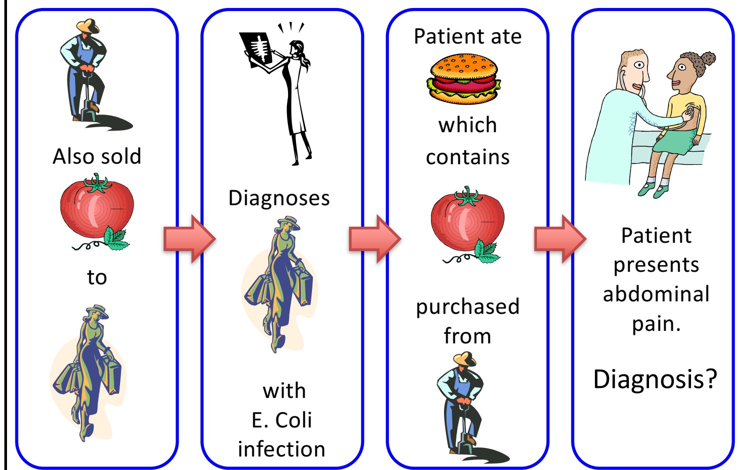
Big Data: Graph Processing

COS 418: *Distributed Systems*
Lecture 21

Kyle Jamieson

[Content adapted from J. Gonzalez]



Big Data is Everywhere

6 Billion Flickr Photos

28 Million Wikipedia Pages

900 Million Facebook Users

72 Hours a Minute YouTube

- **Machine learning** is a reality
- How will we design and implement **"Big Learning"** systems?

3

We could use ....

Threads, Locks, & Messages

"Low-level parallel primitives"

Shift Towards Use Of Parallelism in ML

GPUs Multicore Clusters Clouds Supercomputers

- Programmers **repeatedly** solve the same **parallel design challenges:**
  - Race conditions, distributed state, communication…
- Resulting code is very **specialized**:
  - **Difficult** to maintain, extend, debug…

Idea: Avoid these problems by using **high-level abstractions**

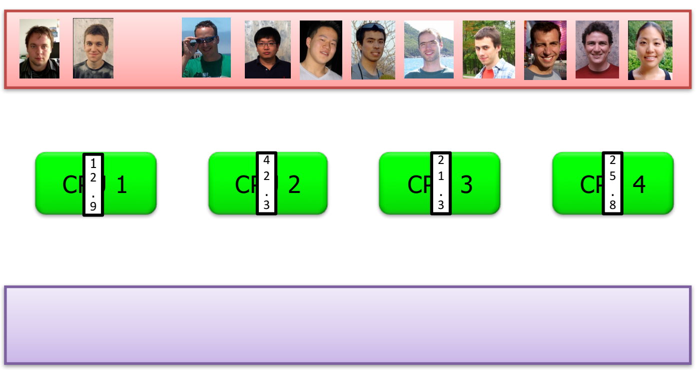
5

… a ***better*** answer:

MapReduce / Hadoop

Build learning algorithms on top of high-level parallel abstractions

MapReduce – Map Phase



Embarrassingly Parallel independent computation
No Communication needed

7

MapReduce – Map Phase

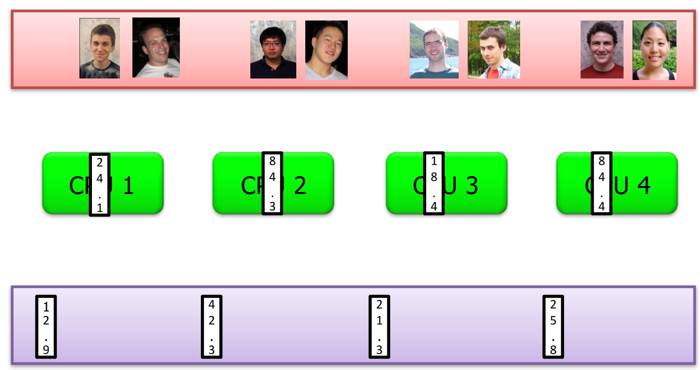
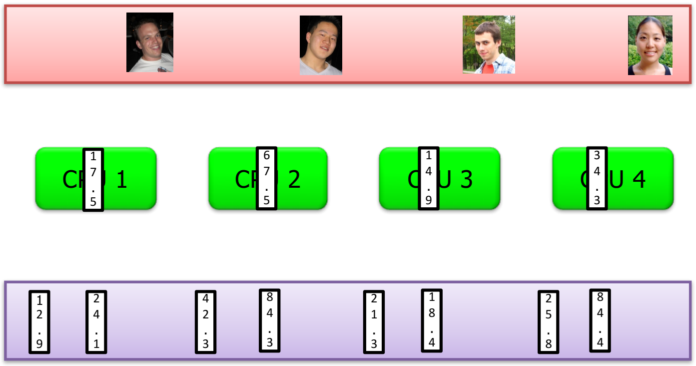


