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

©Emily Fox 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

©Emily Fox 2014
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(\text{\textit{word}}, \text{\textit{count list}}(\text{\textit{word}}))`
    - `for i in count` `c += count[i]`
    - `emit(\text{\textit{word}}, c)`

- **Reduce:**
  - Aggregate values for each key
  - Must be commutative-associate operation
  - Data-parallel over keys
  - Generate (key, value) pairs
  - Example: word count
    - `reduce(\text{\textit{word}}, [1, 17, 0, 0, 12])`
    - `emit('\text{\textit{UW}}, 30')`

- 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

Big Data

Map Phase

Shuffle Phase

Reduce Phase

Split data across machines

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

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?

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

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

- 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

- Women on the Verge of a Nervous Breakdown
- The Celebration
- City of God
- Wild Strawberries
- La Dolce Vita
Flashback to 1998

Why?

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

"Page rank"

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

more than just friend interactions

node attributes

© Emily Fox 2014