Case Study 4: Collaborative Filtering

Graph-Parallel Problems

Synchronous v. Asynchronous Computation

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Carlos Guestrin, guest lecturer
May 14th, 2015

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

©Emily Fox 2015
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:
  - Data-parallel over elements, e.g., documents
  - Generate (key, value) pairs
    - "value" can be any data type
  - Transform a data element

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

Example:

- Map:
  - word count
  - map (document) for word in doc
  - emit (word, 1)

- Reduce:
  - reduce (word, count+1)
  - for i in count
  - 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!

©Emily Fox 2015
Map-Reduce – Execution Overview

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?

\[
\begin{bmatrix}
L_u^{(t+1)} \\
R_v^{(t+1)}
\end{bmatrix}
\leftarrow
\begin{bmatrix}
(1 - \eta_t \lambda_u) L_u^{(t)} - \eta_t \epsilon_t R_v^{(t)} \\
(1 - \eta_t \lambda_v) R_v^{(t)} - \eta_t \epsilon_t L_u^{(t)}
\end{bmatrix}
\]

\[
\epsilon_t = L_u^{(t)} \cdot R_v^{(t)} - r_{uv}
\]

- Map and Reduce functions???

- Map-Reduce:
  - Data-parallel over all mappers
  - Data-parallel over reducers with same key

- Here, one update at a time!
Flashback to 1998

Why?

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

Social Media  Science  Advertising  Web

• Graphs encode the relationships between:

  People  Products  Ideas
  Facts  Interests

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

©Emily Fox 2015
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*

Topic Modeling (e.g., LDA)
ML Tasks Beyond Data-Parallelism

Map Reduce

Data-Parallel

Graph-Parallel

<table>
<thead>
<tr>
<th>Feature Extraction</th>
<th>Cross Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing Sufficient Statistics</td>
<td></td>
</tr>
</tbody>
</table>

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
PageRank

What’s the rank of this user?

Rank?

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

©Emily Fox 2015
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

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

- Semi-Supervised Learning
  - Label Propagation
  - CoEM

- Collaborative Filtering
  - Tensor Factorization

- Data-Mining
  - PageRank
  - Triangle Counting

©Emily Fox 2015
Graph Computation:

**Synchronous**

v.

**Asynchronous**

Bulk Synchronous Parallel Model: Pregel (Giraph)

[Valiant '90]
Map-Reduce – Execution Overview

Map Phase

- Split data across machines
- (k₁, v₁)
- (k₂, v₂)
- ...

Shuffle Phase

- Assign tuples (k, v) to machine h[k]
- (k₁, v₁)
- (k₂, v₂)
- ...

Reduce Phase

- (k₁, v₁)
- (k₂, v₂)
- ...

BSP – Execution Overview

Compute Phase

- (vid₁, vid₂, v₁)
- (vid₂, vid₂, v₂)
- ...

Communicate Phase

- Message machine for every edge (vid, vid', val)
- (vid₁, vid', v₁)
- (vid₂, vid', v₂)
- ...

Split graph across machines
Bulk synchronous parallel model provably inefficient for some ML tasks

Analyzing Belief Propagation

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

Challenge = Boundaries

Synthetic Noisy Image

Cumulative Vertex Updates

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
Synchronous v. Asynchronous

- Bulk synchronous processing:
  - Computation in phases
    - All vertices participate in a phase
    - Though OK to say no-op
    - All messages are sent
  - Simpler to build, like Map-Reduce
    - No worries about race conditions, barrier guarantees data consistency
    - Simpler to make fault-tolerant, save data on barrier
  - Slower convergence for many ML problems
  - In matrix-land, called Jacobi iteration
  - Implemented by Google Pregel 2010

- Asynchronous processing:
  - Vertices see latest information from neighbors
    - Most closely related to sequential execution
  - Harder to build:
    - Race conditions can happen all the time
    - More complex fault tolerance
    - When are you done?
    - Must implement scheduler over vertices
  - Faster convergence for many ML problems
  - In matrix-land, called Gauss-Seidel Iteration
  - Implemented by GraphLab 2010, 2012