Image Features

8

MapReduce – Map Phase



Embarrassingly Parallel independent computation

MapReduce – Reduce Phase

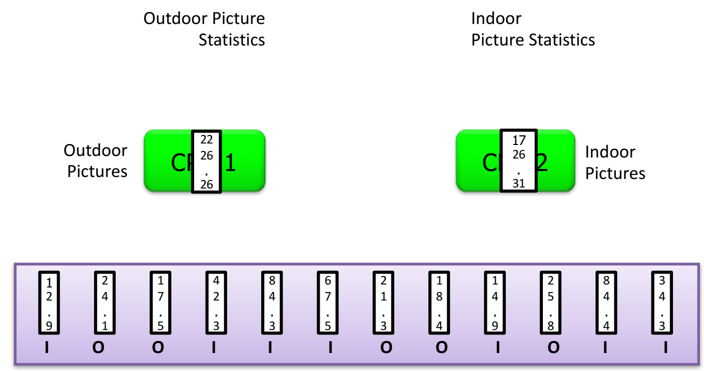


Image Features

Map-Reduce for Data-Parallel ML

- Excellent for **large data-parallel** tasks!

Data-Parallel

Map Reduce

Feature Extraction

Algorithm Tuning

Basic Data Processing

Is there more to Machine Learning

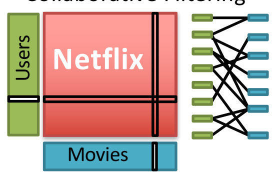
?

Exploiting Dependencies

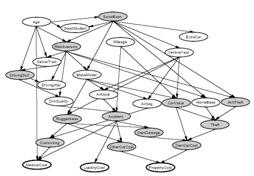
Graphs are Everywhere

Social Network

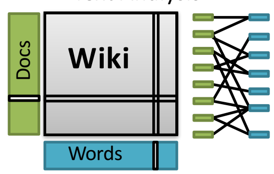
Collaborative Filtering



Probabilistic Analysis



Text Analysis



Concrete Example

Label Propagation

Label Propagation Algorithm

- Social Arithmetic:

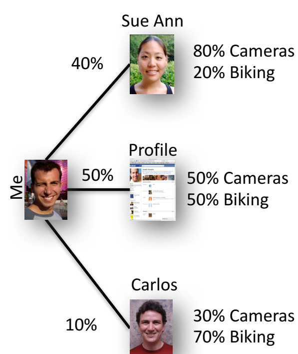
  50% What I list on my profile
  40% Sue Ann Likes
  \+ 10% Carlos Like

  I Like: 60% Cameras, 40% Biking

- Recurrence Algorithm:

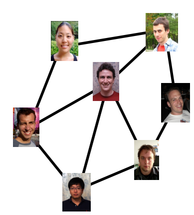
$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

  – iterate until convergence
- Parallelism:
  – Compute all *Likes[i]* in parallel

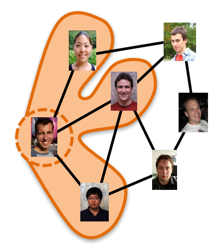


Properties of Graph Parallel Algorithms

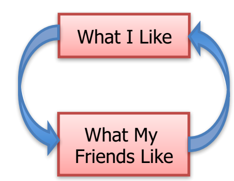
Dependency Graph



Factored Computation



Iterative Computation



Map-Reduce for Data-Parallel ML

- Excellent for **large data-parallel** tasks!

Data-Parallel — Graph-Parallel

MapReduce

Feature Extraction

Algorithm Tuning

Basic Data Processing

MapReduce?

