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

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)

- 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)

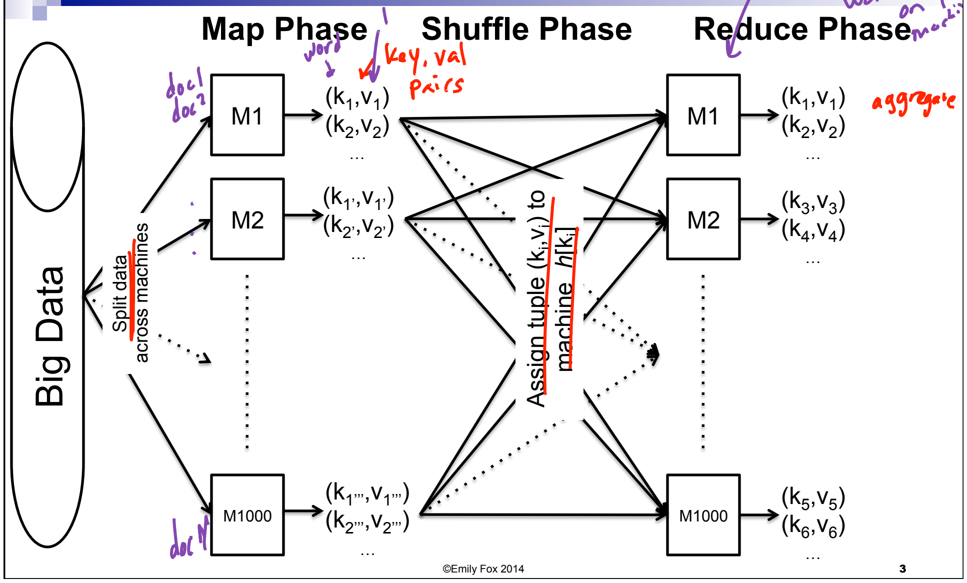
Example: word count
map(document)
for word in doc
emit(word,1)

Reduce(word, count: list(int))
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

all (k,v) pairs for a given word appear on 1 machine



Issues with Map-Reduce Abstraction

- Often all data gets moved around cluster ← in shuffle phase
  - Very bad for iterative settings
- Definition of Map & Reduce functions can be unintuitive in many apps
  - Graphs are challenging
- Computation is synchronous ← MapReduce waits for all mappers to finish

SGD for Matrix Factorization in Map-Reduce?

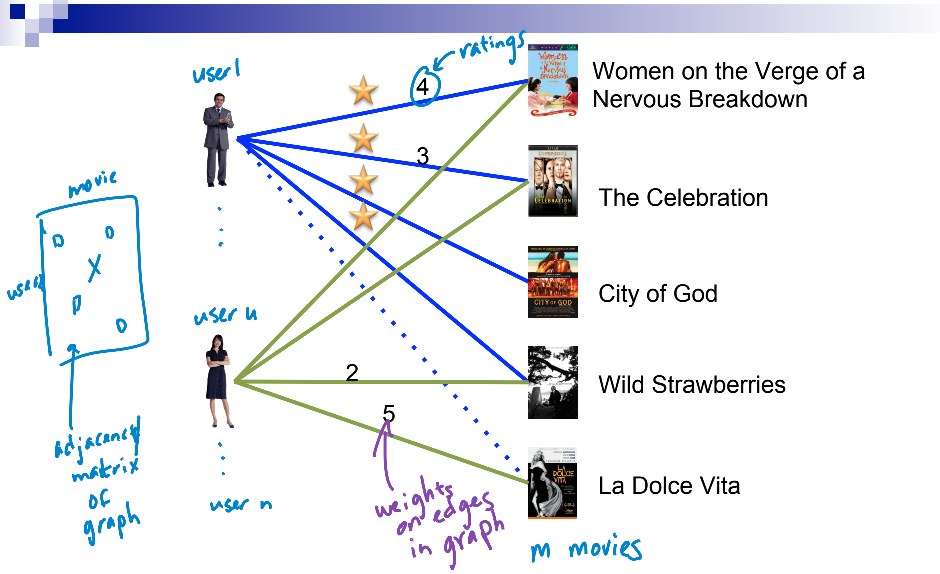
$$\begin{bmatrix} L_u^{(t+1)} \\ R_v^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} (1-\eta_t\lambda_u)L_u^{(t)} - \eta_t\epsilon_t R_v^{(t)} \\ (1-\eta_t\lambda_v)R_v^{(t)} - \eta_t\epsilon_t L_u^{(t)} \end{bmatrix}$$

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

- 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

transitioned to

altavista SEARCH SOFTWARE

Why?

