\_W}) \text{ AS size}; \]

\[ SUMS\_SIZES = JOIN SIZES BY c, SUMS BY c; \]
\[ C = FOREACH SUMS\_SIZES \text{ GENERATE } c, w, \text{sum} / \text{size} \text{ AS } i_c; \]
MapReduce job setup time

- In an iterative process, setting up a MapReduce job at each iteration is costly
- Solution: *forward scheduling*
  - Setup the next job before the previous completed

*Panda et al, Chapter 2*
**k-means in DryadLINQ**

*Budiu et al, Chapter 3*

Vector `NearestCenter(Vector point, IQueryable<Vector> centers)`
{
    var nearest = centers.First();
    foreach (var center in centers)
        if ((point - center).Norm() < (point - nearest).Norm())
            nearest = center;
    return nearest;
}

IQueryable<Vector> `KMeansStep(IQueryable<Vector> vectors, IQueryable<Vector> centers)`
{
    return vectors.GroupBy(vector => `NearestCenter(vector, centers)`)
        .Select(g => g.Aggregate((x,y) => x+y) / g.Count());
}
DryadLINQ: \( k \)-means execution plan
Takeaways on MapReduce wrappers

• Machine learning in SQL is fairly awkward 😊
• DryadLINQ looks much more suitable
  – Beta available at
  – Check out Chapter 3 for a Kinect application!!!
• Writing high-level code requires deep understanding of low-level processes
## Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td><strong>HPC Clusters</strong></td>
<td><strong>MPI / MapReduce</strong></td>
<td><strong>Terabytes</strong></td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
HPC clusters

• High Performance Computing clusters / blades / supercomputers
  – Thousands of cores
• Great variety of architectural choices
  – Disk organization, cache, communication etc.
• Fault tolerance mechanisms are not crucial
  – Hardware failures are rare
• Most typical communication protocol: MPI
  – Message Passing Interface

Gropp et al, MIT Press 1994
Message Passing Interface (MPI)

• Runtime communication library
  – Available for many programming languages
• `MPI_Bsend(void* buffer, int size, int destID)`
  – Serialization is on you 😊
• `MPI_Recv(void* buffer, int size, int sourceID)`
  – Will wait until receives it
• `MPI_Bcast` — broadcasts a message
• `MPI_Barrier` — synchronizes all processes
MapReduce vs. MPI

- MPI is a generic framework
  - Processes send messages to other processes
  - Any computation graph can be built
- Most suitable for the master/slave model
$k$-means using MPI

- Slaves read data portions
- Master **broadcasts centroids** to slaves
- Slaves **assign data instances** to clusters
- Slaves **compute new local centroids** and local cluster sizes
  - Then send them to the master
- Master **aggregates local centroids** weighted by local cluster sizes into new global centroids

*Pednault et al, Chapter 4*
Two features of MPI parallelization

Pednault et al, Chapter 4

- State-preserving processes
  - Processes can live as long as the system runs
  - No need to read the same data again and again
  - All necessary parameters can be preserved locally

- Hierarchical master/slave paradigm
  - A slave can be a master of other processes
  - Could be very useful in dynamic resource allocation
    - When a slave recognizes it has too much stuff to process
Takeaways on MPI

- Old, well established, well debugged
- Very flexible
- Perfectly suitable for iterative processing
- Fault intolerant
- Not that widely available anymore 😞
  - An open source implementation: OpenMPI
  - MPI can be deployed on Hadoop

Ye et al, CIKM 2009
## Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Multicore

• One machine, up to dozens of cores
• Shared memory, one disk
• Multithreading as a parallelization scheme
• Data might not fit the RAM
  – Use streaming to process the data in portions
  – Disk access may be the bottleneck
• If it does fit, RAM access is the bottleneck
  – Use uniform, small size memory requests

Tatikonda & Parthasarathy, Chapter 20
## Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td><strong>GPU</strong></td>
<td><strong>CUDA</strong></td>
<td><strong>Gigabytes</strong></td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Graphics Processing Unit (GPU)

• GPU has become General-Purpose (GP-GPU)
• CUDA is a GP-GPU programming framework
  – Powered by NVIDIA
• Each GPU consists of hundreds of multiprocessors
• Each multiprocessor consists of a few ALUs
  – ALUs execute the same line of code synchronously
• When code branches, some multiprocessors stall
  – Avoid branching as much as possible
Machine learning with GPUs

