GraphLab: A New Framework for Parallel Machine Learning

YUCHENG LOW, JOSEPH GONAZLEZ, AAPO KYROLA, DANNY BICKSON, CARLOS GUESTRIN, JOSEPH M. HELLERSTEIN

PRESENTED BY: STEPHEN MACKE

The Problem

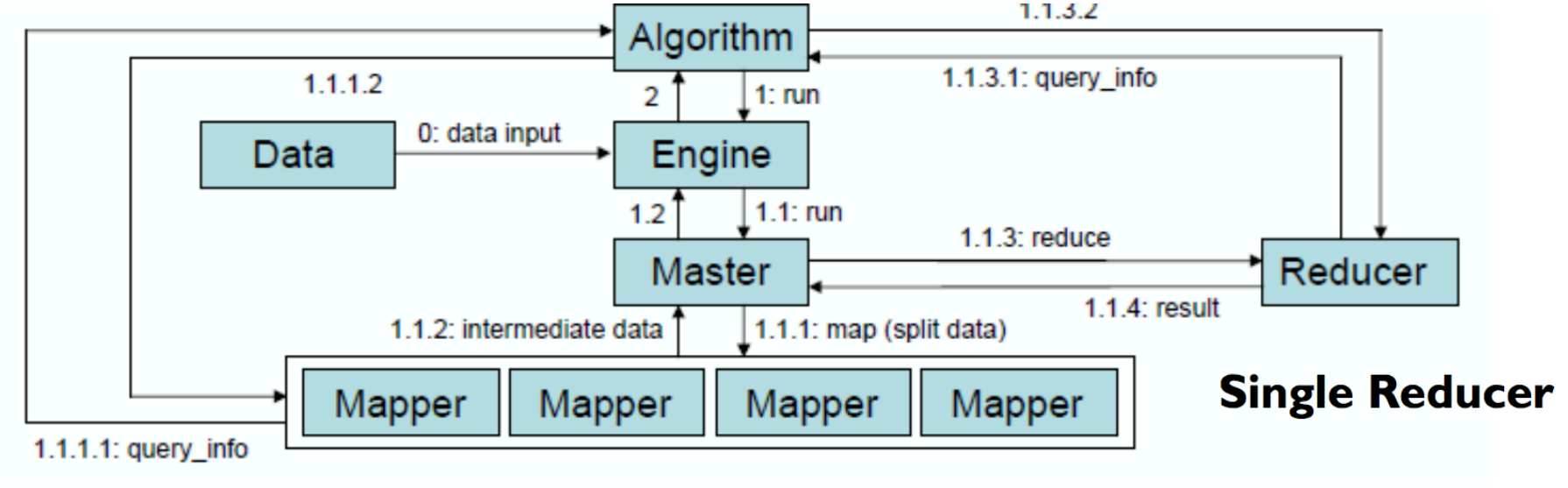
- Need to perform machine learning at scale
- Hardware advances hitting power wall
  - Need to parallelize (multicore + distributed)
  - But parallelization is hard!

Low-level approaches

- Pthreads
- MPI
- (OpenMP)
- (CUDA, SSE instructions)

Solution: MapReduce?

- Only works well for embarassingly parallel problems
- Iterative MapReduce typically requires global synchronization at a single reducer



Other Abstractions

- Systolic, Dataflow
  - Nodes vertices, edges communication links
  - **Bulk synchronous – bad for "Gauss-Seidel"-style schedules**
- DAG Abstraction (Dryad, Pig Latin)
  - Dataflow graph depends on # iterations
  - Must be specified in advance – not suitable for iterative computation
- Graph-based messaging (e.g. Pregel)
  - Require users to explicitly manage communication

Digression: Jacobi vs. Gauss-Seidel schedules

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nn}x_n &= b_n
\end{aligned}
$$

Digression: Jacobi vs. Gauss-Seidel schedules

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots a_{2n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots a_{n,n-1}x_{n-1})$$

Digression: Jacobi vs. Gauss-Seidel schedules

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots a_{2n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots a_{n,n-1}x_{n-1})$$