Google! BETA

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

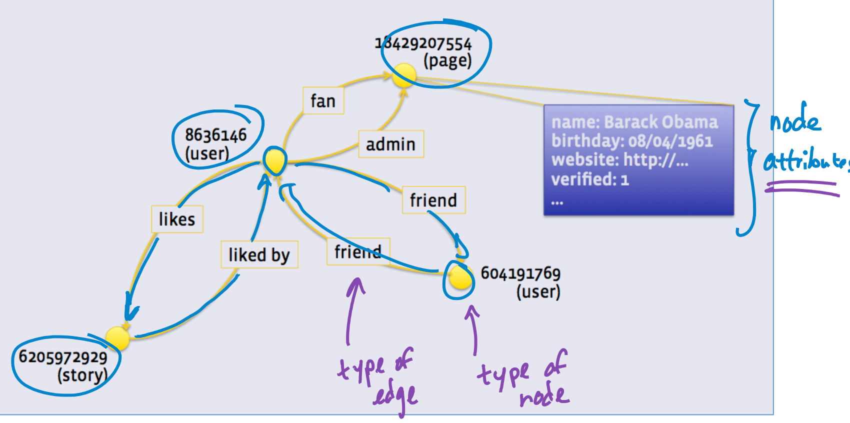
"page rank" ← see this later

Facebook Graph

Data model

Objects & Associations

more than just friend/friend interactions



Slide from Facebook Engineering presentation 8

Label a Face and Propagate

©Emily Fox 2014

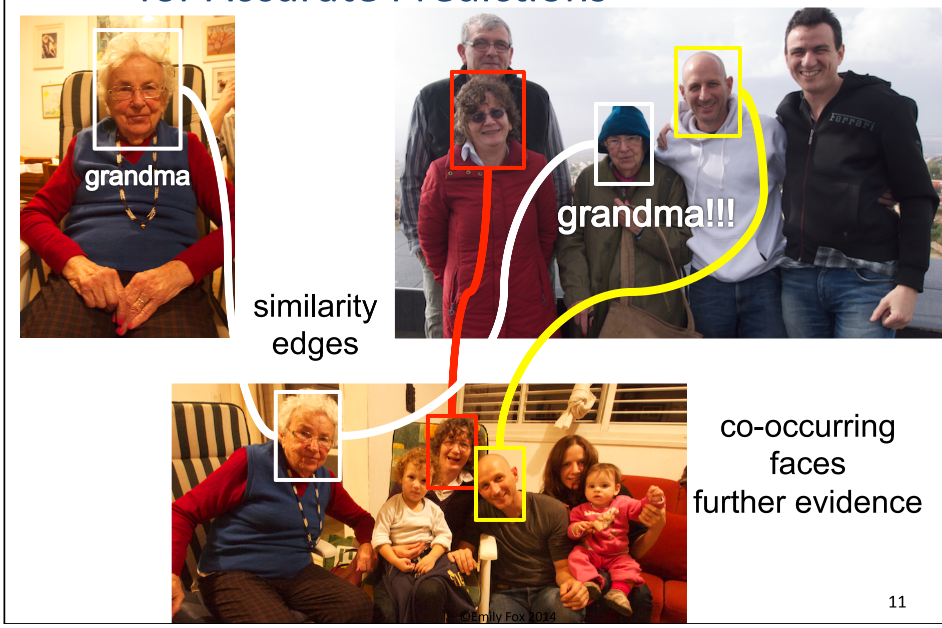
9

Pairwise similarity not enough…

©Emily Fox 2014

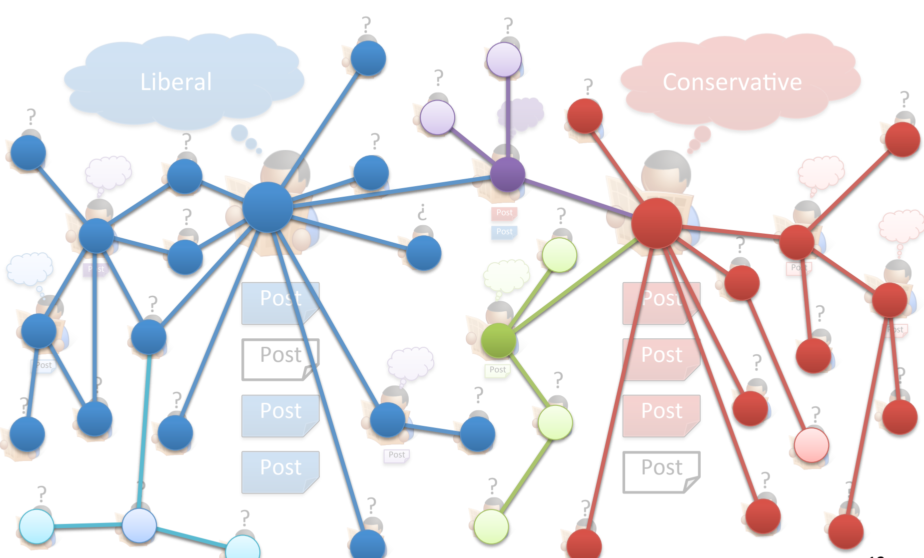
10

Propagate Similarities & Co-occurrences for Accurate Predictions



11

Example: *Estimate Political Bias*

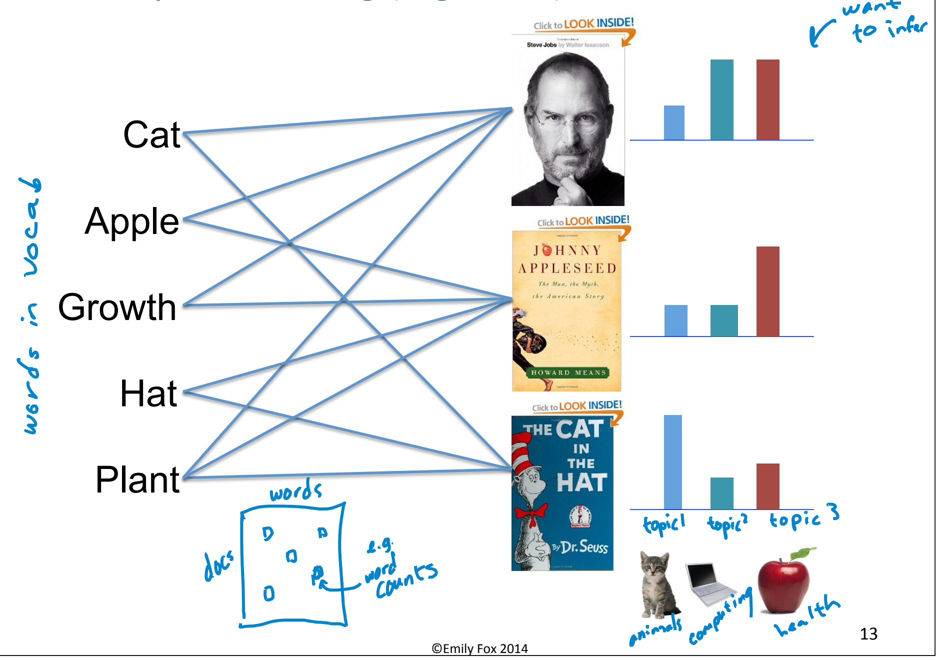


12

Topic Modeling (e.g., LDA)

