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 25th, 2014

©Emily Fox 2014

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

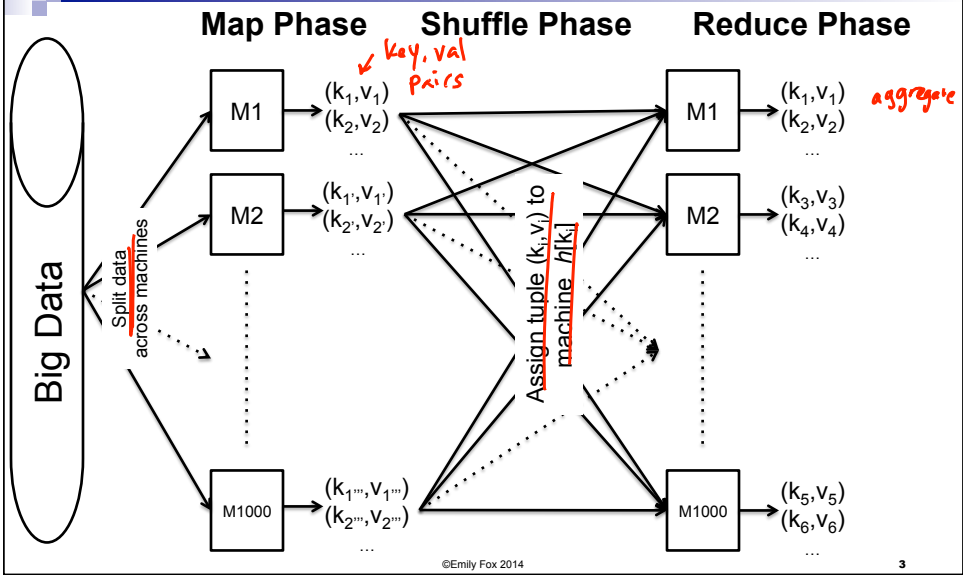
reduce('UW', [1, 17, 0, 0, 12])
emit('UW', 30)

```
Reduce(word, counts=list(int))
  c = 0
  for i in counts
    c += counts[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 2014

Map-Reduce – Execution Overview



©Emily Fox 2014

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

©Emily Fox 2014

SGD for Matrix Factorization in Map-Reduce?

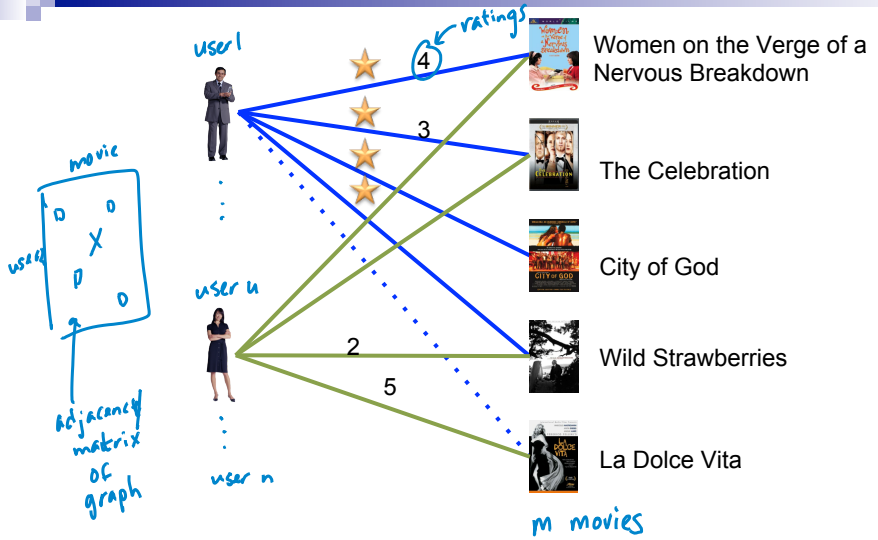
$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1-\eta_t\lambda_u)L_u^{(t)} - \eta_t\epsilon_t R_v^{(t)} \\ (1-\eta_t\lambda_v)R_v^{(t)} - \eta_t\epsilon_t L_u^{(t)} \end{bmatrix}$$

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

- 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

Why?

