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 20th, 2014

ML in the Context of Parallel Architectures

- GPUs
- Multicore
- Clusters
- Clouds
- Supercomputers

But scalable ML in these systems is hard, especially in terms of:
1. Programmability
2. Data distribution
3. Failures

we’ll go through these ideas...
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
  - Example: word count
    - `map(document)` for word in doc
      - `emit(word, 1)`

- **Reduce:**
  - Take all values associated with a key and aggregate
  - Example: `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
  - Map phase produces key-value pairs of the form (k, v)

- **Shuffle Phase**
  - shuffle key-value pairs to their destination
  - (k, v) pairs are grouped by key k

- **Reduce Phase**
  - Reduce phase processes the grouped data
  - (k, v) pairs are aggregated and reduced to produce output

**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?

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

- 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!

Social Media  Science  Advertising  Web

- Graphs encode the relationships between:
  - People
  - Facts
  - Products
  - Interests
  - Ideas

- **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
Example: *Estimate Political Bias*

Topic Modeling (e.g., LDA)
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.

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 prob. transitioning (similarity) from \( j \) to \( i \)
Properties of Graph Parallel Algorithms

Dependency Graph

Local Updates

Iterative Computation

My Rank

Friends Rank

Addressing Graph-Parallel ML

Data-Parallel

Map Reduce

Graph-Parallel

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
Graph Computation:

Synchronous

v.

Asynchronous

Bulk Synchronous Parallel Model:
Pregel (Giraph)

[Valiant '90]
Map-Reduce – Execution Overview

Map Phase  Shuffle Phase  Reduce Phase

Big Data

M1  (k1, v1)  (k2, v2)  ...
M2  (k1', v1')  (k2', v2')  ...
M1000  (k1''', v1''')  (k2''', v2'''')  ...

Split data across machines

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

Shuffle Phase

M1  (k1, v1)  (k2, v2)  ...
M2  (k3, v3)  (k4, v4)  ...
M1000  (k5, v5)  (k6, v6)  ...

BSP – Execution Overview

Compute Phase  Communicate Phase

Big Graph

M1  (vid1, vid1', v1)  (vid2, vid2', v2)  ...
M2  (vid1', vid1, v1')  (vid2', vid2, v2')  ...
M1000  (vid1''', vid1''', v1''')  (vid2''', vid2''', v2'''')  ...

Split graph across machines

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

Communicate Phase
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

Many Updates

Few 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

- **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

Acknowledgements

- Slides based on Carlos Guestrin’s GraphLab talk