docs in corpus

want to infer



ML Tasks Beyond Data-Parallelism

Data-Parallel ⟷ Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

focus before

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

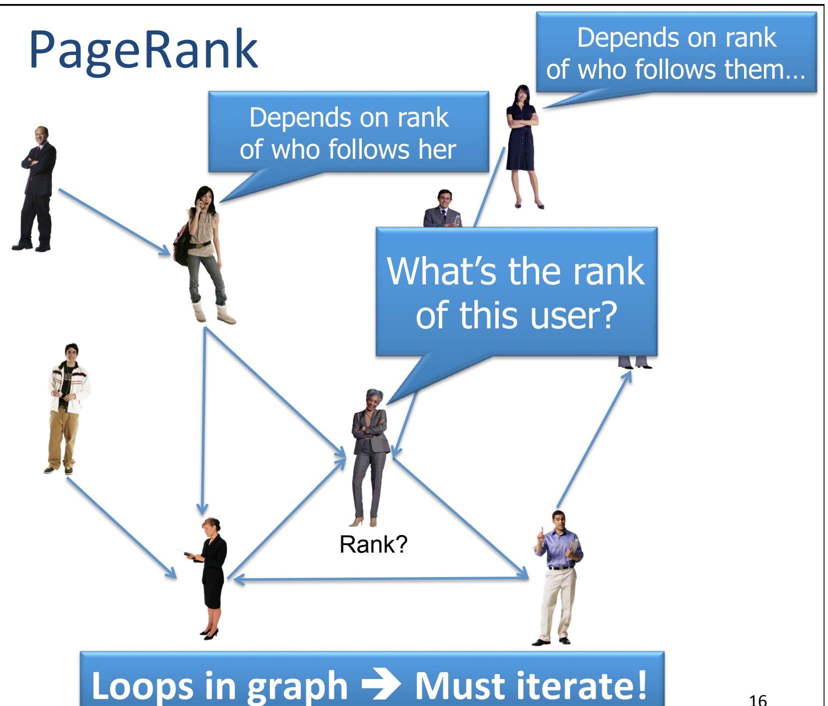
Collaborative Filtering
Tensor Factorization

Graph Analysis
PageRank
Triangle Counting

but many graph-structured problems

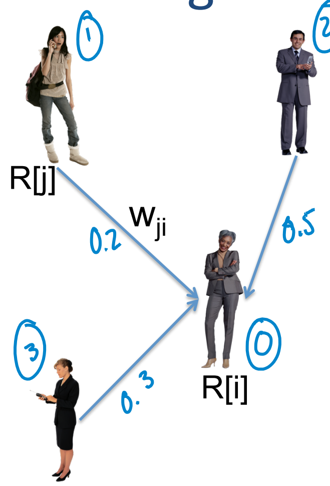
Example of a Graph-Parallel Algorithm

PageRank



16

PageRank Iteration



rank of user i

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

edges in graph

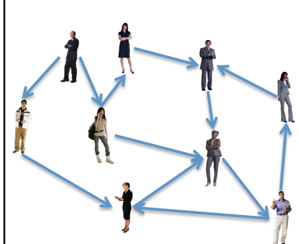
$R[0] = 0.15 + 0.85\,(0.2\,R[1] + 0.5\,R[2] + 0.3\,R[3])$

0.15

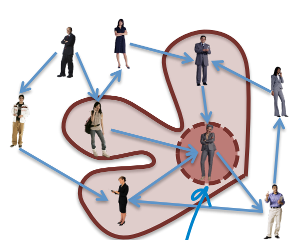
- α is the random reset probability ← prob. of landing on i directly
- w_{ji} is the prob. transitioning (similarity) from j to i

Properties of Graph Parallel Algorithms

Dependency Graph

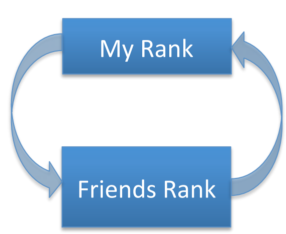


Local Updates



algorithm shown on prev. slide

Iterative Computation



guaranteed to conv. for page rank

Addressing Graph-Parallel ML

Data-Parallel ⟷ Graph-Parallel

Map Reduce	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

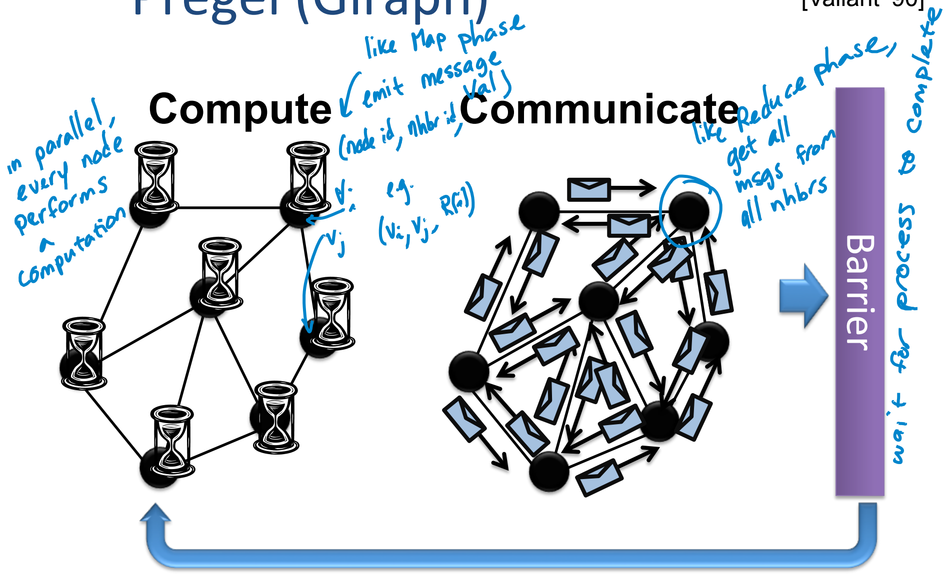
v.