transitioned to

Google! BETA

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

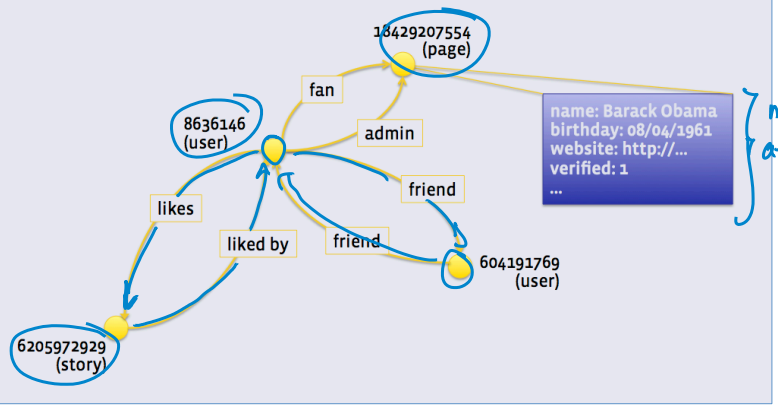
↑ "page rank"

Facebook Graph

Data model

more than just friend/friend interactions

Objects & Associations



Slide from Facebook Engineering presentation

©Emily Fox 2014

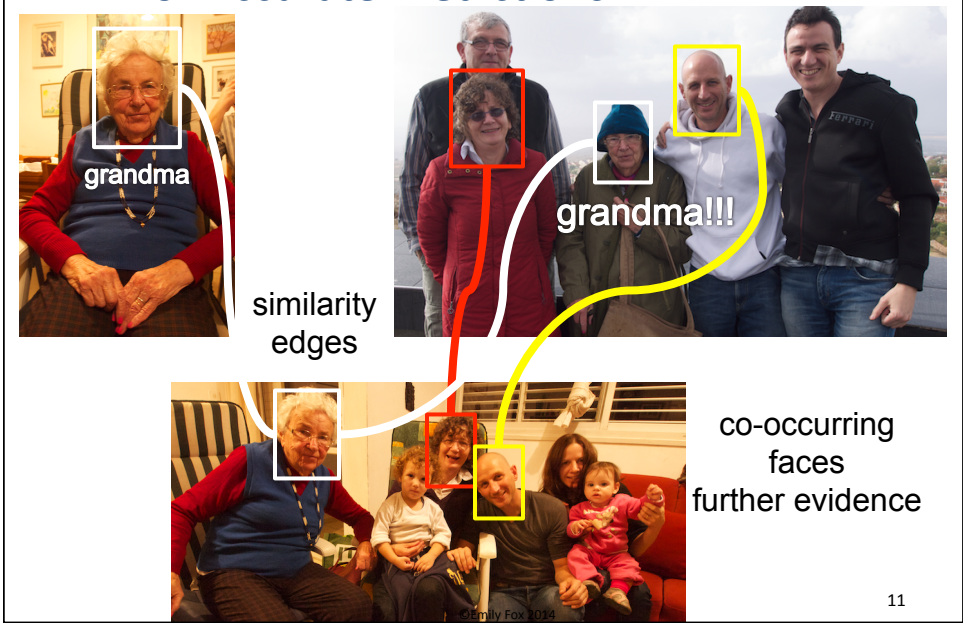
Label a Face and Propagate

©Emily Fox 2014

Pairwise similarity not enough…

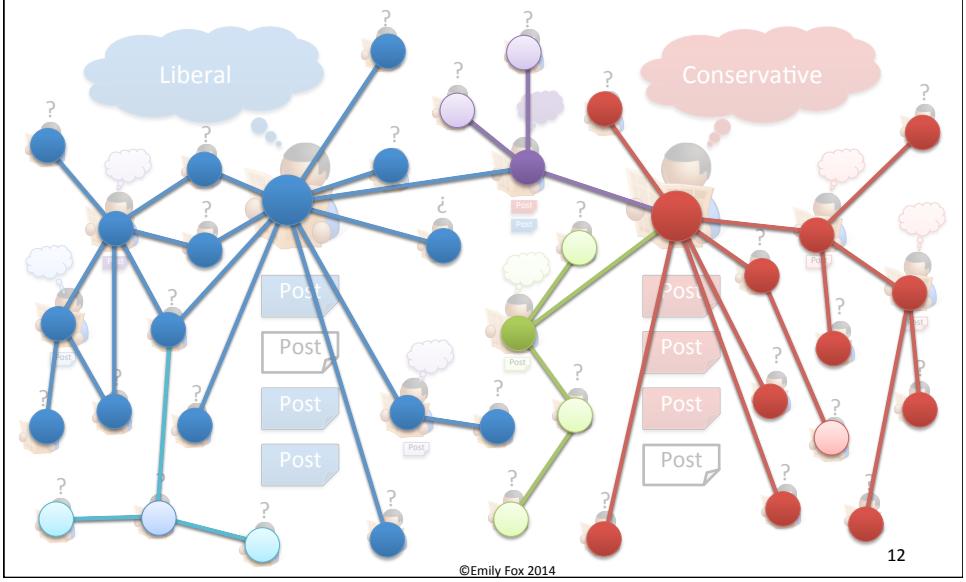
©Emily Fox 2014

Propagate Similarities & Co-occurrences for Accurate Predictions



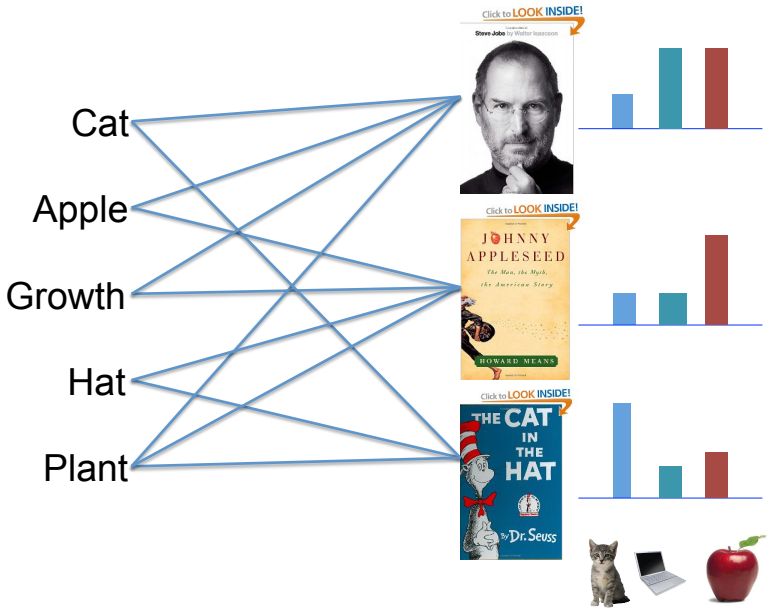
11

Example: *Estimate Political Bias*



12

Topic Modeling (e.g., LDA)



©Emily Fox 2014

ML Tasks Beyond Data-Parallelism

Data-Parallel → Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

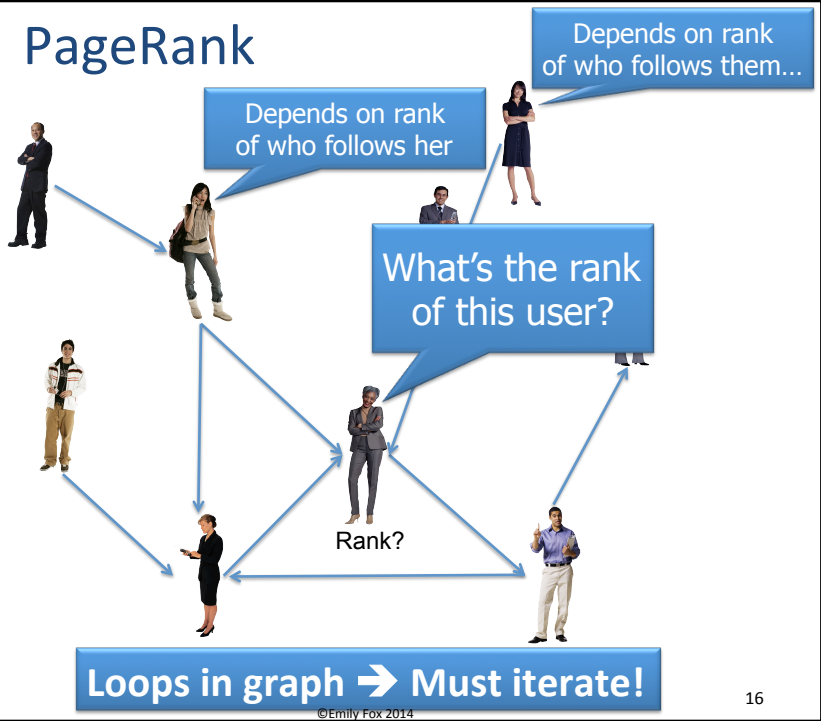
Collaborative Filtering
Tensor Factorization

