Case Study 4: Collaborative Filtering

GraphLab

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington
Carlos Guestrin
March 14th, 2013

©Carlos Guestrin 2013

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
Facebook Graph

Data model
Objects & Associations

Addressing Graph-Parallel ML

Map Reduce

Graph-Parallel Abstraction

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
Asynchronous Belief Propagation

**Challenge = Boundaries**

- Synthetic Noisy Image
- Cumulative Vertex Updates
- Algorithm identifies and focuses on hidden sequential structure

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

©Carlos Guestrin 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

```c
pagerank(i, scope){
}
```
Connected Components

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**

- Inconsistent updates
- Consistent 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

1536 Machines
423 Minutes

64 Machines, 1024 Cores
1.5 Minutes

©Carlos Guestrin 2013

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/Facilities</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>95 Cores</td>
<td>7.5 hrs</td>
</tr>
<tr>
<td>Distributed</td>
<td>32 EC2 machines</td>
<td>80 secs</td>
</tr>
<tr>
<td>GraphLab</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

©Carlos Guestrin 2013
Interpreting Low-Rank Matrix Completion

(aka Matrix Factorization)

\[ X = L R' \]
Matrix Completion as a Graph

\[ X = \]

\( X_{ij} \) known for black cells
\( X_{ij} \) unknown for white cells
Rows index movies
Columns index movies

Coordinate Descent for Matrix Factorization: Alternating Least-Squares

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

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

\[ \min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 \quad \min_{R_v} \sum_{u \in U_u} (L_u \cdot R_v - r_{uv})^2 \]

SGD for Matrix Factorization in Map-Reduce?

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

\[
\begin{bmatrix}
L_u^{(t+1)} \\
R_v^{(t+1)}
\end{bmatrix}
\leftarrow
\begin{bmatrix}
(1 - \eta_t \lambda_u)L_u^{(t)} - \eta_t \epsilon_t R_v^{(t)} \\
(1 - \eta_t \lambda_v)R_v^{(t)} - \eta_t \epsilon_t L_u^{(t)}
\end{bmatrix}
\]
GraphChi: Going small with GraphLab

Solve huge problems on small or embedded devices?

Key: Exploit non-volatile memory (starting with SSDs and HDs)

GraphChi – disk-based GraphLab

Challenge: Random Accesses

Novel GraphChi solution: Parallel sliding windows method  
minimizes number of random accesses
Naive Graph Disk Layouts

- Symmetrized adjacency file with values,

<table>
<thead>
<tr>
<th>vertex</th>
<th>in-neighbors</th>
<th>out-neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3:2.3, 19:1.3, 49:0.65,...</td>
<td>781:2.3, 881:4.2, ...</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3:1.4, 9:12.1,...</td>
<td>5:1.3, 28:2.2,...</td>
</tr>
</tbody>
</table>

- ... or with file index pointers

<table>
<thead>
<tr>
<th>vertex</th>
<th>in-neighbor-ptr</th>
<th>out-neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3:881, 19:10092, 49:20763,...</td>
<td>781:2.3, 881:4.2, ...</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>3:882, 9:2872,...</td>
<td>5:1.3, 28:2.2,...</td>
</tr>
</tbody>
</table>

Parallel Sliding Windows Layout

Shard: in-edges for subset of vertices; sorted by source_id

- Shards small enough to fit in memory; balance size of shards
Parallel Sliding Windows Execution

Load subgraph for vertices 1..100

Vertices 1..100
Vertices 101..700
Vertices 701..1000
Vertices 1001..10000

in-edges for vertices 1..100 sorted by source_id

Load all in-edges in memory

What about out-edges? Arranged in sequence in other shards! And sequential writes!

Parallel Sliding Windows Execution

Load subgraph for vertices 101..700

Vertices 1..100
Vertices 101..700
Vertices 701..1000
Vertices 1001..10000

in-edges for vertices 1..100 sorted by source_id

Load all in-edges in memory

Only $O(P^2)$ random reads per pass on entire graph
Triangle Counting on Twitter Graph

40M Users
1.2B Edges

Total: 34.8 Billion Triangles

1636 Machines
423 Minutes

59 Minutes, 1 Mac Mini!

64 Machines, 1024 Cores
1.5 Minutes

Hadoop

GraphChi

GraphLab2

©Carlos Guestrin 2013
Hadoop results from [Suri & Vassilvitskii '11]

GraphLab
Release 2.1 available now
http://graphlab.org
Documentation... Code... Tutorials... (more on the way)

GraphChi 0.1 available now
http://graphchi.org
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