Asynchronous

key questions

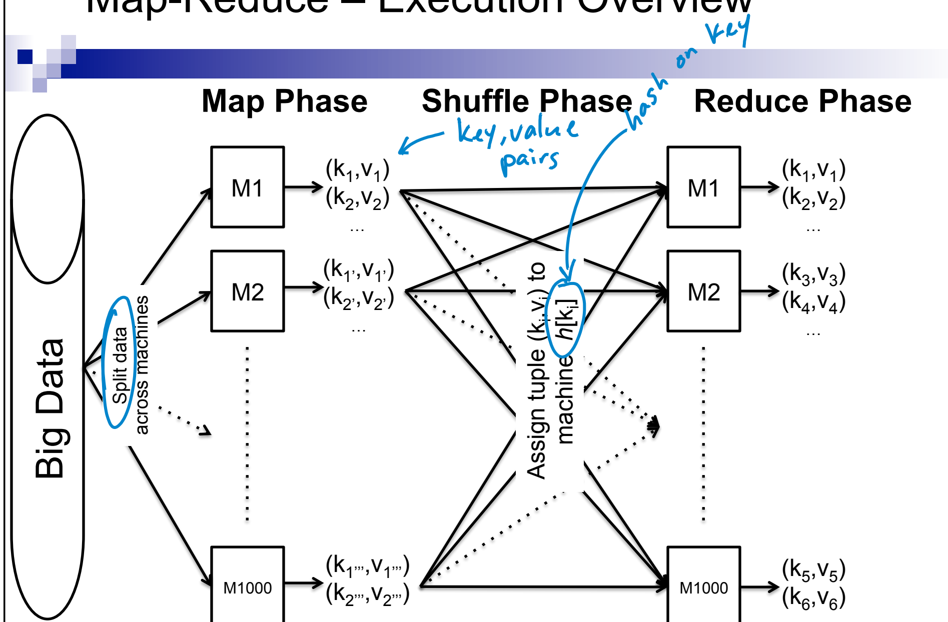
Bulk Synchronous Parallel Model: Pregel (Giraph)

[Valiant '90]

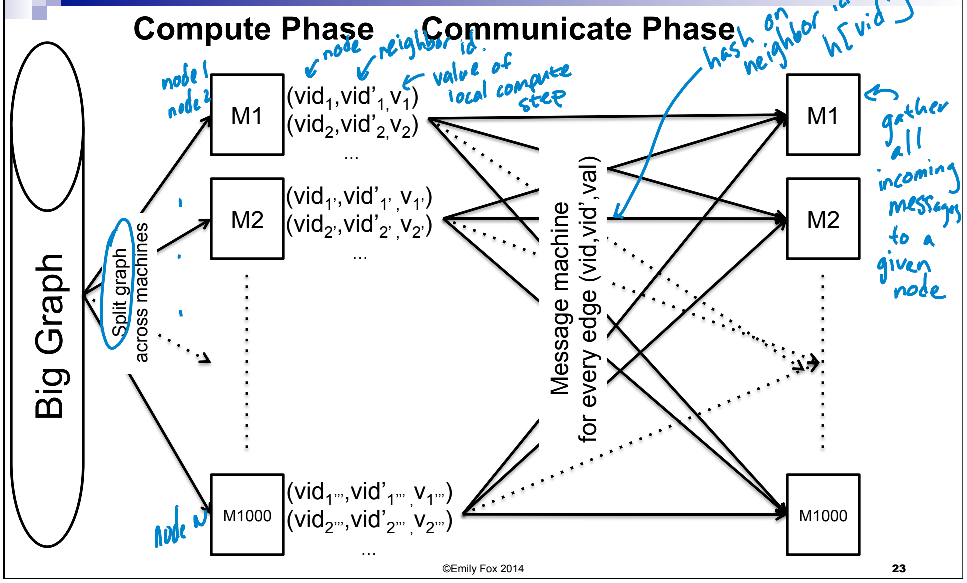


Recall:

Map-Reduce – Execution Overview



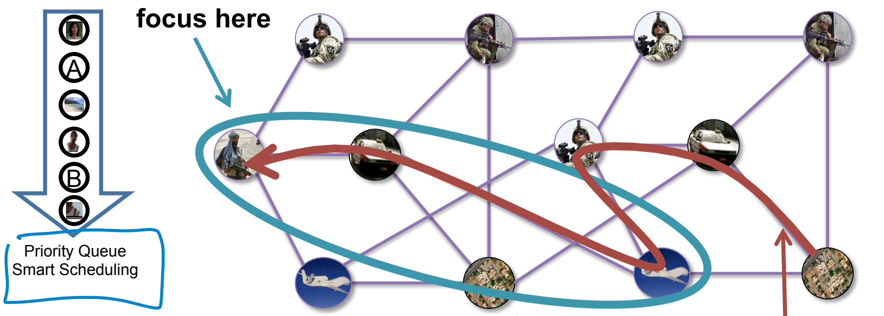
BSP – Execution Overview



Bulk synchronous parallel model
provably inefficient
for some ML tasks

Analyzing Belief Propagation

[Gonzalez, Low, G. '09]



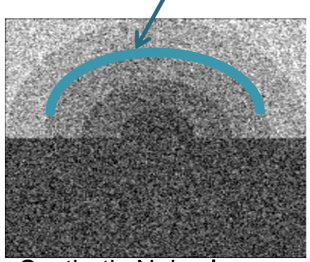
Specific case of importance of asynch. comp.