• To fully utilize a GPU, the data needs to fit in RAM
  – This limits the maximal size of the data
• GPUs are optimized for speed
  – A good choice for *real-time* tasks
• A typical usecase: a model is trained offline and then applied in real-time (*inference*)
  – Machine vision / speech recognition are example domains

*Coates et al, Chapter 18*
*Chong et al, Chapter 21*
**k-means clustering on a GPU**

*Hsu et al, Chapter 5*

- **Cluster membership assignment** done on GPU:
  - Centroids are uploaded to every multiprocessor
  - A multiprocessor works on one data vector at a time
  - Each ALU works on one data dimension

- **Centroid recalculation** is then done on CPU

- Most appropriate for processing *dense* data

- Scattered memory access should be avoided

- A multiprocessor reads a data vector while its ALUs process a previous vector
Performance results

- 4 millions 8-dimensional vectors
- 400 clusters
- 50 $k$-means iterations

- 9 seconds!!!
Parallelization: platform choices

<table>
<thead>
<tr>
<th>Platform</th>
<th>Communication Scheme</th>
<th>Data size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer-to-Peer</td>
<td>TCP/IP</td>
<td>Petabytes</td>
</tr>
<tr>
<td>Virtual Clusters</td>
<td>MapReduce / MPI</td>
<td>Terabytes</td>
</tr>
<tr>
<td>HPC Clusters</td>
<td>MPI / MapReduce</td>
<td>Terabytes</td>
</tr>
<tr>
<td>Multicore</td>
<td>Multithreading</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>GPU</td>
<td>CUDA</td>
<td>Gigabytes</td>
</tr>
<tr>
<td>FPGA</td>
<td>HDL</td>
<td>Gigabytes</td>
</tr>
</tbody>
</table>
Field-programmable gate array (FPGA)

- Highly specialized hardware units
- Programmable in Hardware Description Language (HDL)
- Applicable to training and inference

Durdanovic et al, Chapter 7
Farabet et al, Chapter 19

- Check out Chapter 7 for a hybrid parallelization: multicore (coarse-grained) + FPGA (fine-grained)
How to choose a platform

• Obviously depending on the size of the data
  – A cluster is a better option if data doesn’t fit in RAM

• Optimizing for speed or for throughput
  – GPUs and FPGAs can reach enormous speeds

• Training a model / applying a model
  – Training is usually offline
Thank You!

http://hunch.net/~large_scale_survey
Parallel Information-Theoretic Co-Clustering

Bekkerman & Scholz, Chapter 13
Illustration of Distributional Co-Clustering

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \sum X = 3 \quad 1 \quad 5 \quad 1 \quad 4 \quad 1 \quad 5 \quad 6 \quad 4 \]

\[ \sum Y = 3 \quad 1 \quad 5 \quad 1 \quad 4 \quad 1 \quad 5 \quad 6 \quad 4 \]

\[ \sum = 30 \]
Illustration of Distributional Co-Clustering

\[ R(3, 8) = \frac{1}{3} \] 
\[ R(3) = \frac{1}{30}.1 \] 
\[ R(8) = \frac{1}{30}.2 \]
Illustration of Distributional Co-Clustering
Illustration of Distributional Co-Clustering

\[
\begin{pmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
0 & 0 & 3 \\
0 & 1 & 2 \\
1 & 1 & 2 \\
1 & 1 & 3 \\
0 & 1 & 1 \\
0 & 2 & 2 \\
0 & 2 & 1
\end{pmatrix}
\]

\[
\sum
\begin{pmatrix}
4 & 10 & 15 \\
3 & 3 & 3 \\
3 & 4 & 5 \\
2 & 4 & 3 \\
30
\end{pmatrix}
\]

\[
R(Y|x_3) = (0 0 1)
\]
Illustration of Distributional Co-Clustering

\[ H(x_2, y_2) = 1 \]

\[ R(Y|X) = (25, 25, 55) \]

Mutual Information

\[ \sum \sum p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \]
Information-Theoretic Co-Clustering

• Construct clusterings of $X$ and $Y$ by optimizing Mutual Information

$$\arg \max_{X,Y} \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$
Two optimization strategies

• Both strategies:
  – Randomly initialize clusterings of $X$ and $Y$
  – Alternate reclustering $X$ wrt $Y$ and $Y$ wrt $X$

• Strategy 1: Centroid-based

  – At iteration $t$, assign each $x$ to cluster $\tilde{x}$ with
    \[
    \arg \min_{\tilde{x}} D_{KL}(P_Y|x) | P(t)(Y|\tilde{x}) | \\
    \]  
    “centroid”