Lasso

Label Propagation

Kernel Methods

Belief Propagation

Tensor Factorization

PageRank

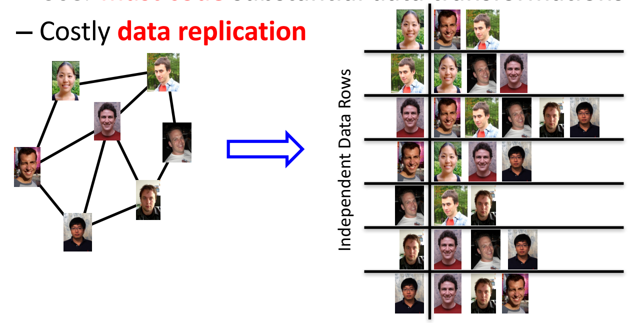
Deep Belief Networks

Neural Networks

17

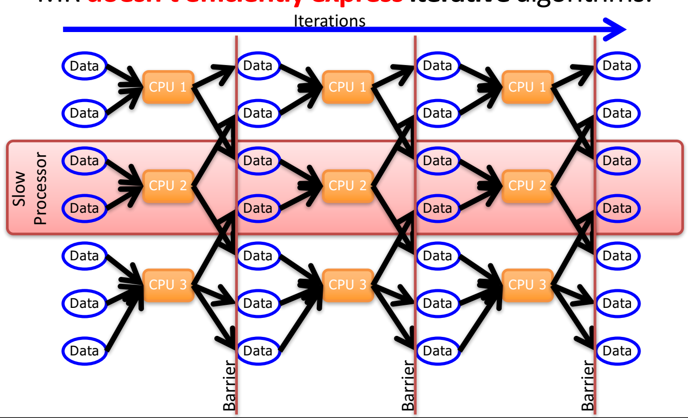
Problem: Data Dependencies

- MapReduce **doesn't efficiently express** data dependencies
  - User **must code substantial data transformations**
  - Costly **data replication**



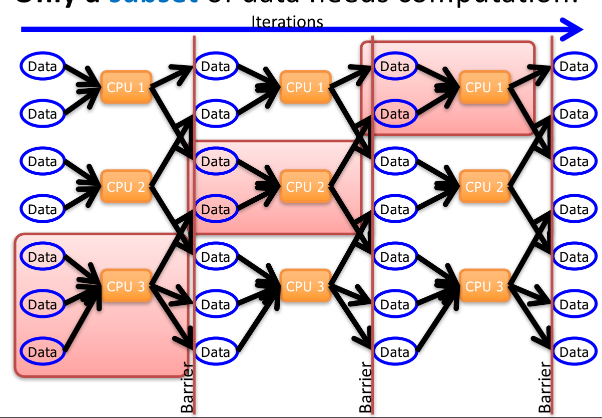
Iterative Algorithms

- MR **doesn't efficiently express iterative** algorithms:



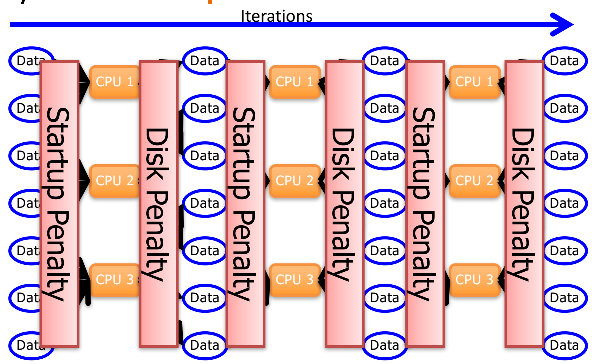
MapAbuse: Iterative MapReduce

- **Only a subset** of data needs computation:



MapAbuse: Iterative MapReduce

- System is **not optimized for iteration:**



ML Tasks Beyond Data-Parallelism

Data-Parallel — Graph-Parallel

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

GraphLab

Graphical Models
Gibbs Sampling
Belief Propagation
Variational Opt.

