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

use more processors

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...

talked about these ideas w/ MapReduce lecture

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

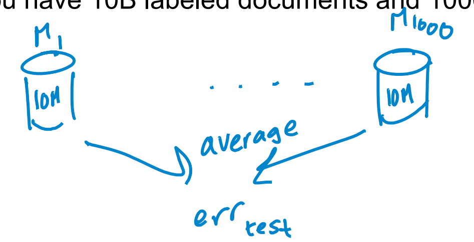
now →

graph-parallel

this quarter

Simplest Type of Parallelism: Data Parallel Problems

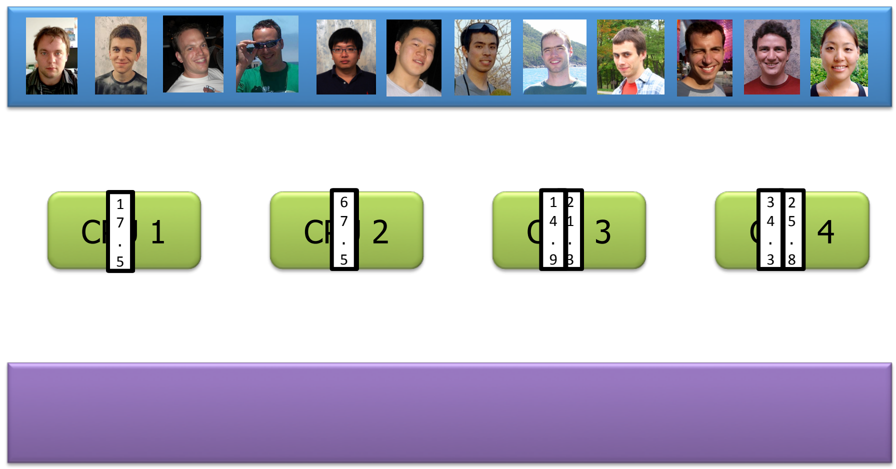
- You have already learned a classifier w^*
  - What's the test error? $err_{test} = \frac{1}{N_{test}} \sum_i | y^{(i)} - sign(w^* \cdot x^{(i)}) |$
- You have 10B labeled documents and 1000 machines



locally computing error on subset of data

- 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~~ prev. 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: Transforms a data element
  - Data-parallel over elements, e.g., documents
  - Generate (key,value) pairs
    - "value" can be any data type

('UW',17)

In this example: ('Mary',1) ('UW',1) ('Mary',1)

Example: word count

map(document)
for word in doc
emit(word,1)

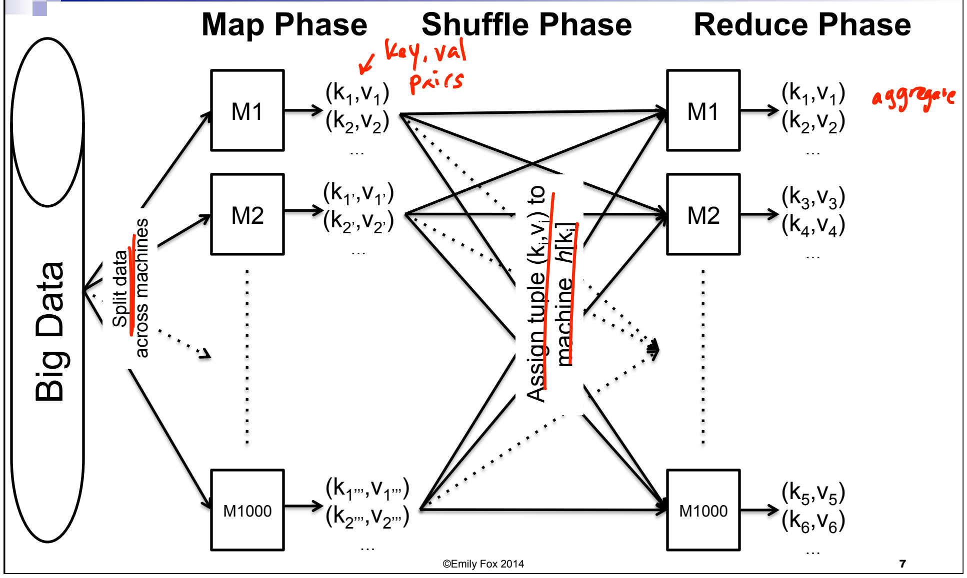
- Reduce: Take all values associated w/ a key and aggregate
  - Aggregate values for each key
  - Must be commutative-associate operation
  - Data-parallel over keys
  - Generate (key,value) pairs

reduce('UW',[1,17,0,0,12])
emit('UW',30)

Reduce(word, counts: list(int))
c=0
for i in counts
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



Issues with Map-Reduce Abstraction

- Often all data gets moved around cluster
  - Very bad for iterative settings

in shuffle phase

- Definition of Map & Reduce functions can be unintuitive in many apps
  - Graphs are challenging
- Computation is synchronous

SGD for Matrix Factorization in Map-Reduce?

