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

<table>
<thead>
<tr>
<th>Map:</th>
<th>Transforms a data element</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-parallel over elements, e.g., documents</td>
</tr>
<tr>
<td></td>
<td>Generate (key, value) pairs</td>
</tr>
<tr>
<td></td>
<td><em>value</em> can be any data type</td>
</tr>
</tbody>
</table>

```
Let's example some data where each document is represented by a list of counts for each word.
We map over each document, generating key-value pairs (word, count).
```

<table>
<thead>
<tr>
<th>Reduce:</th>
<th>Aggregate values for each key</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Must be commutative-associate operation</td>
</tr>
<tr>
<td></td>
<td>Data-parallel over keys</td>
</tr>
<tr>
<td></td>
<td>Generate (key, value) pairs</td>
</tr>
</tbody>
</table>

```
We reduce over the key, aggregating the counts for each word.
Finally, we emit the word and its total count.
```

- 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

Reduce Phase

Shuffle Phase

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!

Matrix Factorization as a Graph
Flashback to 1998

Why?

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

"Page rank" -> see this later

Facebook Graph

Data model
Objects & Associations

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*
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

"but many graph-structured problems"
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 prob. transitioning (similarity) from \( j \) to \( i \)

\[ R[0] = 0.15 + 0.85(0.2 R[1] + 0.5 R[2] + 0.3 R[3]) \]

\( \alpha \) is the prob. of landing on 0 directly

Properties of Graph Parallel Algorithms

Dependency Graph

Local Updates

Iterative Computation

- My Rank
- Friends Rank

algorithm shown on prev. slide

guaranteed to conv. for page rank
Addressing Graph-Parallel ML

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

Graph-Parallel Abstraction

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]

©Emily Fox 2014

**Map-Reduce – Execution Overview**

©Emily Fox 2014

1. **Map Phase**
   - (k₁,v₁)
   - (k₂,v₂)
   - ...

2. **Shuffle Phase**
   - Assign tuple (k,v) to machine [hash on key]

3. **Reduce Phase**
   - (k₁,v₁)
   - (k₂,v₂)
   - ...

**Big Data**

Split data across machines
BSP – Execution Overview

Compute Phase

Communicate Phase

Big Graph

Split graph across machines

Message machine for every edge (vid, vid', val)

Bulk synchronous parallel model provably inefficient for some ML tasks
Analyzing Belief Propagation

[gonzalez, low, g. '09]

Focus here

Specific case of importance of async. comp.

Important influence

Focus our updating here; want to discover this path

Asynchronous Belief Propagation

Challenge = Boundaries

Synthetic Noisy Image

Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

© Emily Fox 2014
BSP ML Problem: BP on real data

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

---

©Emily Fox 2014
Case Study 4: Collaborative Filtering

GraphLab

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

The GraphLab Goals

Know how to solve ML problem on 1 machine

Efficient parallel predictions

4 components:
- Data graph
- Update function
- Scheduler
- Consistency model
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

Update Function Example:
Connected Components

1. Initialize all nodes w/ unique labels
2. Pick a node:
   component = min (self, neighbors)
3. Return to step 2
Update Function Example: Connected Components

- Label of node $i$
- Initial: $\text{component}[i] = i$
- Update ($i$, scope)
  - $\text{component}[x] = \min\{\text{component}[x], \min_{j \in \text{neigh}(i)} \text{component}[j]\}$
  - All in "scope"

The Scheduler

The **scheduler** determines order vertices are updated.

Scheduler
Example Schedulers

- Round-robin
- Selective scheduling (skipping):
  - round robin but jump over un-scheduled vertex
- FIFO
- Prioritize scheduling (e.g. splash BP)
  - 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?

Can do updates together if scopes don’t overlap.

In some cases, you don’t change your neighbors (e.g., PageRank)
but in other cases like BP you do.
Need for Consistency?