Semi-Supervised Learning
Label Propagation
CoEM

Collaborative Filtering
Tensor Factorization

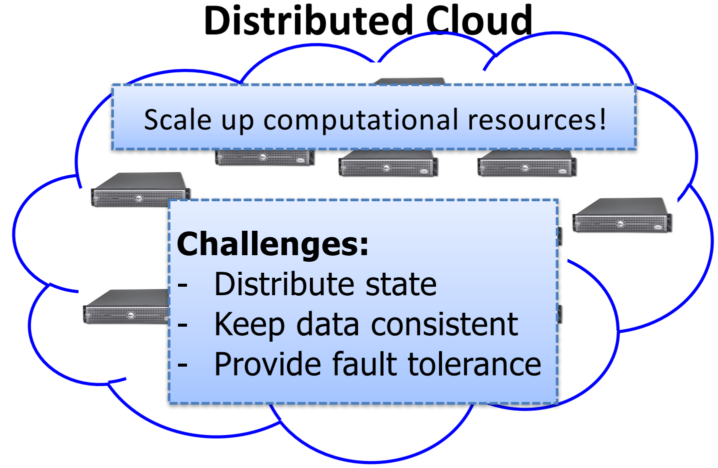
Graph Analysis
PageRank
Triangle Counting

22

- **Limited** CPU Power
- **Limited** Memory
- **Limited** Scalability

23

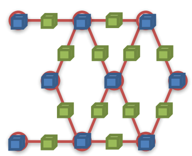
Distributed Cloud



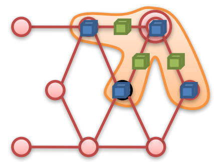
24

The GraphLab Framework

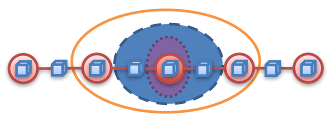
Graph Based
Data Representation



Update Functions
User Computation



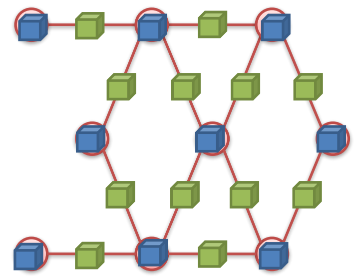
Consistency Model



25

Data Graph

Data is associated with both vertices and edges



Graph:
• Social Network

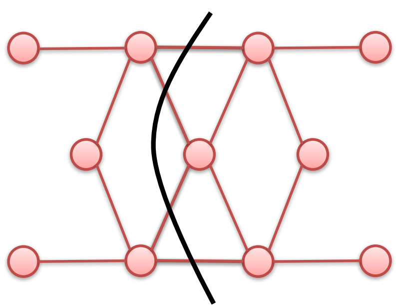
Vertex Data:
• User profile
• Current interests estimates

Edge Data:
• Relationship
(friend, classmate, relative)

26

Distributed Data Graph

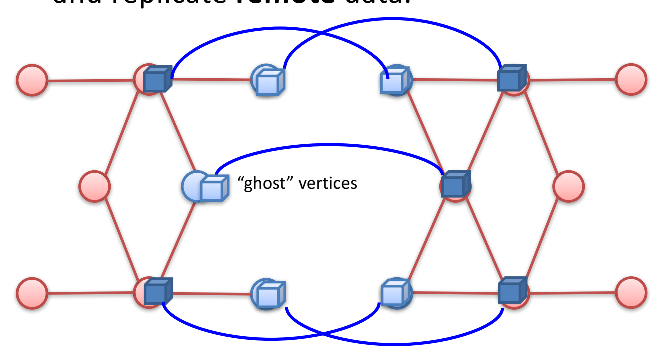
Partition the graph across multiple machines:



27

Distributed Data Graph

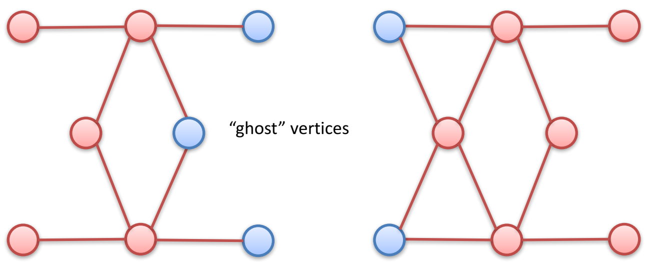
- ***Ghost vertices*** maintain adjacency structure and replicate **remote** data.