Focus our updating computations here

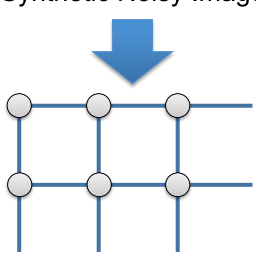
want to discover this path

Asynchronous Belief Propagation

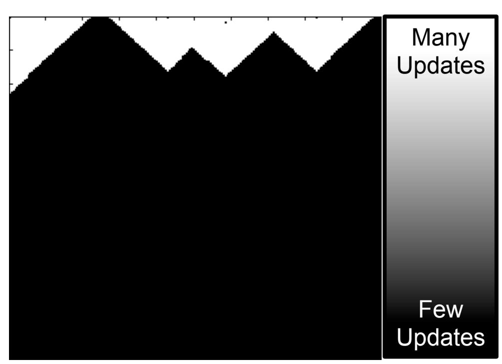
Challenge = Boundaries



Synthetic Noisy Image



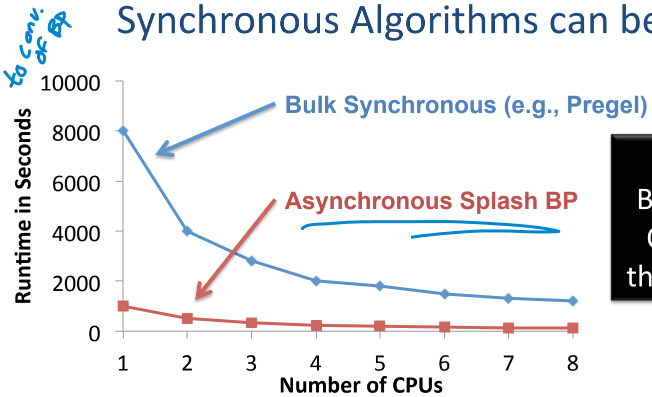
Graphical Model



Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

BSP ML Problem: BP on real data
Synchronous Algorithms can be **Inefficient**



Theorem:
Bulk Synchronous BP
O(#vertices) slower
than Asynchronous BP

1 processor asynch. faster than 8 proc. of synch!

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

using current state +

- 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

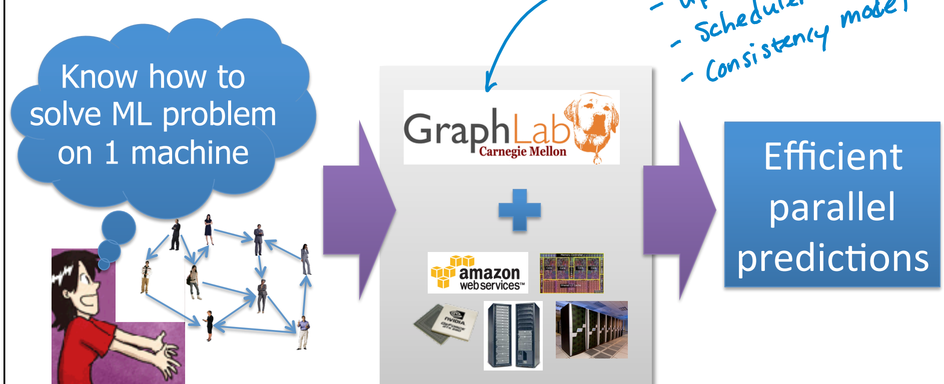
Case Study 4: Collaborative Filtering

GraphLab

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
February 25th, 2014

The **GraphLab** Goals

4 components:
- Data graph
- Update fcn
- Scheduler
- Consistency model

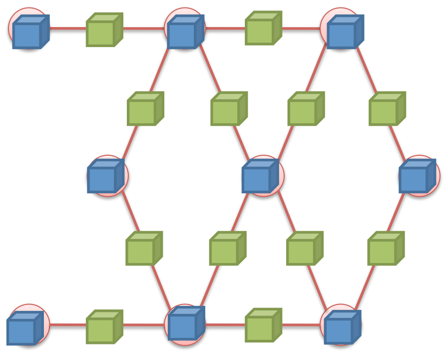


1 Data Graph

Data associated with vertices and edges

nodes/vertices

edges



Graph:
- Social Network

vertex = user
edge = relationship

Vertex Data:
- User profile text
- Current interests estimates

"data" can live at vertices

Edge Data:
- Similarity weights

"data" can live on edges, too

©Emily Fox 2014 31

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

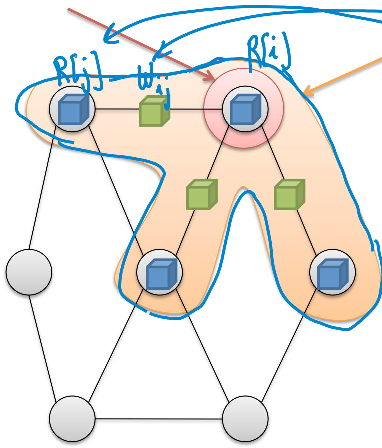
"Think like a Vertex."

