Case Study 4: Collaborative Filtering

Graph-Parallel Problems

Synchronous v. Asynchronous Computation

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
February 25th, 2014

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

- **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
  - Example: word count
    - `reduce(word, count, list)`
    - `C = 0`
    - 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!
Map-Reduce – Execution Overview

**Map Phase**
- Split data across machines

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

**Reduce Phase**
- Aggregate key-val pairs

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?

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

- 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

©Emily Fox 2014
Flashback to 1998

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

Why?

used to be popular
transitioned to

Facebook Graph

Data model

Objects & Associations

name: Barack Obama
birthday: 11/06/1961
website: http://...

verified: 1

© Emily Fox 2014

Slide from Facebook Engineering presentation
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)**

- Cat
- Apple
- Growth
- Hat
- Plant

**ML Tasks Beyond Data-Parallelism**

- Data-Parallel
  - Map Reduce
  - Feature Extraction
  - Cross Validation
  - Computing Sufficient Statistics
- Graph-Parallel
  - 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?

Depends on rank of who follows her

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 probability transitioning (similarity) from \( j \) to \( i \)

Properties of Graph Parallel Algorithms

Dependency Graph

Local Updates

Iterative Computation

\( \text{My Rank} \)

\( \text{Friends Rank} \)
Addressing Graph-Parallel ML

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

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

Graph Computation:

* Synchronous

* Asynchronous
Bulk Synchronous Parallel Model: Pregel (Giraph)

Compute

Communicate

Map-Reduce – Execution Overview

Map Phase

Shuffle Phase

Reduce Phase

© Emily Fox 2014
Bulk synchronous parallel model
provably inefficient
for some ML tasks
Analyzing Belief Propagation

[1] Gonzalez, Low, G. ’09

Asynchronous Belief Propagation

Challenge = Boundaries

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

- **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
      - Must protect against this issue
    - 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

The **GraphLab** Goals

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

GraphLab

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
February 25th, 2014

©Emily Fox 2014
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 \textbf{vertex} transforms data in \textit{scope} of vertex

\texttt{pagerank(i, scope)\{ [diagram] \}}

Update Function Example: Connected Components

[diagram]
Update Function Example: Connected Components

The Scheduler

The scheduler determines order vertices are updated
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

Train RMSE vs Updates (Millions)

- Inconsistent updates
- Consistent updates

Netflix data, 8 cores
©Emily Fox 2014

The GraphLab Framework

Graph Based
Data Representation

Update Functions
User Computation

Scheduler

Consistency Model

©Emily Fox 2014
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

Hadoop results from [Suri & Vassilvitskii '11]

©Emily Fox 2014

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

©Emily Fox 2014
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>

©Emily Fox 2014
Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)

\[ X = L \cdot R' \]

- \( r_{uv} = L_u \cdot R_v \)
- Movie topic \( i \) is "romance"
- User \( u \) likes topic \( i \)
- Movie \( v \) is about topic \( i \)

Matrix Completion as a Graph

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

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

- Fix movie factors, optimize for user factors
  - Independent least-squares over users
  \[
  \min_{L_u} \sum_{v \in V_u} \left( L_u \cdot R_v - r_{uv} \right)^2 + \lambda_u \|L \|
  \]
- Fix user factors, optimize for movie factors
  - Independent least-squares over movies
  \[
  \min_{R_v} \sum_{u \in U_v} \left( L_u \cdot R_v - r_{uv} \right)^2 + \lambda_v \|R \|
  \]
- System may be underdetermined: use regularization
- Converges to local optima

Alternating Least Squares Update Function

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

\[ \epsilon_t = L^{(t)}_u \cdot R^{(t)}_v - r_{uv} \]

Latent user and movie factors:

- Observations
- Hyperparameters:
  - Place priors on \( \phi \)

Want to predict new movie rating:

\[ p(r_{uv} | X, \phi) = \int p(r_{uv} | L_u, R_v) p(L, R | X, \phi) \, dL \, dR \]
Bayesian PMF Gibbs Sampler

- Outline of Bayesian PMF sampler

1. Init $L^{(0)}$, $R^{(0)}$
2. For $k=1,\ldots,Niter$
   (i) Sample hyperparams $\phi^{(k)}$, $\beta^{(k)}$, $\phi^{(k)}$
   (ii) For each user $u=1,\ldots,n$ sample in parallel
        $L^{(k+1)}_u \sim P(L_u | X, R^{(k)}, \phi^{(k)})$
   (iii) For each movie $v=1,\ldots,m$ sample in parallel
        $R^{(k+1)}_{v} \sim P(R_v | X, L^{(k+1)}, \phi^{(k)})$

Very similar to ideas of ALS (systematically)

Bayesian PMF Example

- For user $u$:
  $p(L_u | X, R, \phi_u) \propto p(L_u | \phi_u) \prod_{v \in V_u} p(r_{uv} | L_u, R_v, \phi_R)

\begin{align*}
\text{log prior: } & N(\mu_u, \Sigma_u) \prod_{v \in V_u} N(c_{uv} | \mu_u, \Sigma_u) \\
\text{log likelihood: } & N(\mu_u, \Sigma_u) \prod_{v \in V_u} N(c_{uv} | \mu_u, \Sigma_u)
\end{align*}

where $\hat{\Sigma}_u^{-1} = \Sigma_u^{-1} + \sigma_v^2 \Sigma_{R_v R_v}$

Symmetrically for $R_v$ conditioned on $L$ (breaks down over movies)

Luckily, we can use this to get our desired posterior samples
PMF Gibbs Sampling in GraphLab

\[ p(L_u | X, R, \phi_u) = \mathcal{N}(\mu_u, \Sigma_u) \]

\[ \Sigma_u = \Sigma_u^{-1} + \sigma^2 \sum_{v \in V_u} R_u R_v \]

\[ \mu_u = \Sigma_u \left( \sigma^2 \sum_{v \in V_u} r_{uv} R_v + \mu_v \right) \]
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 and Gibbs for matrix factorization/PMF in GraphLab

Reading

- Papers under “Case Study IV: Parallel Learning with GraphLab”
- Optional:
  - Parallel Splash BP
Acknowledgements

- Slides based on Carlos Guestrin’s GraphLab talk