CIS 602-01: Scalable Data Analysis

Graph Data

Dr. David Koop
Machine Learning

- Traditional Programming

  Data → Computer → Output
  Program → Computer

- Machine Learning

  Data → Computer → Program
  Output → Computer
Machine Learning

• Every machine learning algorithm has three components:
  - Representation
  - Evaluation
  - Optimization
Representation

• Decision trees
• Sets of rules / Logic programs
• Instances
• Graphical models (Bayes/Markov nets)
• Neural networks
• Support vector machines
• Model ensembles
• Etc.
Evaluation

- Accuracy
- Precision and recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- Etc.
Optimization

- Combinatorial optimization
  - E.g.: Greedy search
- Convex optimization
  - E.g.: Gradient descent
- Constrained optimization
  - E.g.: Linear programming
Types of Learning

- **Supervised (inductive) learning**
  - Training data includes desired outputs
- **Unsupervised learning**
  - Training data does not include desired outputs
- **Semi-supervised learning**
  - Training data includes a few desired outputs
- **Reinforcement learning**
  - Rewards from sequence of actions
Scalable Machine Learning Constraints

- Constrained by **size**:  
  - Training dataset fits in RAM, need to predict for a much larger dataset  
  - Training dataset doesn't even fit in RAM, want to scale out by  
    - adopting algorithms that work in batches locally, or  
    - on a distributed cluster.

- Constrained by **time**:  
  - Want to fit more models (think hyper-parameter optimization or ensemble learning) on my dataset in a given amount of time.  
  - Scale out by fitting more models in parallel:  
    - on my laptop by using more cores, or  
    - on a cluster.
Sampling: How much training data?

- You may not need to scale
- How large should the sample be? Use statistics

![Learning Curves (Naive Bayes)](image-url)
Scaling scikit-learn

- Fit, then predict
- May be able to fit with training set in memory
- Can run predict out of core
  - Use dask map-partitions
  - Use the cloud (kubernetes)
- Watch Tom's blog for more posts:
  - how dask can speed up your existing pipelines by executing them in parallel
  - scikit-learn's out of core API for when your training dataset doesn't fit in memory
  - using dask to implement distributed machine learning algorithms

[T. Augspurger, 2017]
Deep Learning at Google

Across many products/areas:
- Android
- Apps
- drug discovery
- Gmail
- Image understanding
- Maps
- Natural language understanding
- Photos
- Robotics research
- Speech
- Translation
- YouTube
- ... many others ...

[J. Dean, 2015]
Developer Wish List for Machine Learning System

• **Ease of expression**: for lots of crazy ML ideas/algorithms
• **Scalability**: can run experiments quickly
• **Portability**: can run on wide variety of platforms
• **Reproducibility**: easy to share and reproduce research
• **Production readiness**: go from research to real products

[J. Dean, 2015]
Computation is a dataflow graph

Graph of Nodes, also called Operations or ops.

- biases
- weights
- examples
- labels

[J. Dean, 2015]
Computation is a dataflow graph with tensors.
Computation is a dataflow graph with state

'Biases' is a variable

Some ops compute gradients

−= updates biases

'Biases' is a variable

Some ops compute gradients

−= updates biases

[J. Dean, 2015]
Computation is a **distributed** dataflow graph

Devices: Processes, Machines, GPUs, etc

[J. Dean, 2015]
Computation is a **distributed** dataflow graph

Devices: Processes, Machines, GPUs, etc

[J. Dean, 2015]
Experiment Turnaround Time & Research Productivity

• Minutes, Hours:
  - Interactive research! Instant gratification!

• 1-4 days
  - Tolerable
  - Interactivity replaced by running many experiments in parallel

• 1-4 weeks
  - High value experiments only
  - Progress stalls

• >1 month
  - Don't even try

[J. Dean, 2015]
How to Scale?

• **Model Parallelism**
  - Best way to decrease training time: decrease the step time
  - Many models have lots of inherent parallelism
  - Problem is distributing work so communication doesn’t kill you
  - Can do across single core (SIMD), multiple cores, devices (e.g. GPUs), machines

• **Data Parallelism**
  - Use multiple model replicas to process different examples at once
  - Speedups depend on the kind of model
    - Dense models: 10-40x speedup from 50 replicas
    - Sparse models: support many more replicas (as many as 1,000)
  - Really important for Google's problems

[J. Dean, 2015]
Model Parallelism

Minimal network traffic: The most densely connected areas are on the same partition

Layer N
...
Layer 1
Layer 0

[J. Dean, 2015]
Data Parallelism

\[ p += \Delta p \]

[J. Dean, 2015]
Assignment 3

- Analysis of the IRS Form 990 Filings Data
- Sign up for AWS Educate
- Part 1: Run Spark Locally over Index
- Part 2: Run Spark Locally over Subset
- Part 3: Run Spark on AWS over Full Dataset (Year)
- Windows Issues:
  - Install Spark separately from conda's pyspark install
  - Run notebook normally and use findspark to link to the pyspark libraries
Projects