-Malewicz et al. [SIGMOD'10]

2 Update Functions

scope = neighboring vertices + edges

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



read values from "scope" variable

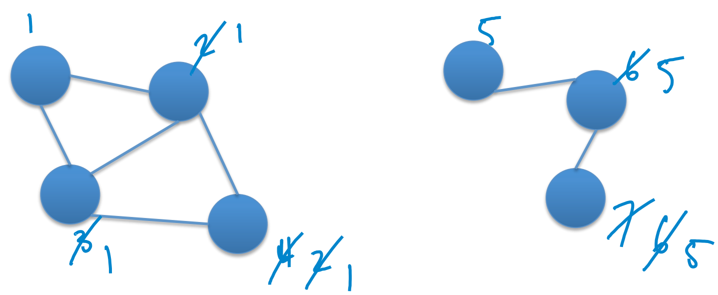
pagerank(i, scope){

$$R[i] = 0.15 + 0.85 \sum_{j \in nbhrs(i)} w_{ij} R[j]$$

contained in "scope"

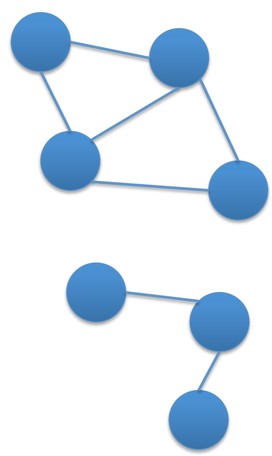
}

Update Function Example: Connected Components



1. Initialize all nodes w/ unique labels
2. Pick a node:
   component = min (self, neighbors)
3. Return to step 2

Update Function Example: Connected Components



label of node i

init component[i] = i

update(i, scope) {

component[i] =

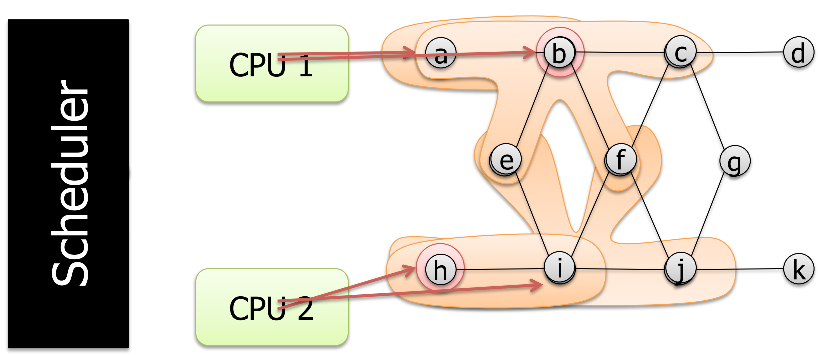
min[component[i], $\min_{j \in nbr(i)}$ component[j]]

all in "scope"

}

③ The Scheduler

The **scheduler** determines order vertices are updated



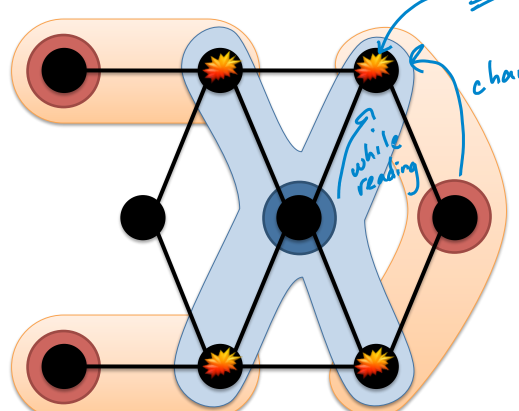
Example Schedulers

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

4 Ensuring Race-Free Code

How much can computation **overlap**?

race cond.

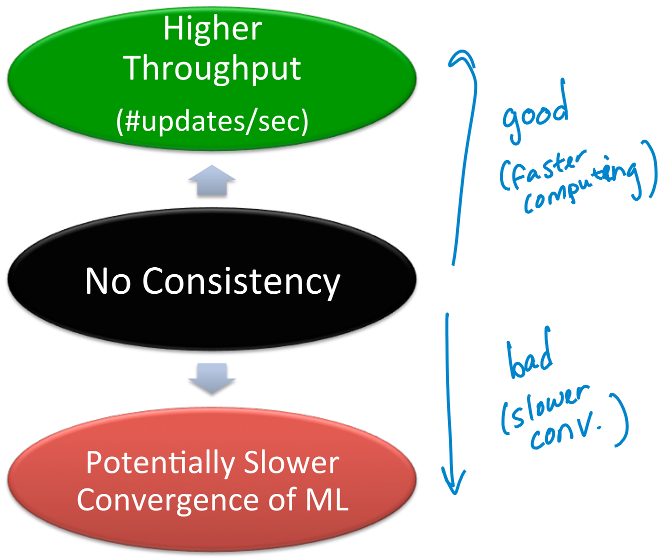


change value

in some cases, you don't change your nbrs (page rank) but in other cases like BP you do

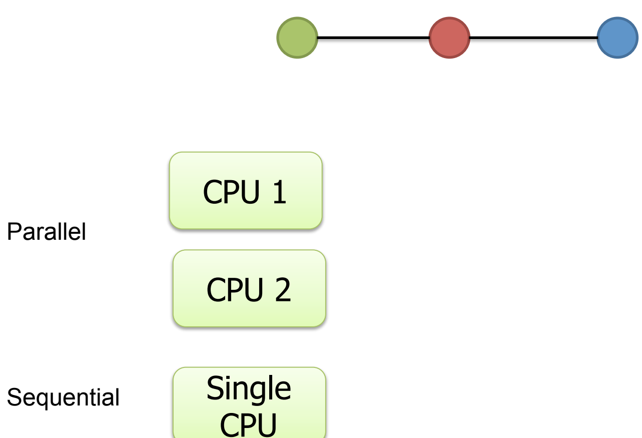
can do updates together if scopes don't overlap

Need for Consistency?