Graph Analysis
PageRank
Triangle Counting

©Emily Fox 2014

Example of a Graph-Parallel Algorithm

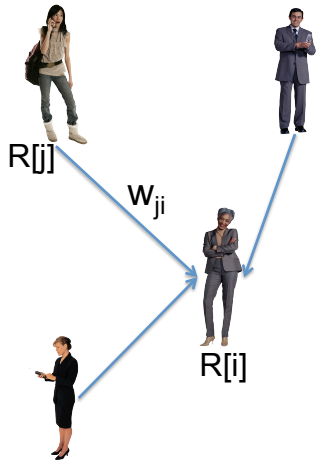
PageRank



©Emily Fox 2014

16

PageRank Iteration



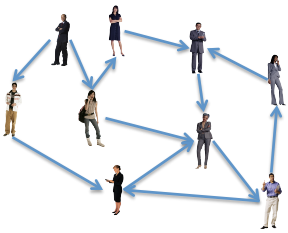
$$R[i] = \alpha + (1-\alpha) \sum_{(j,i) \in E} w_{ji} R[j]$$

- α is the random reset probability
- w_{ji} is the prob. transitioning (similarity) from j to i

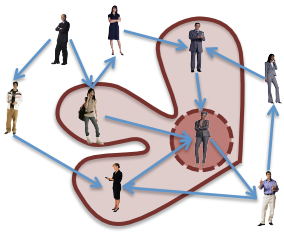
©Emily Fox 2014

Properties of Graph Parallel Algorithms

Dependency Graph



Local Updates



Iterative Computation

©Emily Fox 2014

Addressing Graph-Parallel ML

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graph-Parallel Abstraction

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

Data-Mining
PageRank
Triangle Counting

©Emily Fox 2014

Graph Computation:

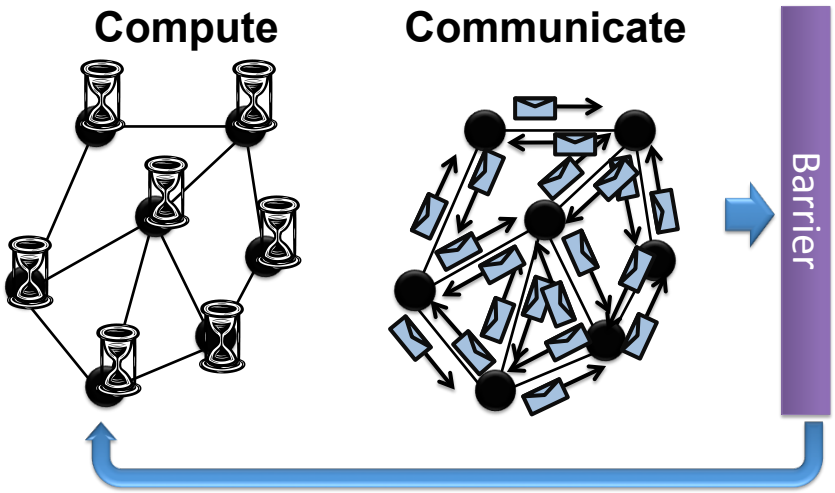
Synchronous

v.