• Reviewing project proposals today and tomorrow
• Looking for scalability aspects: if you can easily complete analyses with pandas or matplotlib, it probably doesn't meet the criteria
• If your project doesn't quite meet the criteria, I will try to offer suggestions to satisfy the requirements
• Continue to work on projects
Graphs as Data

Nodes

<table>
<thead>
<tr>
<th>ID</th>
<th>Atom</th>
<th>Electrons</th>
<th>Protons</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>S</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>N</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Edges

<table>
<thead>
<tr>
<th>ID1</th>
<th>ID2</th>
<th>Bonds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Social Networks

[Facebook Map, December 2010]

[P. Butler, 2010]
Graphs

• "Every time someone visits news feed there are on average 1,500 potential stories from friends, people they follow and pages for them to see, and most people don’t have enough time to see them all" - Lars Backstrom, Facebook Engineer, 2013
Graph Analytics

K. Salama
Frameworks In-depth

SIGMOD 2016 Tutorial
Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets

N. Satish et al.
Evaluating Different Graph Algorithms

- Select a variety of different algorithms:
  - PageRank
  - Breadth First Search
  - Collaborative Filtering
  - Triangle Counting

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Graph type</th>
<th>Vertex property</th>
<th>Edge access pattern</th>
<th>Message size (Bytes/edge)</th>
<th>Vertex active?</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>Directed, unweighted edges</td>
<td>Double (pagerank)</td>
<td>Streaming</td>
<td>Constant (8)</td>
<td>All iterations</td>
</tr>
<tr>
<td>Breadth First Search</td>
<td>Undirected, unweighted edges</td>
<td>Int (distance)</td>
<td>Random</td>
<td>Constant (4)</td>
<td>Some iterations</td>
</tr>
<tr>
<td>Collaborative Filtering</td>
<td>Bipartite graph; Undirected, weighted edges</td>
<td>Array of Doubles (p_u or q_v)</td>
<td>Streaming</td>
<td>Constant (8K)</td>
<td>All iterations</td>
</tr>
<tr>
<td>Triangle Counting</td>
<td>Directed, unweighted edges</td>
<td>Long (Ntriangles)</td>
<td>Streaming</td>
<td>Variable (0-10^6)</td>
<td>Non-iterative</td>
</tr>
</tbody>
</table>

[Satish et al., 2014]
Graph Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Programming model</th>
<th>Multi node</th>
<th>Language usage</th>
<th>Graph Partitioning</th>
<th>Communication layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>N/A</td>
<td>Yes</td>
<td>C/C++</td>
<td>N/A</td>
<td>MPI</td>
</tr>
<tr>
<td>Galois</td>
<td>Vertex</td>
<td>Yes</td>
<td>C++</td>
<td>1-D</td>
<td>Sockets</td>
</tr>
<tr>
<td>CombBLAS</td>
<td>Sparse matrix</td>
<td>Yes</td>
<td>C++</td>
<td>2-D</td>
<td>MPI</td>
</tr>
<tr>
<td>SocialLite</td>
<td>Datalog</td>
<td>Yes</td>
<td>Java</td>
<td>1-D</td>
<td>Sockets</td>
</tr>
<tr>
<td>Giraph</td>
<td>Task-based</td>
<td>No</td>
<td>C/C++</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Vertex</td>
<td>Yes</td>
<td>Java</td>
<td>1-D</td>
<td>Netty</td>
</tr>
</tbody>
</table>

Figure 2: An example directed graph with 4 vertices

Table 2: High level comparison of the graph frameworks

[Satish et al., 2014]
### Real-world datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Vertices</th>
<th># Edges</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facebook [34]</td>
<td>2,937,612</td>
<td>41,919,708</td>
<td>Facebook user interaction graph</td>
</tr>
<tr>
<td>Wikipedia [14]</td>
<td>3,566,908</td>
<td>84,751,827</td>
<td>Wikipedia Link graph</td>
</tr>
<tr>
<td>Netflix [9]</td>
<td>480,189 users</td>
<td>99,072,112</td>
<td>Netflix Prize</td>
</tr>
<tr>
<td>Twitter [20]</td>
<td>61,578,415</td>
<td>1,468,365,182</td>
<td>Twitter follower graph</td>
</tr>
<tr>
<td>Yahoo Music [7]</td>
<td>1,000,990 users</td>
<td>252,800,275</td>
<td>Yahoo! KDDCup 2011 music ratings</td>
</tr>
<tr>
<td>[Synthetic Graph500 [23]]</td>
<td>536,870,912</td>
<td>8,589,926,431</td>
<td>Described in Section 4</td>
</tr>
<tr>
<td>[Synthetic Collaborative Filtering]</td>
<td>63,367,472 users</td>
<td>16,742,847,256</td>
<td>Described in Section 4</td>
</tr>
</tbody>
</table>

[Satish et al., 2014]
Results (Run Time)

(a) PageRank

(b) Breadth-First Search
Results (Run Time)

(c) Collaborative Filtering

(d) Triangle counting
Results (Scalability): Synthetic Graphs

**Figures 4(a), 4(b), 4(c) and 4(d) show the results of multi node runs on synthetically generated data sets for our benchmarks.**

The table shows performance slowdowns of different frameworks used across various algorithms.