28

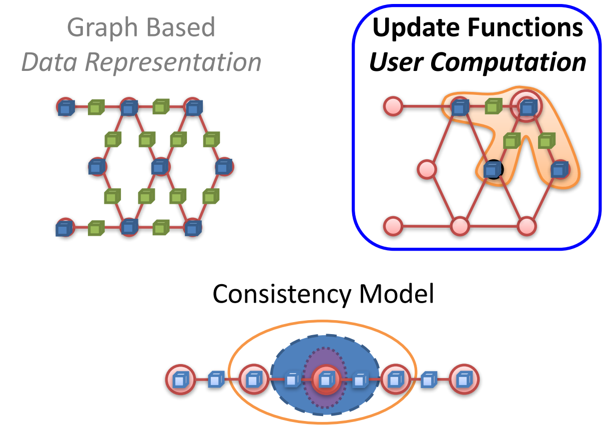
Distributed Data Graph

- Cut efficiently using **HPC Graph partitioning** tools (ParMetis / Scotch / …)



29

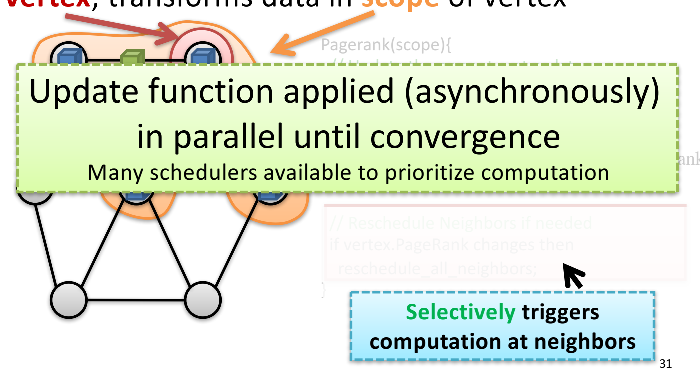
The GraphLab Framework



30

Update Function

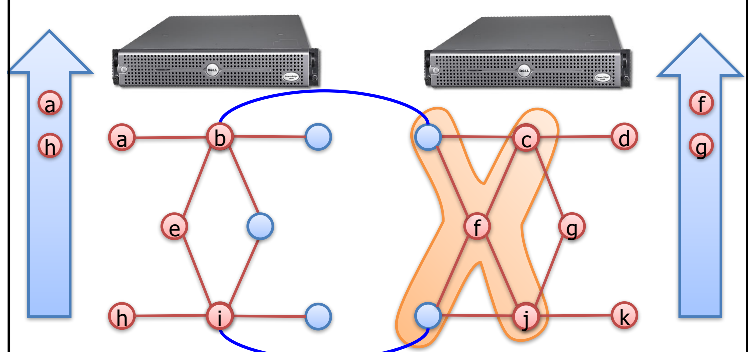
A user-defined **program,** applied to a **vertex**; transforms data in **scope** of vertex



31

Distributed Scheduling

Each machine maintains a **schedule** over the vertices it **owns**

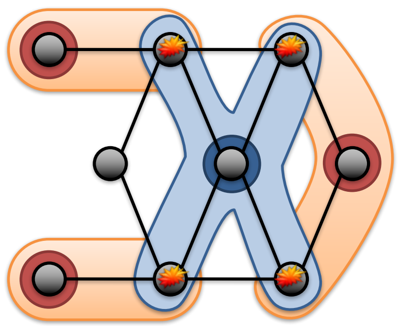


Distributed Consensus used to identify completion

32

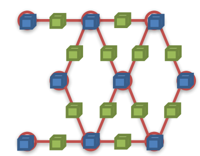
Ensuring Race-Free Code

- How much can computation overlap?