Asynchronous

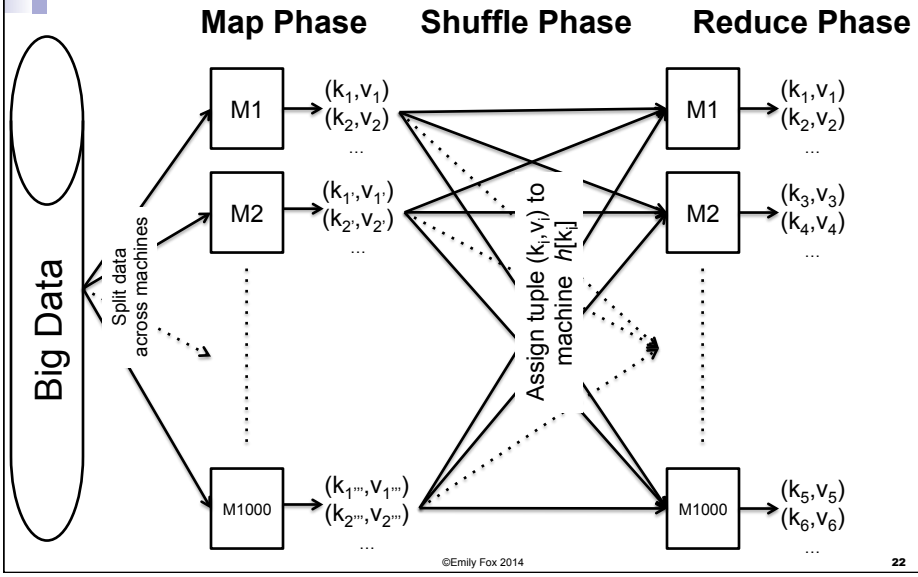
Bulk Synchronous Parallel Model: Pregel (Giraph)

[Valiant '90]



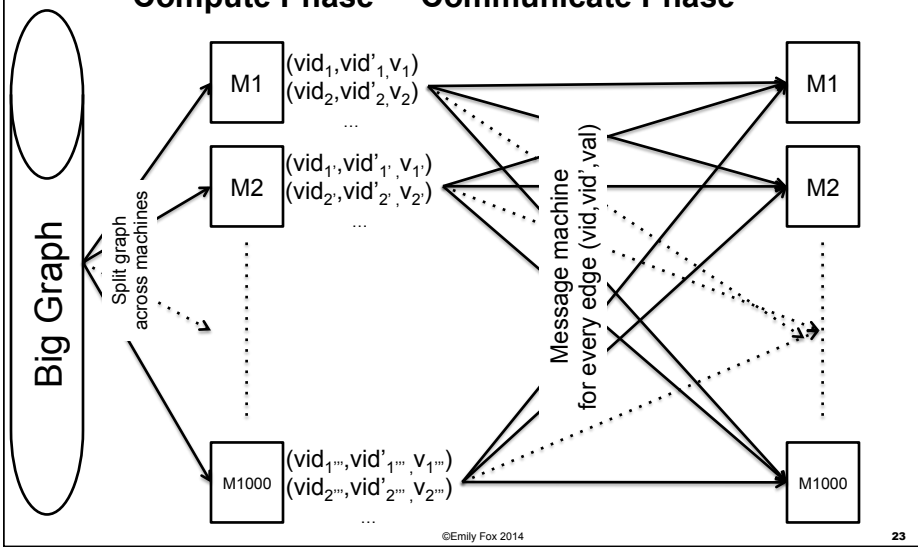
©Emily Fox 2014

Map-Reduce – Execution Overview



©Emily Fox 2014

BSP – Execution Overview

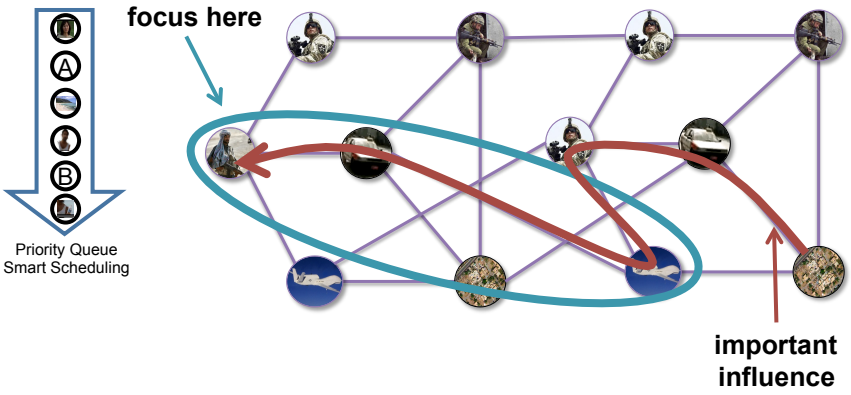


©Emily Fox 2014

Bulk synchronous parallel model
provably inefficient
for some ML tasks

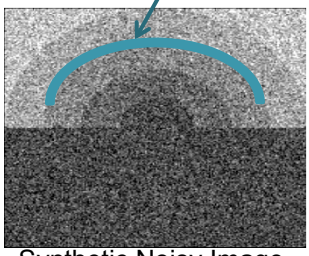
Analyzing Belief Propagation

[Gonzalez, Low, G. '09]

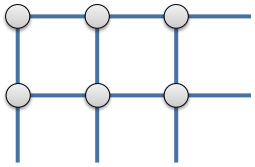


Asynchronous Belief Propagation

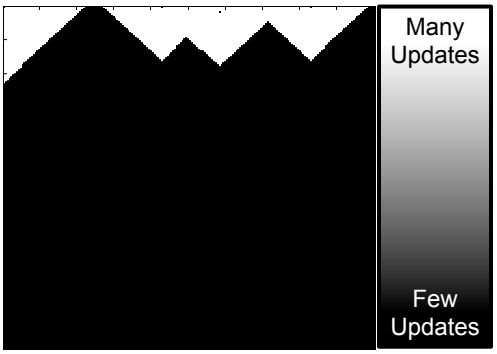
Challenge = Boundaries



Synthetic Noisy Image



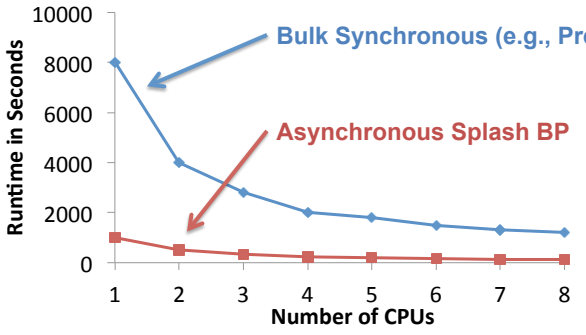
Graphical Model



Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

BSP ML Problem:
Synchronous Algorithms can be **Inefficient**



Theorem:
Bulk Synchronous BP O(#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

Case Study 4: Collaborative Filtering

GraphLab

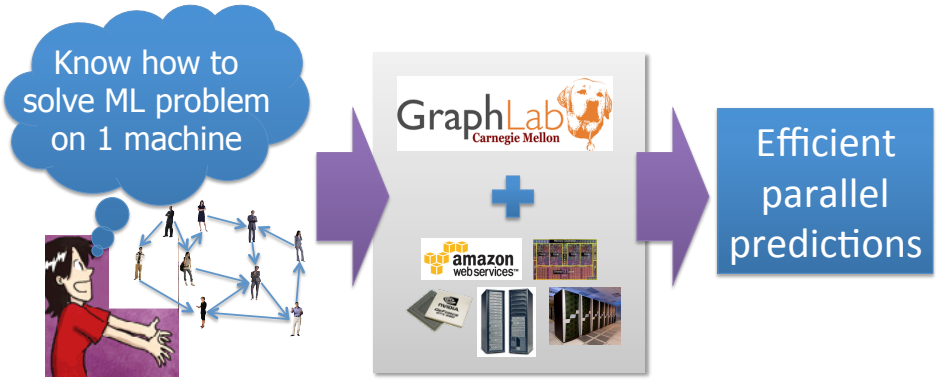
Machine Learning for Big Data
CSE547/STAT548, University of Washington

Emily Fox

February 25th, 2014

©Emily Fox 2014