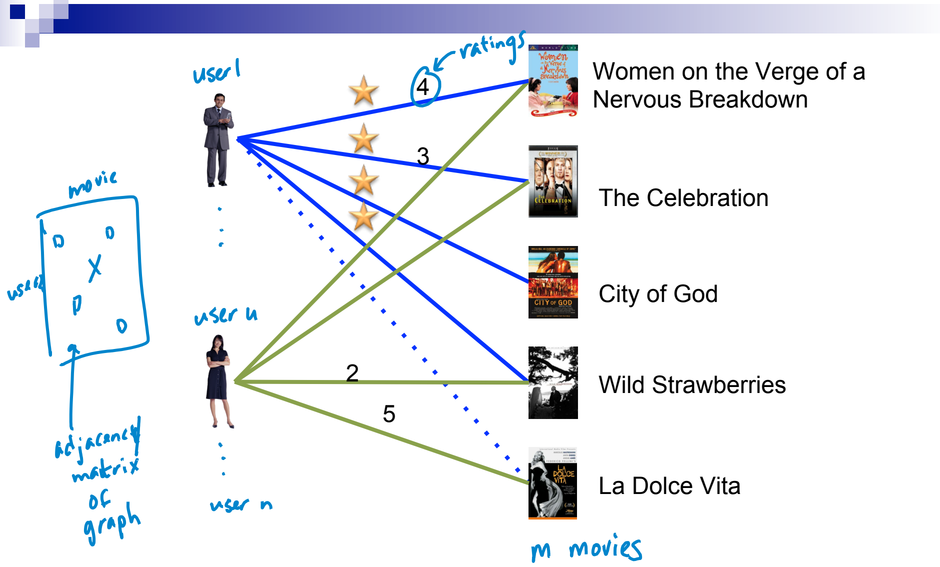
$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1-\eta_t\lambda_u)L_u^{(t)} - \eta_t\epsilon_t R_v^{(t)} \\ (1-\eta_t\lambda_v)R_v^{(t)} - \eta_t\epsilon_t L_u^{(t)} \end{bmatrix}$$

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

- Map and Reduce functions???
- Map-Reduce:
  - Data-parallel over all mappers
  - Data-parallel over reducers with same key
- Here, one update at a time!

does not nicely fit into data parallel setting

Matrix Factorization as a Graph



Flashback to 1998

used to be popular

altavista
SEARCH SOFTWARE

Why?

transitioned to

Google! BETA

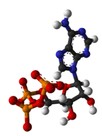
First Google advantage:
a **Graph Algorithm** & a **System to Support** it!

"page rank"

---

graph data is everywhere!

Social Media	Science	Advertising	Web



acm

NETFLIX

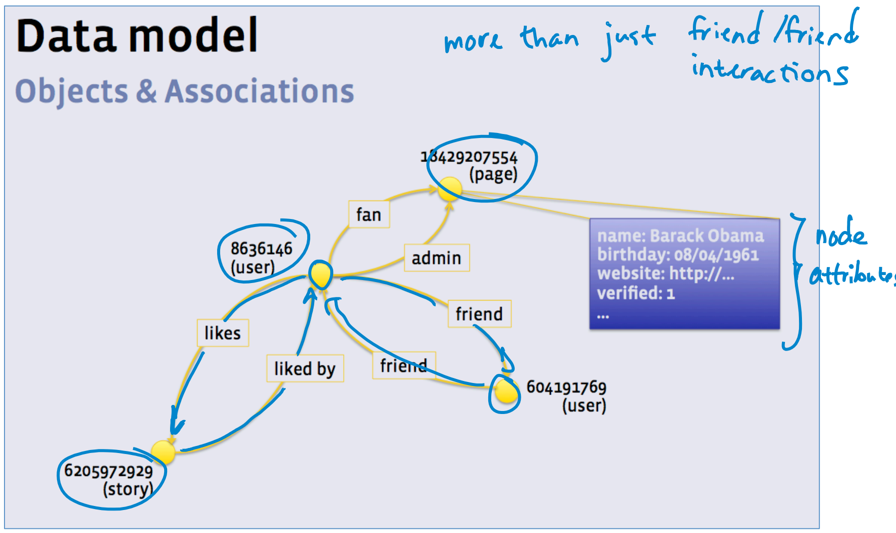
- **Graphs** encode the **relationships** between:

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

12

Facebook Graph



Slide from Facebook Engineering presentation 13