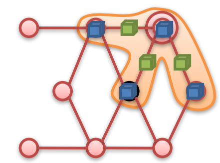


The GraphLab Framework

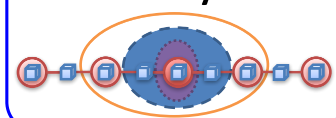
Graph Based
Data Representation



Update Functions
User Computation



Consistency Model



PageRank Revisited

Pagerank(scope) {

$\text{vertex.PageRank} = \alpha$

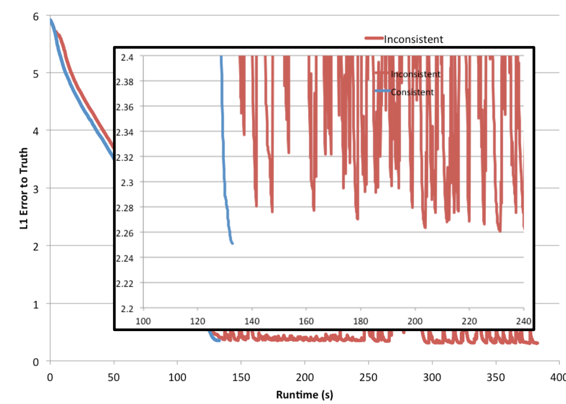
ForEach inPage:

$\text{vertex.PageRank} \mathrel{+}= (1-\alpha) \times \text{inPage.PageRank}$

...

}

PageRank data races confound convergence



Racing PageRank: Bug

Pagerank(scope) {

$\boxed{\text{vertex.PageRank}} = \alpha$

ForEach inPage:

$\boxed{\text{vertex.PageRank}} \mathrel{+}= (1-\alpha) \times \text{inPage.PageRank}$

…

}

37

Racing PageRank: Bug Fix

Pagerank(scope) {

tmp $= \alpha$

ForEach inPage:

tmp $\mathrel{+}= (1-\alpha) \times \text{inPage.PageRank}$

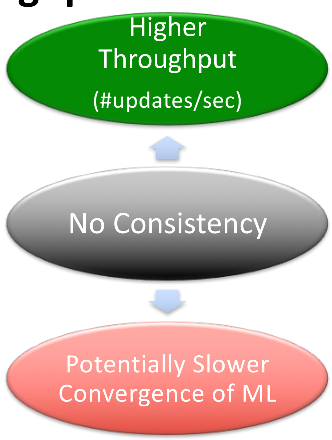
$\boxed{\text{vertex.PageRank} = \text{tmp}}$

…

}

38

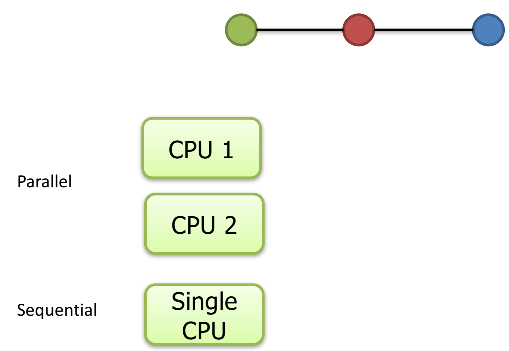
Throughput != Performance



39

Serializability

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



40

Serializability Example

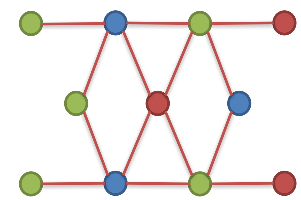
Update functions ***one*** vertex apart can be run in parallel.

Edge Consistency

41

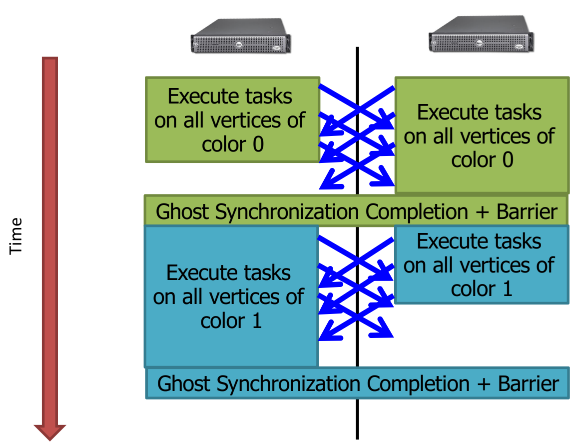
Distributed Consistency

- **Solution 1:** ***Chromatic Engine***
  - Edge Consistency via **Graph Coloring**