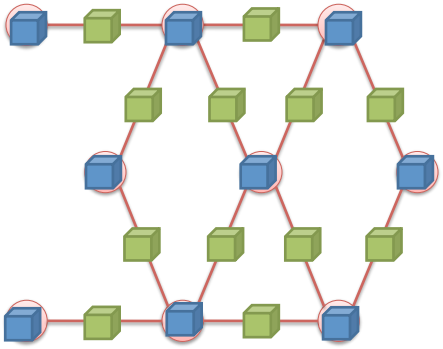
The **GraphLab** Goals



©Emily Fox 2014

Data Graph

Data associated with vertices and edges



Graph:
- Social Network

Vertex Data:
- User profile text
- Current interests estimates

Edge Data:
- Similarity weights

©Emily Fox 2014

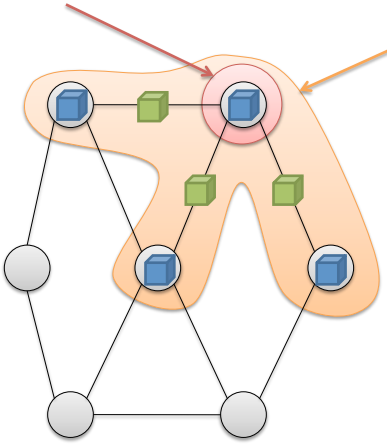
How do we *program* **graph** computation?

"Think like a Vertex."

-Malewicz et al. [SIGMOD'10]

Update Functions

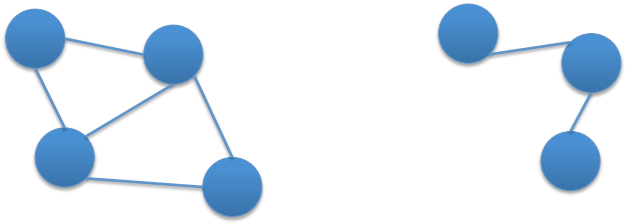
User-defined program: applied to **vertex** transforms data in **scope** of vertex



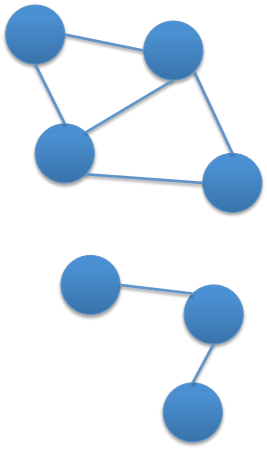
```
pagerank(i, scope){

}
```


Update Function Example: Connected Components

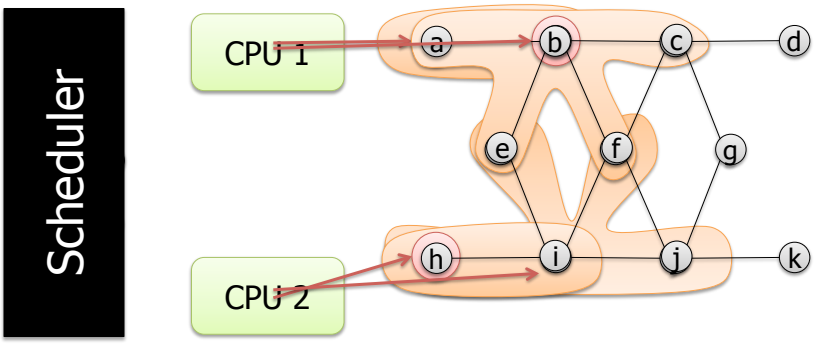


Update Function Example: Connected Components



The Scheduler

The **scheduler** determines order vertices are updated



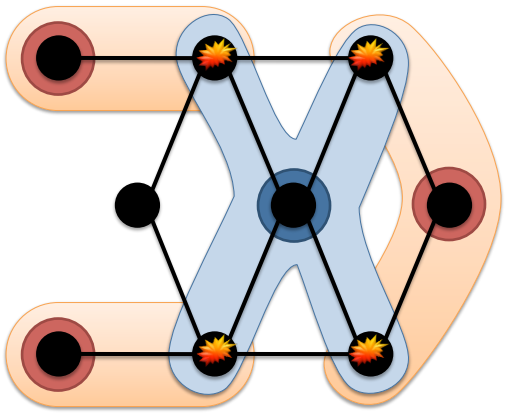
Example Schedulers

- Round-robin
- Selective scheduling (skipping):
  - round robin but jump over un-scheduled vertice
- FIFO
- Prioritize scheduling
  - Hard to implement in a distributed fashion
    - Approximations used (each machine has its own priority queue)