**Table 6:**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PageRank</th>
<th>BFS</th>
<th>Collaborative Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CombBLAS</strong></td>
<td>13.1</td>
<td>3.5</td>
<td>7.1</td>
</tr>
<tr>
<td><strong>GraphLab</strong></td>
<td>29.5</td>
<td>12.1</td>
<td>18.9</td>
</tr>
<tr>
<td><strong>Socialite</strong></td>
<td>494.3</td>
<td>54.4</td>
<td>74.4</td>
</tr>
<tr>
<td><strong>Giraph</strong></td>
<td>1248</td>
<td>1 6</td>
<td>13 32</td>
</tr>
</tbody>
</table>

**Note:** The trends on the synthetic dataset are in line with real-world data, showing that our synthetic generators are effective in modeling real-world data.
Results (Scalability): Synthetic Graphs

BFS (Weak scaling, 128M undirected edges/node)

Overall time (seconds)

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Native</th>
<th>Combblas</th>
<th>Graphlab</th>
<th>Socialite</th>
<th>Giraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
</tr>
</tbody>
</table>

(b) Breadth-First Search
Results (Scalability): Synthetic Graphs

Collaborative Filtering (Weak scaling, 250 M edges/node)

- Native
- Combblas
- Graphlab
- Socialite
- Giraph

Time per iteration (seconds)

Number of nodes

(c) Collaborative Filtering
Results (Scalability): Synthetic Graphs

Triangle Counting (Weak scaling, 32M edges/node)

- **Native**
- **Combblas**
- **Graphlab**
- **Socialite**
- **Giraph**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PageRank</th>
<th>BFS</th>
<th>Coll. Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combblas</td>
<td>3.5</td>
<td>7.1</td>
<td>29.5</td>
</tr>
<tr>
<td>Graphlab</td>
<td>11.2</td>
<td>7.1</td>
<td>12.1</td>
</tr>
<tr>
<td>Socialite</td>
<td>7.9</td>
<td>7.1</td>
<td>1.5</td>
</tr>
<tr>
<td>Giraph</td>
<td>54.4</td>
<td>87.9</td>
<td>74.4</td>
</tr>
</tbody>
</table>

As a convenient summary of performance, Table 6 shows the cumulative slowdowns of different frameworks on multi-node bench marks on different frameworks for our applications. Each entry is a compliance of performance slowdowns of different frameworks on multi-node benchmarking.

Table 6:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PageRank</th>
<th>BFS</th>
<th>Collaborative Filtering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Combblas</td>
<td></td>
<td></td>
<td>3.5</td>
</tr>
<tr>
<td>Graphlab</td>
<td>11.2</td>
<td></td>
<td>7.1</td>
</tr>
<tr>
<td>Socialite</td>
<td>7.9</td>
<td></td>
<td>7.1</td>
</tr>
<tr>
<td>Giraph</td>
<td>54.4</td>
<td></td>
<td>87.9</td>
</tr>
</tbody>
</table>

Finally, note that the trends on the synthetic dataset are in line well with real-world data, showing that our synthetic generators are effective in modeling real-world data.
Results (Multiple Nodes): Real-world Data

In many cases, the data communicated among workers is too large to fit in memory. Each worker requires more memory and we cannot add more workers even though the number of cores per node is 24. This is because memory limitations restrict the number of workers that can be assigned to a single node. Low CPU utilization across the board. This is because memory limitations result in a net benefit of about 3.2 and 2.2X respectively (and a corresponding reduction in the number of reads/writes). For BFS and Pagerank, this results in a benefit of slightly over 2X. GraphLab keeps a cuckoo hash data structure that allows for a fast union of neighbor bit-vectors and delta coding. For BFS and Pagerank, this reduces the number of reads/writes. Careful selection of data structure can benefit performance significantly. For example, algorithms like BFS and Triangle Counting can take advantage of bit-vectors instead of cache misses. In our native BFS and Triangle Counting code, this results in a net benefit of about 3.2 and 2.2X respectively (and a corresponding reduction in the number of reads/writes). Similar analysis for other frameworks as well.

In the next section, we will discuss optimizations that we performed to achieve native code level performance. For example, we look at only the measured network down-sizes we see with graph frameworks compared to native implementations. For example, we look at only the measured network down-sizes we see with graph frameworks compared to native implementations. For example, we look at only the measured network down-sizes we see with graph frameworks compared to native implementations. For example, we look at only the measured network down-sizes we see with graph frameworks compared to native implementations. For example, we look at only the measured network down-sizes we see with graph frameworks compared to native implementations.

Given the various performance issues we see in the frameworks, we next delve deeper into other metrics beside runtime to gain more insight into the performance characteristics. The CPU utilization of various frameworks is measured using the sar/sysstat monitoring tools. These results are summarized in Figure 6. We briefly describe our key findings in this section.
Results: Resource Utilization

(a) PageRank

(b) Breadth-First Search
Results: Resource Utilization

(c) Collaborative Filtering
(d) Triangle Counting
GraphX: Unifying Data-Parallel and Graph-Parallel Analytics

J. Gonzalez