- Solution 2: Distributed Locking

Chromatic Distributed Engine

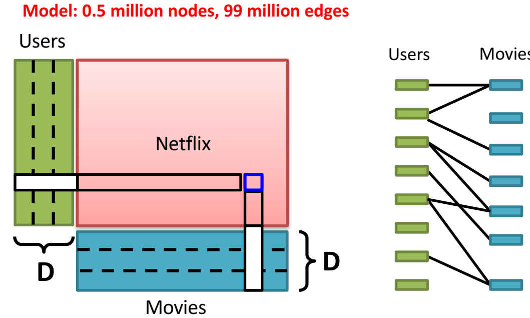


43

Matrix Factorization

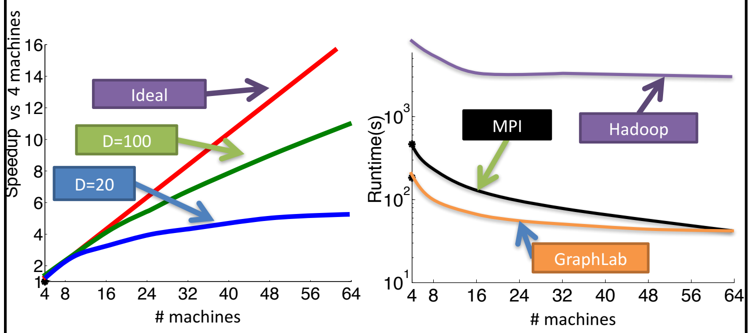
- **Netflix Collaborative Filtering**
  - Alternating Least Squares Matrix Factorization

Model: 0.5 million nodes, 99 million edges



44

Netflix Collaborative Filtering



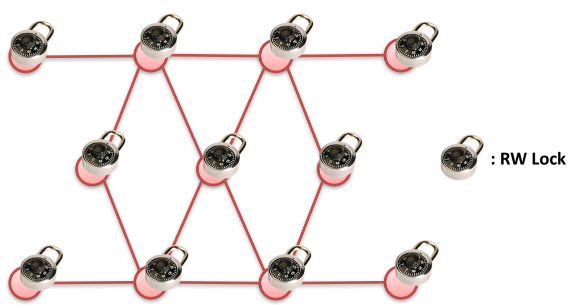
(D = 20)

Distributed Consistency

- **Solution 1:** ***Chromatic Engine***
  - Edge Consistency via **Graph Coloring**
  - **Requires a graph coloring** to be available
  - Frequent barriers → **inefficient** when only **some** vertices active
- **Solution 2: Distributed Locking**

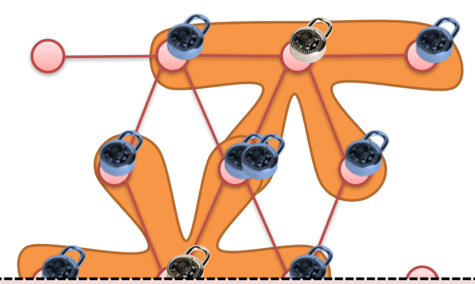
Distributed Locking

Edge Consistency can be guaranteed through locking.



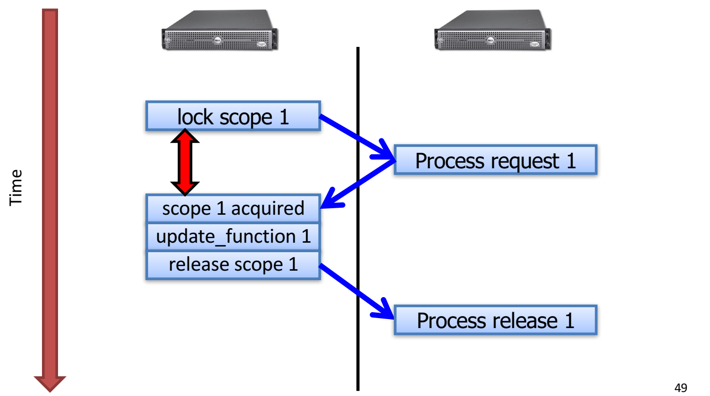
Consistency Through Locking

Acquire write-lock on center vertex, read-lock on adjacent.