Case Study 4: Collaborative Filtering

GraphLab

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Carlos Guestrin, guest lecturer
May 14th, 2015
Data Graph

Data associated with vertices and edges

Graph:
• Social Network

Vertex Data:
• User profile text
• Current interests estimates

Edge Data:
• Similarity weights

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

Example:
Connected Components
Update Function Example: Connected Components

The scheduler determines order vertices are updated

©Emily Fox 2015
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 \textit{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.

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

Parallel

CPU 1

CPU 2

Sequential

Single CPU
Consistency in Collaborative Filtering

![Graph showing Train RMSE over Updates and Millions](image)

- Inconsistent updates
- Consistent updates

Netflix data, 8 cores

The GraphLab Framework

- Graph Based Data Representation
- Update Functions User Computation
- Scheduler
- Consistency Model

©Emily Fox 2015
Triangle Counting in Twitter Graph

Total:
34.8 Billion Triangles

- 40M Users
- 1.2B Edges

Hadoop results from [Suri & Vassilvitskii ’11]

CoEM (Jones et al., 2005)

Named Entity Recognition Task

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

dog <X> ran quickly

Australia travelled to <X>

Catalina Island <X> is pleasant
Never Ending Learner Project (CoEM)

**Vertices:** 2 Million

**Edges:** 200 Million

<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</td>
<td>32 EC2 machines</td>
<td>80 secs</td>
</tr>
</tbody>
</table>

What do I recommend???

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

©Emily Fox 2015
Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)

\[ \mathbf{X} = \mathbf{L} \mathbf{R}' \]

Rows index movies
Columns index users

Matrix Completion as a Graph

\[ \mathbf{X} = \]

- \( X_{ij} \) known for black cells
- \( X_{ij} \) unknown for white cells
- Rows index users
- Columns index movies
Coordinate Descent for Matrix Factorization: Alternating Least-Squares

\[
\min_{L,R} \sum_{(u,v) : r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\| + \lambda_v \|R\|
\]

- **Fix movie factors, optimize for user factors**
  - Independent least-squares over users
  \[
  \min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\|
  \]

- **Fix user factors, optimize for movie factors**
  - Independent least-squares over movies
  \[
  \min_{R_u} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2 + \lambda_v \|R\|
  \]

- System may be underdetermined:
  - Use regularization
  - Converges to local optima

---

Alternating Least Squares Update Function

\[
\min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 \quad \text{and} \quad \min_{R_u} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2
\]
SGD for Matrix Factorization in GraphLab

\[ e_t = L_u^{(t)} \cdot R_v^{(t)} - r_{uv} \]

\[
\begin{bmatrix}
L_u^{(t+1)} \\
R_v^{(t+1)}
\end{bmatrix} \leftarrow \begin{bmatrix}
(1 - \eta_t \lambda_u) L_u^{(t)} - \eta_t e_t R_v^{(t)} \\
(1 - \eta_t \lambda_v) R_v^{(t)} - \eta_t e_t L_u^{(t)}
\end{bmatrix}
\]

Out-of-core computation

- Often data doesn’t fit in memory
  - Must use disk/SSDs

- Although random accesses to disk are very slow, high performance is possible with optimized memory layout.
  Implemented in:
  - GraphChi
  - GraphLab Create

- Example performance of GraphLab Create:
  - Common Crawl Graph (3.5 billion Nodes and 128 billion Edges)
  - PageRank iteration in 9 mins on single, commodity machine
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
- ALS, SGD for matrix factorization in GraphLab

Reading

- Papers under “Case Study IV: Parallel Learning with GraphLab”
- Optional:
  - Parallel Splash BP
  http://www.ml.cmu.edu/research/dap-papers/dap-gonzalez.pdf