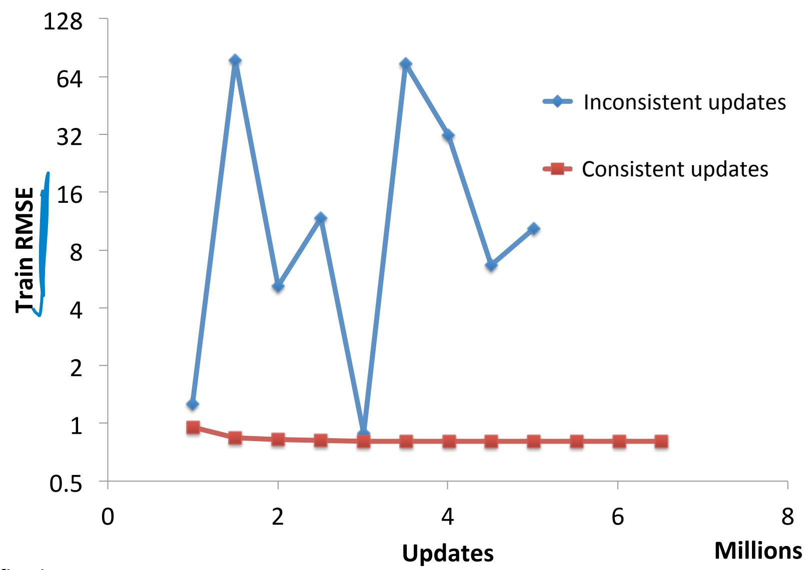


GraphLab Ensures **Sequential Consistency**

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



Consistency in Collaborative Filtering

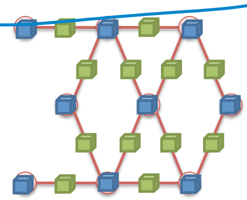


Netflix data, 8 cores

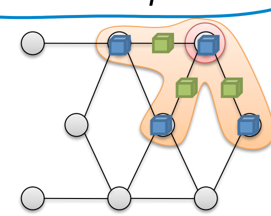
The GraphLab Framework

provided by user

Graph Based
Data Representation

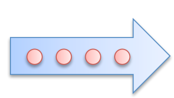


Update Functions
User Computation



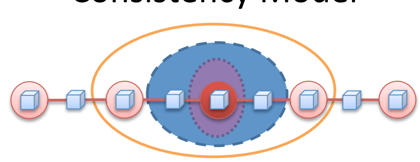
chosen from a list

Scheduler

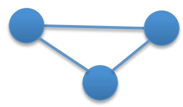


Graphlab handles this

Consistency Model

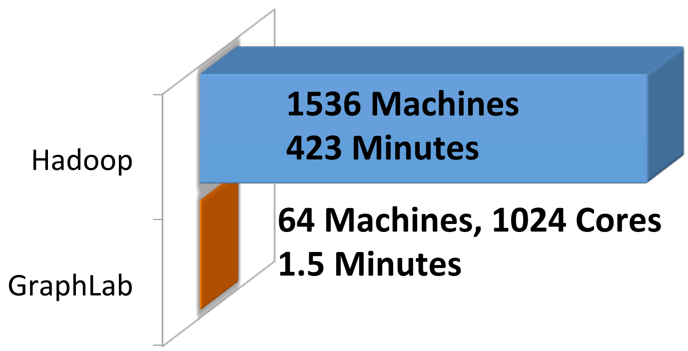


Triangle Counting in Twitter Graph



40M Users
1.2B Edges

Total:
34.8 Billion Triangles



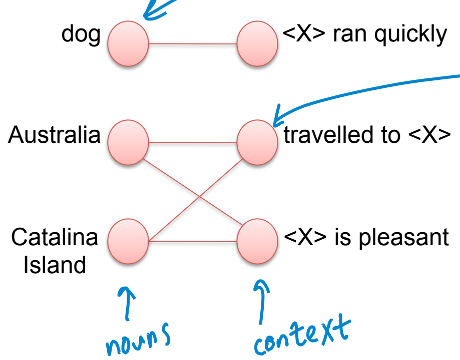
Hadoop results from [Suri & Vassilvitskii '11]

CoEM (Jones et al., 2005)

Named Entity Recognition Task

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

want to learn distribution over {animal, place, ..} for this noun



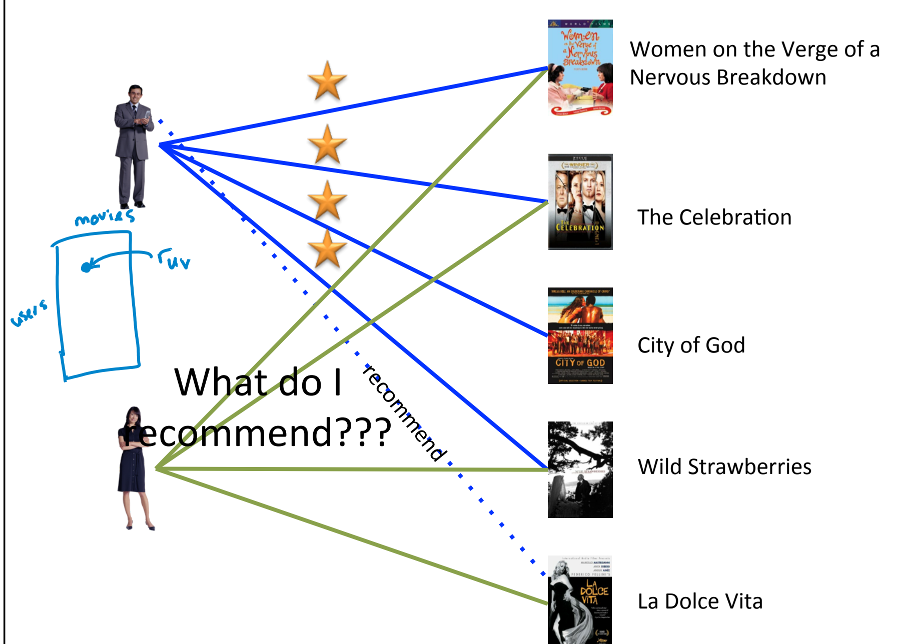
dist. over {animal, place, ..} for this context

→ label nodes
→ Initialize (randomly, smartly)
→ Iterative, EM-like alg.