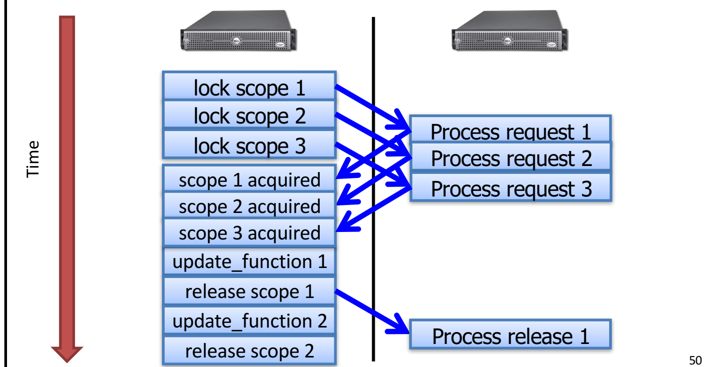
Performance problem: Acquiring a lock from a **neighboring** machine incurs a **latency penalty**

Simple locking



Pipelining hides latency

GraphLab Idea: Hide latency using pipelining



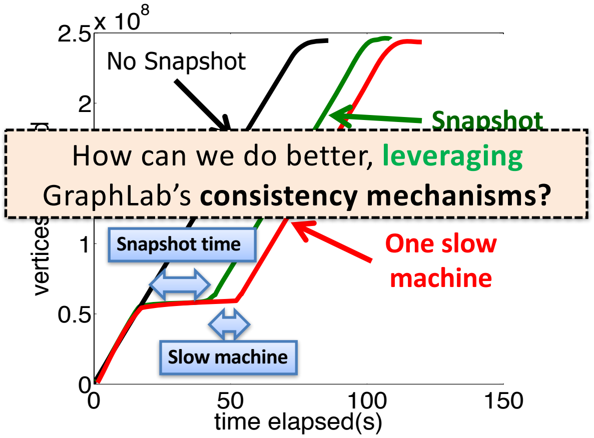
Distributed Consistency

- Solution 1: *Chromatic Engine*
  - Edge Consistency via **Graph Coloring**
  - **Requires a graph coloring** to be available
  - Frequent barriers → **inefficient** when only **some** vertices active
- **Solution 2: Distributed Locking**
  - Residual BP on 190K-vertex/560K-edge graph, 4 machines
  - No pipelining: 472 sec; **with pipelining: 10 sec**

How to handle machine failure?

- *What when machines fail?* **How** do we **provide fault tolerance?**
- Strawman scheme: ***Synchronous snapshot*** checkpointing
  1. Stop the world
  2. Write each machines' state to disk

Snapshot Performance



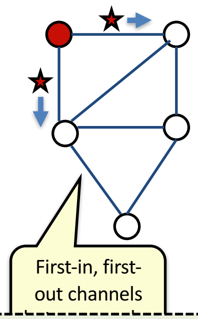
How can we do better, leveraging GraphLab's **consistency mechanisms?**

Chandy-Lamport checkpointing

Step 1. Atomically one *initiator*

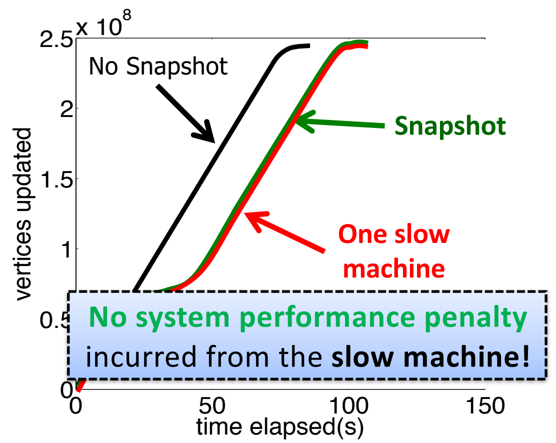
(a) Turns red, (b) Records its own state
(c) sends *marker* to neighbors

Step 2. On receiving marker **non-red** node atomically: (a) Turns red,
(b) Records its own state, (c) sends markers along all outgoing channels



Implemented within GraphLab as an Update Function

Async. Snapshot Performance



No system performance penalty incurred from the **slow machine!**

Summary

- Two different methods of achieving **consistency**
  - Graph Coloring
  - Distributed Locking with pipelining
- **Efficient implementations**
- **Asynchronous FT** w/fine-grained Chandy-Lamport

Performance | **Efficiency** | **Scalability**

Useability

Friday Precept:
Roofnet performance
More Graph Processing

Monday topic:
Streaming Data Processing

57