Higher Throughput (#updates/sec)

No Consistency

Potentially Slower Convergence of ML

©Emily Fox 2014

GraphLab Ensures **Sequential Consistency**

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

©Emily Fox 2014
Consistency in Collaborative Filtering

Graph Lab Framework

Scheduler

Graph Based Data Representation

Update Functions

User Computation

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

CoEM (Jones et al., 2005)

Named Entity Recognition Task

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

want to learn
distribution over
animal, place, ... for this noun

label nodes
initialize (randomly, smartly)
iterative, EM-like alg.

©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 GraphLab</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
Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)

\[ X = LR' \]

\[ r_{uv} \approx L_u R_v \]

Rows index movies
Columns index users

Matrix Completion as a Graph

\[ X = \]

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

\[
\min_{L,R} \sum_{(u,v): r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\|_1 + \lambda_v \|R\|_1
\]

- Fix movie factors, optimize for user factors
  - Independent least-squares over users
    \[
    \min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\|_1
    \]
- Fix user factors, optimize for movie factors
  - Independent least-squares over movies
    \[
    \min_{R_v} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2 + \lambda_v \|R\|_1
    \]
- System may be underdetermined: use regularization
- Converges to local optima

Alternating Least Squares Update Function

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

\[ e_t = L_{(t)}^{(t+1)} \cdot R_{(t+1)}^{(t)} - u_{uv} \]

Bayesian PMF Example

- Latent user and movie factors:
  \[ L_u \sim N(m_u, \Sigma_u) \quad u=1,\ldots,n \]
  \[ R_v \sim N(m_v, \Sigma_v) \quad v=1,\ldots,m \]

- Observations
  \[ c_{uv} \sim N(L_u R_v, \sigma^2) \]

- Hyperparameters:
  \[ \phi = \{ m_u, \Sigma_u, m_v, \Sigma_v, \sigma^2 \} \]

- Want to predict new movie rating:
  \[ p(r_{uv}^*| x, \phi) = \int p(r_{uv}^*| L_u R_v) p(L, R|x, \phi) dL dR \]
  \[ = \int p(r_{uv}^*| u_{uv}) p(L, R|x, \phi) dL dR \]
Bayesian PMF Gibbs Sampler

- Outline of Bayesian PMF sampler
  1. \[ \text{Init } L^{(0)}, R^{(0)} \]
  2. For \[ k=1, \ldots, \text{Niter} \]
     (i) Sample hyperparams \[ \phi^{(k)}, \Sigma^{(k)} \in \mathbb{R}^n \]
     (ii) For each user \( u = 1, \ldots, n \) sample in parallel
         \[ L^{(k+1)} \sim P(L_u | X, R^{(k)}, \phi^{(k)}) \]
     (iii) For each movie \( v = 1, \ldots, m \) sample in parallel
         \[ R_v^{(k+1)} \sim P(R_v | X, L^{(k+1)}, \phi^{(k)}) \]

Very similar to ideas of ALS (systematically)

Bayesian PMF Example

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

  \[
  \propto \mathcal{N}(L_u | \Sigma_u, \Sigma_u) \prod_{v \in V_u} \mathcal{N}(r_{uv} | L_u R_v, \sigma_r^2)
  \]

  \[
  \sim \mathcal{N}(L_u | \Sigma_u, \Sigma_u) \quad \text{via conjugacy}
  \]

  where

  \[
  \Sigma_u^{-1} = \Sigma_u^{-1} + \sigma_r^2 \Sigma_v \quad \text{(updates)}
  \]

  \[
  \Sigma_u' = \Sigma_u (\sigma_u^2 \Sigma_{uv} R_v + \Sigma_u M_u)
  \]

  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

\[
p(L_u \mid X, R, \phi_u) = N(\mu_u, \Sigma_u) \quad \Sigma_u = \Sigma_u^{-1} + \sigma^2 \sum_{v \in V_u} R_u R_v^T \quad \mu_u = \mu_u \left( \sigma^2 \sum_{v \in V_u} r_{uv} R_v + \Sigma_u \mu_u \right)
\]
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
  - [http://www.ml.cmu.edu/research/dap-papers/dap-gonzalez.pdf](http://www.ml.cmu.edu/research/dap-papers/dap-gonzalez.pdf)
Acknowledgements

- Slides based on Carlos Guestrin’s GraphLab talk