©Emily Fox 2014

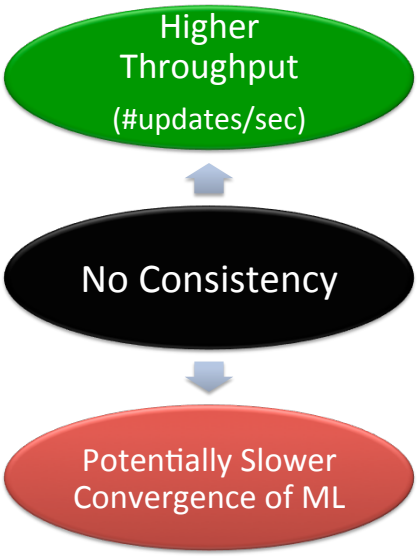
Ensuring Race-Free Code

How much can computation **overlap**?



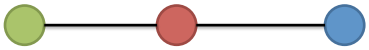
©Emily Fox 2014

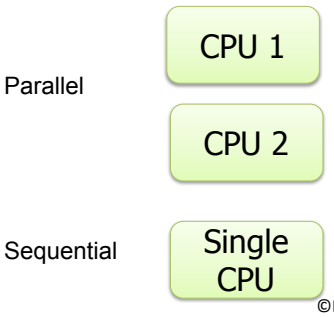
Need for Consistency?



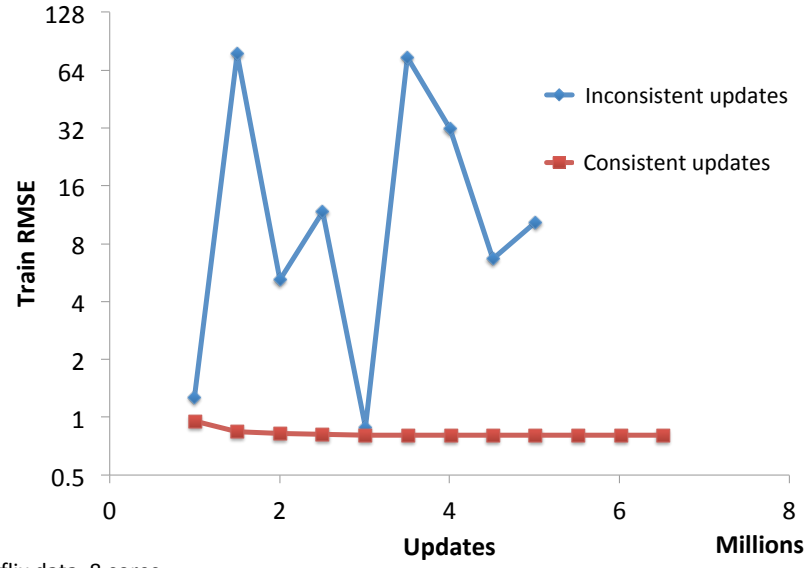
GraphLab Ensures **Sequential Consistency**

For **each parallel execution**, there exists a **sequential execution** of update functions which produces the same result





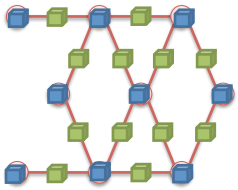
Consistency in Collaborative Filtering



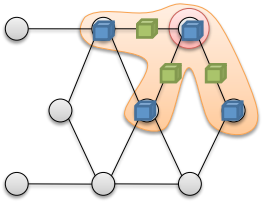
Netflix data, 8 cores

The GraphLab Framework

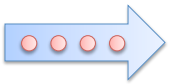
Graph Based
Data Representation



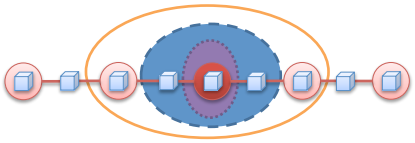
Update Functions
User Computation



Scheduler



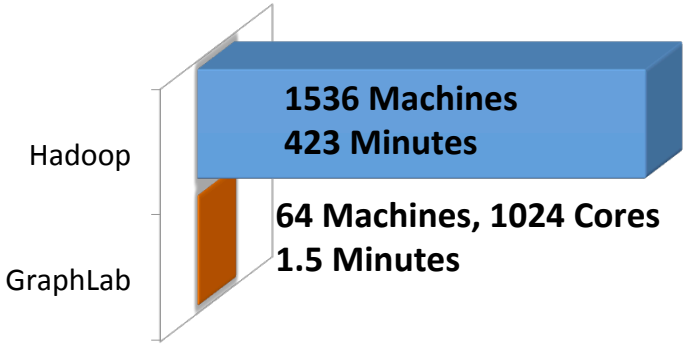
Consistency Model



Triangle Counting in Twitter Graph

40M Users
1.2B Edges

Total:
34.8 Billion Triangles



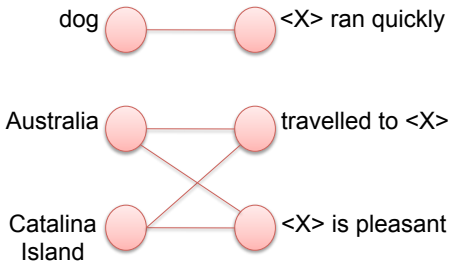
Hadoop results from [Suri & Vassilvitskii '11]

©Emily Fox 2014

CoEM (Jones et al., 2005)

Named Entity Recognition Task

Is "Dog" an animal?