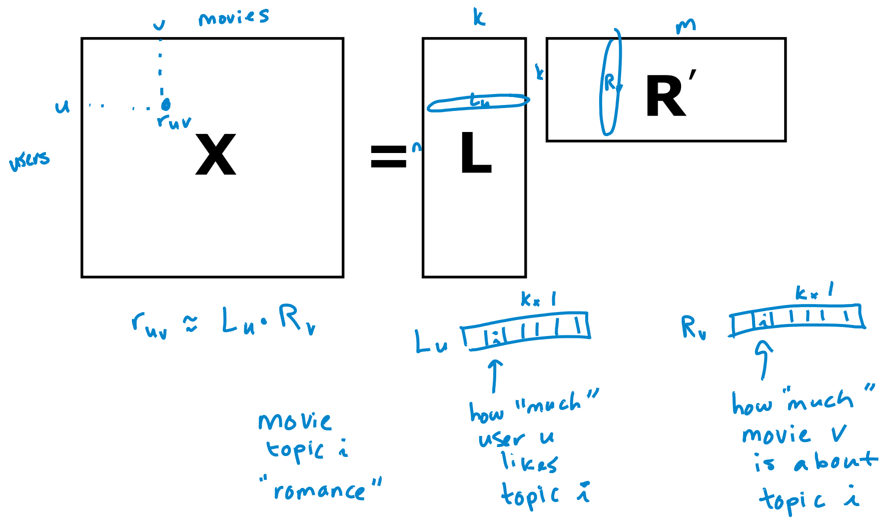
Never Ending Learner Project (CoEM)

Vertices: 2 Million
Edges: 200 Million

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

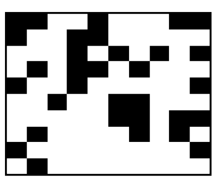


Interpreting Low-Rank Matrix Completion (aka Matrix Factorization)



Matrix Completion as a Graph

X =



X_{ij} known for black cells
X_{ij} unknown for white cells
Rows index users
Columns index movies

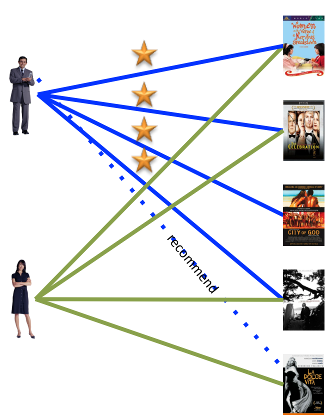
Coordinate Descent for Matrix Factorization: Alternating Least-Squares

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

- 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

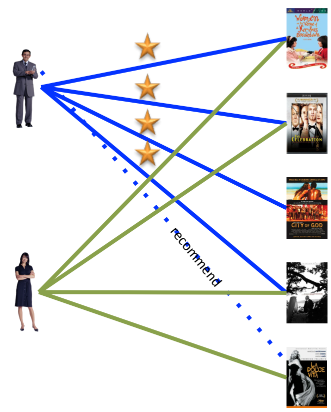
$$\min_{L_u} \sum_{v\in V_u} (L_u \cdot R_v - r_{uv})^2 \qquad \min_{R_v} \sum_{u\in U_v} (L_u \cdot R_v - r_{uv})^2$$



SGD for Matrix Factorization in GraphLab

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

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



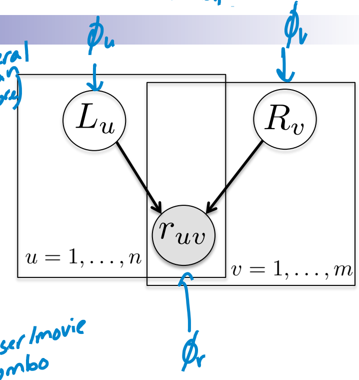
Bayesian PMF Example

∅ full Bayesian approach
place priors on ϕ
as well!

- Latent user and movie factors: (more general than before)

  $L_u \sim N(\mu_u, \Sigma_u) \quad u=1,\dots,n$

  $R_v \sim N(\mu_v, \Sigma_v) \quad v=1,\dots,m$



- Observations $r_{uv} \sim N(L_u' R_v, \sigma_r^2)$
- Hyperparameters:

  $\phi = \{\underbrace{\mu_u, \Sigma_u}_{\phi_u}, \underbrace{\mu_v, \Sigma_v}_{\phi_v}, \underbrace{\sigma_r^2}_{\phi_r}\}$

  new user/movie combo

- Want to predict new movie rating:

  posterior given obs. so far

  $p(r_{uv}^* \mid X, \phi) = \int p(r_{uv}^* \mid L_u, R_v)\, p(L, R \mid X, \phi)\, dL\, dR$

  new rating   obs. ratings

Bayesian PMF Gibbs Sampler

- Outline of Bayesian PMF sampler

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

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

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

$$= N(L_u \mid \tilde{\mu}_u, \tilde{\Sigma}_u)$$ ← via conjugacy

↖ posterior is in the same family as prior

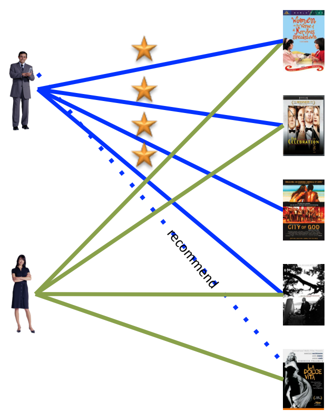
where $\tilde{\Sigma}_u^{-1} = \Sigma_u^{-1} + \sigma_r^{-2} \sum_{v \in V_u} R_v R_v^T$

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

- 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

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



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

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

Acknowledgements

- Slides based on Carlos Guestrin's GraphLab talk