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

- Reduce: Take all values associated with a key and aggregate
  - Aggregate values for each key
  - Must be commutative-associate operation
  - Data-parallel over keys
  - Generate (key,value) pairs

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

©Carlos Guestrin 2013
Map-Reduce – Execution Overview

Map Phase | Shuffle Phase | Reduce Phase
---|---|---

Big Data

- Split data across machines
- Assign tuple \((k, v)\) to machine \(h[k]\)

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?

\[ \epsilon_t = L_{u}^{(t)} \cdot R_{v}^{(t)} - r_{uv} \]

\[
\begin{bmatrix}
L_{u}^{(t+1)} \\
R_{v}^{(t+1)}
\end{bmatrix}
\leftarrow
\begin{bmatrix}
(1 - \eta_t \lambda_u)L_{u}^{(t)} - \eta_t \epsilon_t R_{v}^{(t)} \\
(1 - \eta_t \lambda_v)R_{v}^{(t)} - \eta_t \epsilon_t L_{u}^{(t)}
\end{bmatrix}
\]

- 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

Women on the Verge of a Nervous Breakdown
The Celebration
City of God
Wild Strawberries
La Dolce Vita

©Carlos Guestrin 2013
Flashback to 1998

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

### Graphs encode the relationships between:
- **People**
- **Facts**
- **Products**
- **Interests**
- **Ideas**

**Big:** 100 billions of **vertices** and **edges** and rich metadata
- Facebook (10/2012): 1B users, 144B friendships
- Twitter (2011): 15B follower edges

<table>
<thead>
<tr>
<th>Social Media</th>
<th>Science</th>
<th>Advertising</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="facebook.png" alt="Facebook" /></td>
<td><img src="science.png" alt="Science" /></td>
<td><img src="advertising.png" alt="Advertising" /></td>
<td><img src="web.png" alt="Web" /></td>
</tr>
</tbody>
</table>

©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

©Carlos Guestrin 2013
Example: Estimate Political Bias

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

Data-Parallel → Graph-Parallel

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

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

Example of a Graph-Parallel Algorithm
What’s the rank of this user?

Depends on rank of who follows her

Depends on rank of who follows them...

Loops in graph → Must iterate!

PageRank

α is the random reset probability

\[ w_{ji} \] is the prob. transitioning (similarity) from \( j \) to \( i \)

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

- Dependency Graph
- Local Updates
- Iterative Computation

Addressing Graph-Parallel ML

- Graph-Parallel Abstraction
  - Graph-Parallel
  - Map Reduce
  - Data-Parallel

- Feature Extraction
- Cross Validation
- Computing Sufficient Statistics

- Graphical Models
  - Gibbs Sampling
  - Belief Propagation
  - Variational Opt.

- Semi-Supervised Learning
  - Label Propagation
  - CoEM

- Collaborative Filtering
  - Tensor Factorization

- Data-Mining
  - PageRank
  - Triangle Counting
Graph Computation:

Synchronous

v.

Asynchronous

Bulk Synchronous Parallel Model: Pregel (Giraph)

[Valiant ‘90]
Map-Reduce – Execution Overview

Map Phase

Shuffle Phase

Reduce Phase

BSP – Execution Overview

Compute Phase

Communicate Phase
Bulk synchronous parallel model provably inefficient for some ML tasks

Analyzing Belief Propagation
[Gonzalez, Low, G. '09]

focus here

important influence

Priority Queue
Smart Scheduling

©Carlos Guestrin 2013
Asynchronous Belief Propagation

Challenge = Boundaries

Algorithm identifies and focuses on hidden sequential structure

Graphical Model

BSP ML Problem:
Synchronous Algorithms can be Inefficient

Theorem:
Bulk Synchronous BP $O(#vertices)$ slower than Asynchronous BP

©Carlos Guestrin 2013
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>- Race conditions can happen all the time</td>
</tr>
<tr>
<td>- All messages are sent</td>
<td>- Must protect against this issue</td>
</tr>
<tr>
<td>- Simpler to build, like Map-Reduce</td>
<td>- More complex fault tolerance</td>
</tr>
<tr>
<td>- No worries about race conditions, barrier guarantees data consistency</td>
<td>- When are you done?</td>
</tr>
<tr>
<td>- Simpler to make fault-tolerant, save data on barrier</td>
<td>- Must implement scheduler over vertices</td>
</tr>
<tr>
<td>- Slower convergence for many ML problems</td>
<td>- Faster convergence for many ML problems</td>
</tr>
<tr>
<td>- In matrix-land, called Jacobi Iteration</td>
<td>- In matrix-land, called Gauss-Seidel Iteration</td>
</tr>
<tr>
<td>- Implemented by Google Pregel 2010</td>
<td>- Implemented by GraphLab 2010, 2012</td>
</tr>
</tbody>
</table>

Case Study 4: Collaborative Filtering

GraphLab

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

- Know how to solve ML problem on 1 machine
- Efficient parallel predictions

---

**Data Graph**

Data associated with vertices and edges

- **Graph:** Social Network
- **Vertex Data:**
  - User profile text
  - Current interests estimates
- **Edge Data:** Similarity weights

---

©Carlos Guestrin 2013
How do we program graph computation?

“Think like a Vertex.”
-Malewicz et al. [SIGMOD’10]

Update Functions
User-defined program: applied to vertex transforms data in scope of vertex

\[
\text{pagerank}(i, \text{scope})\{
\}
\]
Update Function Example: Connected Components

The scheduler determines order vertices are updated

©Carlos Guestrin 2013
Example Schedulers

- Round-robin
- Selective scheduling (skipping):
  - round robin but jump over un-scheduled vertex
- FIFO
- Prioritize scheduling
  - Hard to implement in a distributed fashion
    - Approximations used (each machine has its own priority queue)

Ensuring Race-Free Code

How much can computation overlap?
Need for Consistency?

- **Higher Throughput** (#updates/sec)
- **No Consistency**
- **Potentially Slower Convergence of ML**

GraphLab Ensures **Sequential Consistency**

For each parallel execution, there exists a sequential execution of update functions which produces the same result.
Consistency in Collaborative Filtering

![Graph showing train RMSE vs updates for consistent and inconsistent updates.]

Netflix data, 8 cores

©Carlos Guestrin 2013

The GraphLab Framework

- Graph Based
  - Data Representation
- Update Functions
  - User Computation

Scheduler

Consistency Model

©Carlos Guestrin 2013
Triangle Counting in Twitter Graph

Total: 34.8 Billion Triangles

40M Users
1.2B Edges

Hadoop
1536 Machines
423 Minutes

GraphLab
64 Machines, 1024 Cores
1.5 Minutes

CoEM (Jones et al., 2005)

Named Entity Recognition Task

Is “Dog” an animal?
Is “Catalina” a place?

Vertices: 2 Million
Edges: 200 Million

©Carlos Guestrin 2013
### Never Ending Learner Project (CoEM)

<table>
<thead>
<tr>
<th>Method</th>
<th>Cores</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>Distributed GraphLab</td>
<td>32</td>
<td>80 secs</td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2013

### What you need to know…

- Data-parallel versus graph-parallel computation
- Bulk synchronous processing versus asynchronous processing
- GraphLab system for graph-parallel computation
  - Data representation
  - Update functions
  - Scheduling
  - Consistency model

©Carlos Guestrin 2013