Is "Catalina" a place?

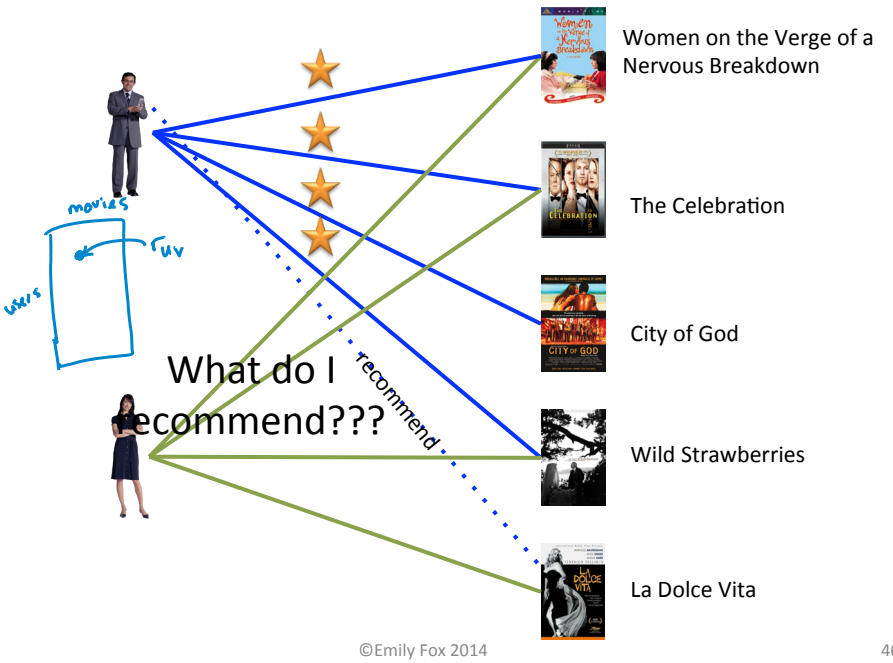


©Emily Fox 2014

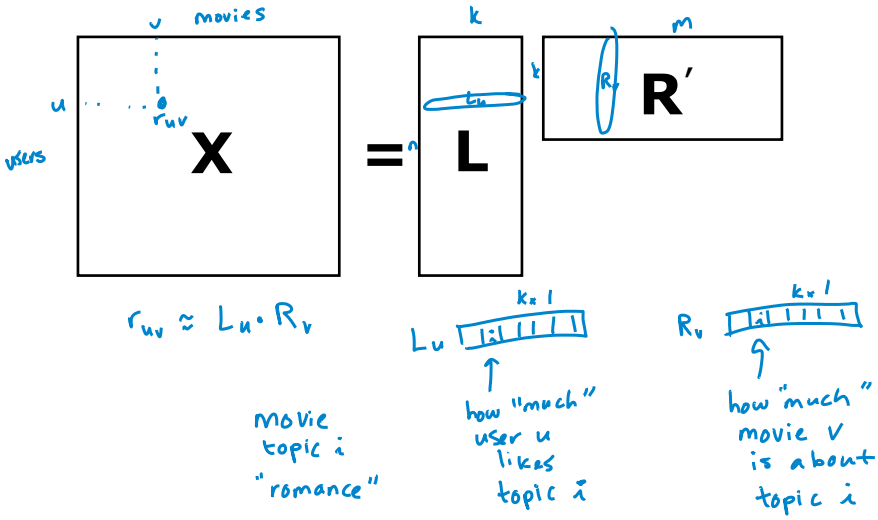
Never Ending Learner Project (CoEM)

Vertices: 2 Million
Edges: 200 Million

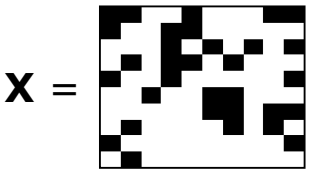
Hadoop	95 Cores	7.5 hrs
Distributed GraphLab	**32 EC2 machines**	**80 secs**



Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)



Matrix Completion as a Graph



X_{ij} known for black cells
X_{ij} unknown for white cells

Rows index users
Columns index movies

Coordinate Descent for Matrix Factorization: Alternating Least-Squares

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

- Fix movie factors, optimize for user factors
  - Independent least-squares over users

$$\min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L\|$$

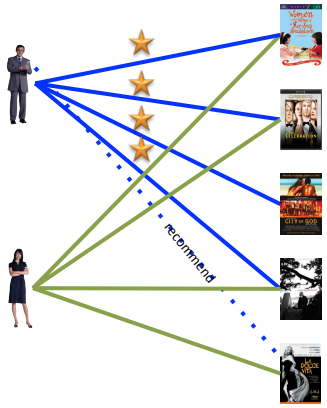
- Fix user factors, optimize for movie factors
  - Independent least-squares over movies

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

- System may be underdetermined: use regularization
- Converges to local optima

Alternating Least Squares Update Function

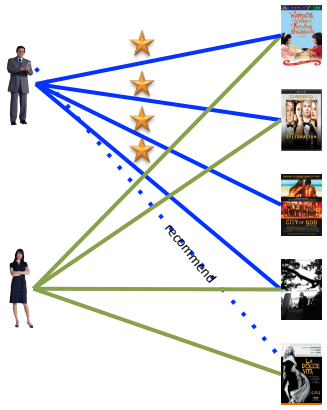
$$\min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 \qquad \min_{R_v} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2$$



SGD for Matrix Factorization in GraphLab

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

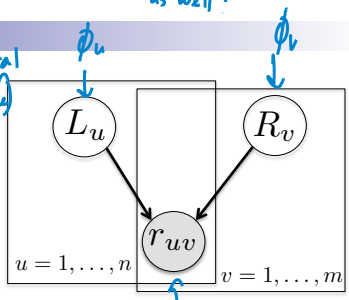
$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1-\eta_t\lambda_u)L_u^{(t)} - \eta_t\epsilon_t R_v^{(t)} \\ (1-\eta_t\lambda_v)R_v^{(t)} - \eta_t\epsilon_t L_u^{(t)} \end{bmatrix}$$



©Emily Fox 2014

Bayesian PMF Example

- Latent user and movie factors:
  $L_u \sim N(\mu_u, \Sigma_u)$ $u = 1, \dots, n$
  $R_v \sim N(\mu_v, \Sigma_v)$ $v = 1, \dots, m$
- Observations $r_{uv} \sim N(L_u' R_v, \sigma_r^2)$
- Hyperparameters:
  $\phi = \{\mu_u, \Sigma_u, \mu_v, \Sigma_v, \sigma_r^2\}$, with ϕ_u, ϕ_v, ϕ_r
- Want to predict new movie rating:
  $p(r_{uv}^* \mid X, \phi) = \int p(r_{uv}^* \mid L_u, R_v)\, p(L, R \mid X, \phi)\, dL\,dR$



©Emily Fox 2014

Bayesian PMF Gibbs Sampler

- Outline of Bayesian PMF sampler

1. Init $L^{(1)}, R^{(1)}$
2. For $k = 1, \ldots, N_{iter}$
   (i) Sample hyperparams $\phi^{(k)} = \{\phi_u^{(k)}, \phi_v^{(k)}, \phi_r^{(k)}\}$
   (ii) For each user $u = 1, \ldots, n$ sample in parallel
   $$L_u^{(k+1)} \sim p(L_u \mid X, R^{(k)}, \phi^{(k)})$$
   (iii) For each movie $v = 1, \ldots, m$ sample in parallel
   $$R_v^{(k+1)} \sim p(R_v \mid X, L^{(k+1)}, \phi^{(k)})$$

Very similar to ideas of ALS (systematically)

Bayesian PMF Example

- For user u:

prior ↙ ↙ likelihood for user u

$$p(L_u \mid X, R, \phi_u) \propto p(L_u \mid \phi_u) \prod_{v \in V_u} p(r_{uv} \mid L_u, R_v, \phi_r)$$

$$\propto \mathcal{N}(L_u \mid \mu_u, \Sigma_u) \prod_{v \in V_u} \mathcal{N}(r_{uv} \mid L_u \cdot R_v, \sigma_r^2)$$

$$= \mathcal{N}(L_u \mid \tilde{\mu}_u, \tilde{\Sigma}_u) \quad \leftarrow \text{via conjugacy}$$

↖ posterior is in the same family as prior

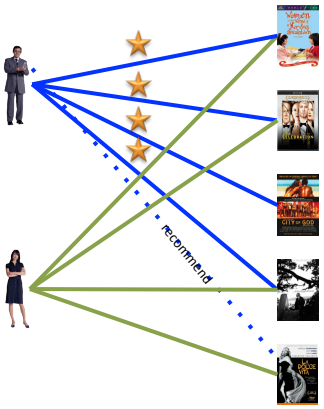
where $\tilde{\Sigma}_u^{-1} = \Sigma_u^{-1} + \sigma_r^{-2} \sum_{v \in V_u} R_v R_v^T$

$$\tilde{\mu}_u = \tilde{\Sigma}_u \left(\sigma_r^{-2} \sum_{v \in V_u} r_{uv} R_v + \Sigma_u \mu_u \right)$$

- Symmetrically for R_v conditioned on L (breaks down over movies)
- Luckily, we can use this to get our desired posterior samples

PMF Gibbs Sampling in GraphLab

$$p(L_u \mid X, R, \phi_u) = N(\tilde{\mu}_u, \tilde{\Sigma}_u) \qquad \tilde{\Sigma}_u = \Sigma_u^{-1} + \sigma_r^{-2} \sum_{v \in V_u} R_v R_v^T \qquad \tilde{\mu}_u = \tilde{\Sigma}_u \left(\sigma_r^{-2} \sum_{v \in V_u} r_{uv} R_v + \Sigma_u \mu_u \right)$$



©Emily Fox 2014

GraphLab

Release 2.2 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
- ALS, SGD and Gibbs for matrix factorization/PMF in GraphLab

©Emily Fox 2014

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

©Emily Fox 2014

Acknowledgements

- Slides based on Carlos Guestrin's GraphLab talk

©Emily Fox 2014