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:**
  - Transform a data element
  - Data-parallel over elements, e.g., documents
  - Generate (key, value) pairs
    - "value" can be any data type
  - Example:
    - `((key, value), (key, value))`
  - In our example:
    - `((key, value), (key, value))`

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

©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

©Emily Fox 2015
SGD for Matrix Factorization in Map-Reduce?

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

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

- 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

Women on the Verge of a Nervous Breakdown

La Dolce Vita

City of God

Wild Strawberries

The Celebration

Women on the Verge of a Nervous Breakdown

La Dolce Vita

Wild Strawberries

City of God

The Celebration
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  Facts  Products  Ideas  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

Label a Face and Propagate
Pairwise similarity not enough...

Propagate Similarities & Co-occurrences for Accurate Predictions

- Not similar enough to be sure
- Who???

- Propagate Similarities
- Co-occurring faces
- Further evidence
Example: \textit{Estimate Political Bias}

![Diagram](diagram1.png)

Topic Modeling (e.g., LDA)

![Diagram](diagram2.png)
ML Tasks Beyond Data-Parallelism

Map Reduce

Data-Parallel

Graph-Parallel

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graphical Models

Gibbs Sampling
Belief Propagation
Variational Opt.

Collaborative Filtering
Tensor Factorization

Semi-Supervised Learning

Label Propagation
CoEM

Graph Analysis
PageRank
Triangle Counting

Example of a Graph-Parallel Algorithm
PageRank

What’s the rank of this user?

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 \( i \)

\[ R[0] = 0.15 + (1-0.15)(0.2 R[1] + 0.5 R[2] + 0.3 R[3]) \]
Properties of Graph Parallel Algorithms

Dependence Graph

Local Updates

Iterative Computation

My Rank

Friends Rank

R[0] & weighted avg of neighbors

iterate until converge

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.

Collaborative Filtering

Tensor Factorization

Semi-Supervised Learning

Label Propagation

CoEM

Data-Mining

PageRank

Triangle Counting

©Emily Fox 2015
Key Question.

Graph Computation:

*Synchronous v. Asynchronous*
Map-Reduce – Execution Overview

Map Phase

M1

(k1,v1)

M2

(k2,v2)

…

M1000

(k1000,v1000)

Shuffle Phase

Assign tuple (k,v) to machine h(k)

Reduce Phase

M1

(k1,v1)

M2

(k2,v2)

…

M1000

(k1000,v1000)

BSP – Execution Overview

Compute Phase

M1

(vid1,vid1',v1)

(vid2,vid2',v2)

…

M1000

(vid1000,vid1000',v1000)

Communicate Phase

Message machine for every edge (vid,vid')

©Emily Fox 2015
Bulk synchronous parallel model **provably inefficient** for some ML tasks

Analyzing Belief Propagation

[Gonzalez, Low, G. '09]

©Emily Fox 2015
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 $O(\#\text{vertices})$ slower than Asynchronous BP
Synchronous v. Asynchronous

- **Synchronous processing:**
  - Computation in phases
    - All vertices participate in a phase
      - Even though asynchronous, no-op is supported
    - 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
  - Iterative algorithm, called Jacobi Iteration
  - Implemented by Google's 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
  - Iterative algorithm, 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

\[ R(i) = \alpha + (1-\alpha) \sum_{j \in \text{scope.neighbors}} \text{weight} \]

1. Initialize: vertex.component = vertex.id
2. Iterate:
   - Pick a node i:
     \[ \text{component}(i) = \min(\text{self.neighbors}) \]
     - If change, add neighbors to queue

©Emily Fox 2015
Update Function Example: Connected Components

\[
\text{init: } \text{component}(i) = i
\]

\[
\text{Update } (i, \text{Scope}): \quad \text{component}[i] = \min (\text{component}[i], \\
\min_{j \in \text{neighbors}} \text{component}[j])
\]

if \text{component}(i) changes:
  \{ schedule neighbors of i \}

The Scheduler

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 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
Consistency in Collaborative Filtering

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
1536 Machines
423 Minutes

GraphLab
64 Machines, 1024 Cores
1.5 Minutes

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

semi supervised learning
label a few nodes

dist over type [animal, place,...]

iterate:
EM-like alg
Never Ending Learner Project (CoEM)

Vertices: 2 Million
Edges: 200 Million

<table>
<thead>
<tr>
<th>Tool</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>
<tr>
<td>GraphLab 2012</td>
<td>EC2</td>
<td></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)

\[ X = LR' \]

Matrix Completion as a Graph

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

Movies index via movie topic "action"
Coordinate Descent for Matrix Factorization: Alternating Least-Squares

\[
\min_{L,R} \sum_{(u,v) : r_{uv} \neq 0} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\| + \lambda_v \|R\|
\]

- **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

\[
\min_{R_v} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2 + \lambda_v \|R\|
\]

- System may be underdetermined:
  - Use regularization

- Converges to local optima

---

Alternating Least Squares Update Function

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

1. **Update (L, scope):**
   - **goal:** estimate \( L_u \)
   - **from scope:** gather factors and neighbors
   - **update:**

\[
X = \left[ \begin{array}{c} L_u \end{array} \right]
\]

- **read all ratings for user**

\[
Y = \left[ \begin{array}{c} r_{uv} \end{array} \right]
\]

- **solve a local least squares problem**

\[
L_u = (X^TX + \lambda_u I)^{-1} X^TY
\]
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