Case Study 4: Collaborative Filtering

Graph-Parallel Problems

Synchronous v. Asynchronous Computation

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Carlos Guestrin
March 12th, 2013

©Carlos Guestrin 2013

Needless to Say, We Need Machine Learning for Big Data

flickr 6 Billion Flickr Photos

28 Million Wikipedia Pages

facebook 1 Billion Facebook Users

YouTube 72 Hours a Minute

“... data a new class of economic asset, like currency or gold.”

©Carlos Guestrin 2013
CPUs Stopped Getting Faster…

ML in the Context of Parallel Architectures

- But scalable ML in these systems is hard, especially in terms of:
  - 1. Programmability
  - 2. Data distribution
  - 3. Failures
Move Towards Higher-Level Abstraction

- Distributed computing challenges are hard and annoying!
  1. Programmability
  2. Data distribution
  3. Failures
- High-level abstractions try to simplify distributed programming by hiding challenges:
  - Provide different levels of robustness to failures, optimizing data movement and communication, protect against race conditions...
  - Generally, you are still on your own WRT designing parallel algorithms
- Some common parallel abstractions:
  - Lower-level:
    - Pthreads: abstraction for distributed threads on single machine
    - MPI: abstraction for distributed communication in a cluster of computers
  - Higher-level:
    - Map-Reduce (Hadoop: open-source version): mostly data-parallel problems
    - GraphLab: for graph-structured distributed problems

Simplest Type of Parallelism: Data Parallel Problems

- You have already learned a classifier
  - What’s the test error?
- You have 10B labeled documents and 1000 machines
- Problems that can be broken into independent subproblems are called data-parallel (or embarrassingly parallel)
- Map-Reduce is a great tool for this…
  - Focus of today’s lecture
  - but first a simple example
Data Parallelism (MapReduce)

Solve a huge number of independent subproblems, e.g., extract features in images

Map-Reduce Abstraction

- **Map**: Transforms a data element
  - Data-parallel over elements, e.g., documents
  - Generate (key, value) pairs
    - "value" can be any data type
  - Example: `map (document) for word in doc emit (word, 1)`

- **Reduce**: Aggregate values for each key
  - Must be commutative-associate operation
  - Data-parallel over keys
    - Example: `reduce (word, count: list([1])) c = 0 for i in count c = c + count[i] emit (word, c)`

- Map-Reduce has long history in functional programming
  - But popularized by Google, and subsequently by open-source Hadoop implementation from Yahoo!
Map-Reduce – Execution Overview

Map Phase

- Split data across machines

Reduce Phase

- Assign tuple \((k_i, v_i)\) to machine \(h[k_i]\)

Shuffle Phase

- Split data
- Across machines

Issues with Map-Reduce Abstraction

- Often all data gets moved around cluster
  - Very bad for iterative settings

- Definition of Map & Reduce functions can be unintuitive in many apps
  - Graphs are challenging

- Computation is synchronous
SGD for Matrix Factorization in Map-Reduce?

\[ e_t = L_{t+1} \cdot R_{t+1} - r_{t+1} \]

- Map and Reduce functions???
- Map-Reduce:
  - Data-parallel over all mappers
  - Data-parallel over reducers with same key
- Here, one update at a time!

Matrix Factorization as a Graph
Flashback to 1998

First Google advantage: a Graph Algorithm & a System to Support it!

Graphs encode the relationships between:

- People
- Facts
- Products
- Ideas
- Interests

Big: 100 billions of vertices and edges and rich metadata

- Facebook (10/2012): 1B users, 144B friendships
- Twitter (2011): 15B follower edges

©Carlos Guestrin 2013
Facebook Graph

Data model
Objects & Associations

Label a Face and Propagate
Pairwise similarity not enough...

Propagate Similarities & Co-occurrences for Accurate Predictions
Example: *Estimate Political Bias*

Latent Topic Modeling (LDA)
ML Tasks Beyond Data-Parallelism

Map Reduce
- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

Graph-Parallel
- Graphical Models
  - Gibbs Sampling
  - Belief Propagation
  - Variational Opt.
- Collaborative Filtering
  - Tensor Factorization
- Semi-Supervised Learning
  - Label Propagation
  - CoEM
- Graph Analysis
  - PageRank
  - Triangle Counting

Example of a Graph-Parallel Algorithm
PageRank

Depends on rank of who follows her

What’s the rank of this user?

Depends on rank of who follows them...

Loops in graph → Must iterate!

PageRank Iteration

\[ R[i] = \alpha + (1 - \alpha) \sum_{(j,i) \in E} w_{ji} R[j] \]

- \( \alpha \) is the random reset probability
- \( w_{ji} \) is the prob. transitioning (similarity) from \( j \) to \( i \)

\[ R[2] = 0.15 + 0.85 \left( 0.2 R[1] + 0.5 R[2] + 0.3 R[3] \right) \]
Properties of Graph Parallel Algorithms

- Dependency Graph
- Local Updates
- Iterative Computation

Addressing Graph-Parallel ML

Data-Parallel

Graph-Parallel Abstraction

Map Reduce

- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

Graph-Parallel

- Collaborative Filtering: Tensor Factorization
- Semi-Supervised Learning: Label Propagation, CoEM
- Data-Mining: PageRank, Triangle Counting
Graph Computation:

Synchronous v. Asynchronous

Bulk Synchronous Parallel Model:
Pregel (Giraph)
Map-Reduce – Execution Overview

**Map Phase**
- M1
- M2
- M1000

**Shuffle Phase**
- Assign tuple (k, v) to machine h[k]

**Reduce Phase**
- M1
- M2
- M1000

BSP – Execution Overview

**Compute Phase**
- M1
- M2
- M1000

**Communicate Phase**
- Message machine for every edge (vid,vid')
Bulk synchronous parallel model provably inefficient for some ML tasks
Asynchronous Belief Propagation

Challenge = Boundaries

Synthetic Noisy Image

Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

BSP ML Problem: Synchronous Algorithms can be Inefficient

Theorem: Bulk Synchronous BP $O(\#\text{vertices})$ slower than Asynchronous BP
## Synchronous v. Asynchronous

<table>
<thead>
<tr>
<th>Synchronous processing:</th>
<th>Asynchronous processing:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Computation in phases</td>
<td>- Vertices see latest information from neighbors</td>
</tr>
<tr>
<td>- All vertices participate in a phase</td>
<td>- Most closely related to sequential execution</td>
</tr>
<tr>
<td>- Though OK to say no-op</td>
<td>- Harder to build:</td>
</tr>
<tr>
<td>- All messages are sent</td>
<td>- Race conditions can happen all the time</td>
</tr>
<tr>
<td>- Simpler to build, like Map-Reduce</td>
<td>- Must protect against this issue</td>
</tr>
<tr>
<td>- No worries about race conditions, barrier guarantees data consistency</td>
<td>- More complex fault tolerance</td>
</tr>
<tr>
<td>- Simpler to make fault-tolerant, save data on barrier</td>
<td>- When are you done?</td>
</tr>
<tr>
<td>- Slower convergence for many ML problems</td>
<td>- Must implement scheduler over vertices</td>
</tr>
<tr>
<td>- In matrix-land, called Jacobi Iteration</td>
<td>- Faster convergence for many ML problems</td>
</tr>
<tr>
<td>- Implemented by Google Pregel 2010</td>
<td>- In matrix-land, called Gauss-Seidel Iteration</td>
</tr>
<tr>
<td></td>
<td>- Implemented by GraphLab 2010, 2012</td>
</tr>
</tbody>
</table>