  – Compute $P(t+1)(Y|\tilde{x})$ for each new cluster $\tilde{x}$
Sequential Co-Clustering

• For each $x$:
  – Remove $x$ from its original cluster
  – For each cluster $\tilde{x}$:
    • Compute the delta in the Mutual Information if $x$ is assigned to $\tilde{x}$
  – Assign $x$ to the cluster such that the delta is maximal
Sequential vs. centroid-based updates
Sequential vs. centroid-based updates
Theoretical results in a nutshell

• The centroid-based algorithm misses updates

• Sequential CC updates more aggressively & faster

• Theorem:
  Sequential CC has a true subset of local optima compared to centroid-based IT-CC
## Results on small data sets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>centroid $\pm$</th>
<th>sequential $\pm$</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>acheyer</em></td>
<td>$39.0 \pm .6$</td>
<td>$46.1 \pm .3$</td>
</tr>
<tr>
<td><em>mgondek</em></td>
<td>$61.3 \pm 1.5$</td>
<td>$63.4 \pm 1.1$</td>
</tr>
<tr>
<td><em>sanders-r</em></td>
<td>$56.1 \pm .7$</td>
<td>$60.2 \pm .4$</td>
</tr>
<tr>
<td><em>20NG</em></td>
<td>$54.2 \pm .7$</td>
<td>$57.7 \pm .2$</td>
</tr>
</tbody>
</table>

Does the sequential strategy work better on large sets?
From inherently sequential to parallel

- Objective function: \[ I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = \]
  \[ \sum_{y \in Y} p(x_1,y) \log \frac{p(x_1,y)}{p(x_1)p(y)} + \sum_{y \in Y} p(x_2,y) \log \frac{p(x_2,y)}{p(x_2)p(y)} + \sum_{y \in Y} p(x_3,y) \log \frac{p(x_3,y)}{p(x_3)p(y)} + \]
  \[ \sum_{y \in Y} p(x_4,y) \log \frac{p(x_4,y)}{p(x_4)p(y)} + \sum_{y \in Y} p(x_5,y) \log \frac{p(x_5,y)}{p(x_5)p(y)} + \sum_{y \in Y} p(x_6,y) \log \frac{p(x_6,y)}{p(x_6)p(y)} \]
Parallel sequential co-clustering

- Initialize clusters at random
- Split clusters to pairs
- Assign each pair to one machine
- “Shuffle” clusters in parallel
  - Try moving each instance from one cluster to another
- Assign a different pair of clusters to each machine
- Repeat to cover all cluster pairs
How can we make sure that each cluster pair is generated exactly once?

With minimal communication costs?
Tournament

0 7 6 7
1 6 0 5
2 5 1 4
3 4 2 3
3 7 2 3
0 6 0 6
1 5 1 5
2 4 2 4
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>67</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>05</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>14</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>23</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>27</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>65</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>04</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>13</td>
<td>02</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Setup

• DataLoom:
  – Parallelization of sequential co-clustering
  – MPI-based implementation

• Centroid-based IT-CC:
  – Implementation in MapReduce

• Double $k$-means:
  – Co-clustering in Euclidean space
Overall costs per full iteration

$k : \text{#clusters}, \ m : \text{#machines}, \ v : \text{#values in } P(Y \mid X)$

- Communication:
  - centroid-based \( O(2m \cdot k \cdot |\tilde{Y}|) = O(m \cdot k^2) \)
  - DataLoom \( O((k-1) \cdot (v/2)) = O(k \cdot v) \)

- In our experiments (many machines, sparse data):
  (sending centroids) \( 2m|\tilde{Y}| \approx v/2 \) (sending cluster)

- CPU cycles: \( O(k \cdot v) \) for both centroid-based & DataLoom
Experimental results

- **RCV1 dataset**
  - 800,000 docs
  - 150,000 words
- **Netflix (KDD Cup '07)**
  - 18,000 movies
  - 480,000 users

- 55 2nd-level Reuters categories
- 800 document / 800 term clusters
- clustering without label information
- choose the mode of each cluster

\[
q(x, y) := p(x) \cdot p(y) \cdot \frac{p(\tilde{x}, \tilde{y})}{p(\tilde{x}) \cdot p(\tilde{y})}
\]
Conclusion

• DataLoom: parallelization of sequential co-clustering

• Theoretical result:
  – Sequential updates superior for cluster updates

• Experimental results:
  – Excellent clustering results on two large benchmarks