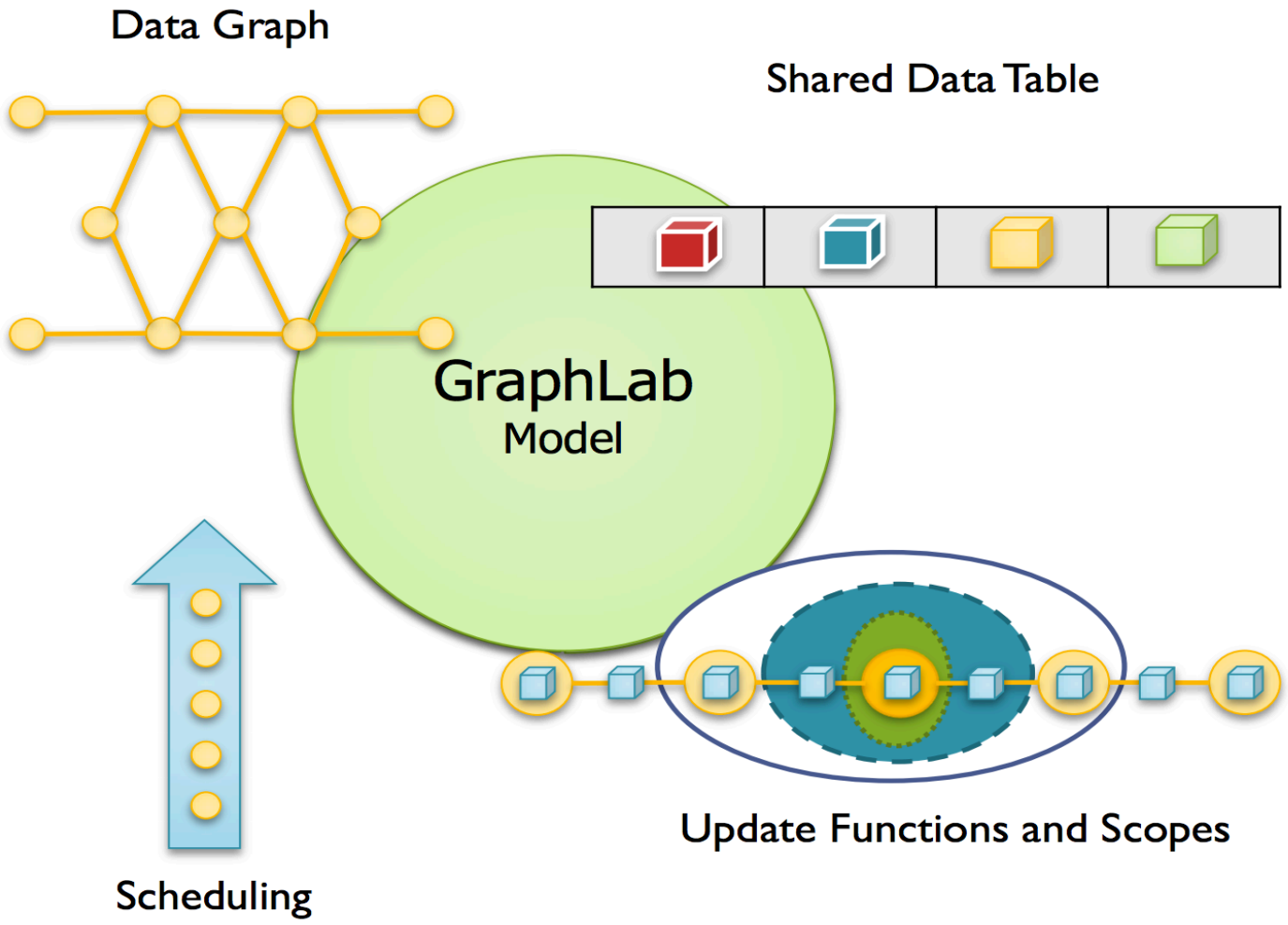
The Main Idea

Perform iterative updates in this “Gauss-Seidel” fashion, asynchronously.

Other Abstractions (cont'd)

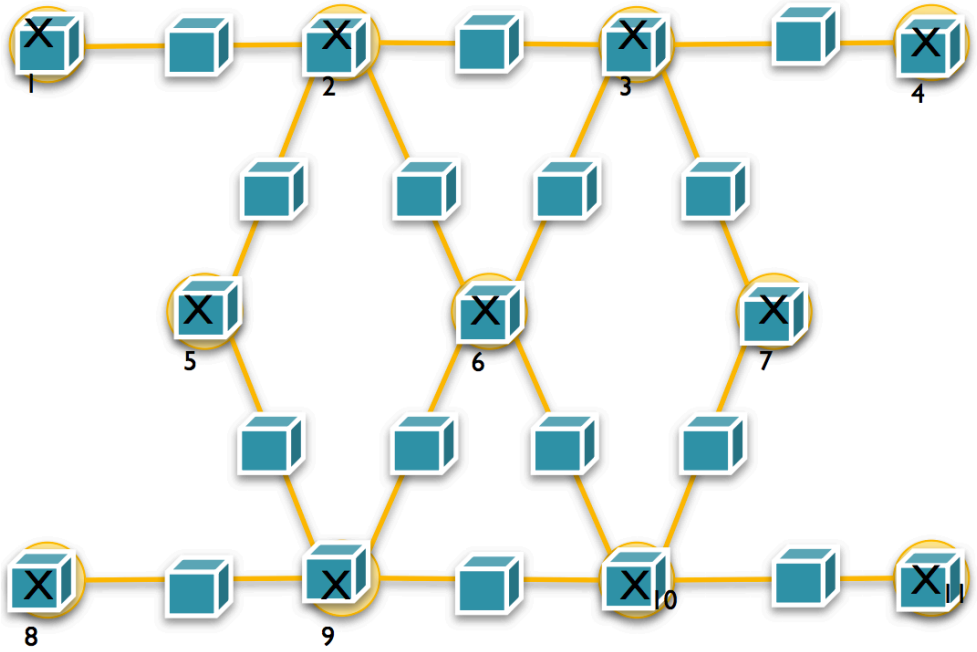
	Computation Model	Sparse Depend.	Async. Comp.	Iterative	Prioritized Ordering	Sequentially Consistent [a]	Distributed
MPI[1]	Messaging	Yes	Yes	Yes	N/A [b]	N/A [b]	Yes
MapReduce[2]	Par. data-flow	No	No	extensions[c]	N/A	N/A	Yes
Dryad[3]	Par. data-flow	Yes	No	extensions[d]	N/A	N/A	Yes
Pregel[4]/BPGL[5]	GraphBSP[6]	Yes	No	Yes	N/A	N/A	Yes
Piccolo[7]	Distr. map[f]	N/A [f]	Yes	Yes	No	accumulators	Yes
Pearce et.al.[8]	Graph Visitor	Yes	Yes	Yes	Yes	No	No
GraphLab	**GraphLab**	Yes	Yes	Yes	Yes[e]	Yes	Yes [g]

GraphLab Components



Data Graph

- A Graph with data associated with every **vertex** and **edge**.

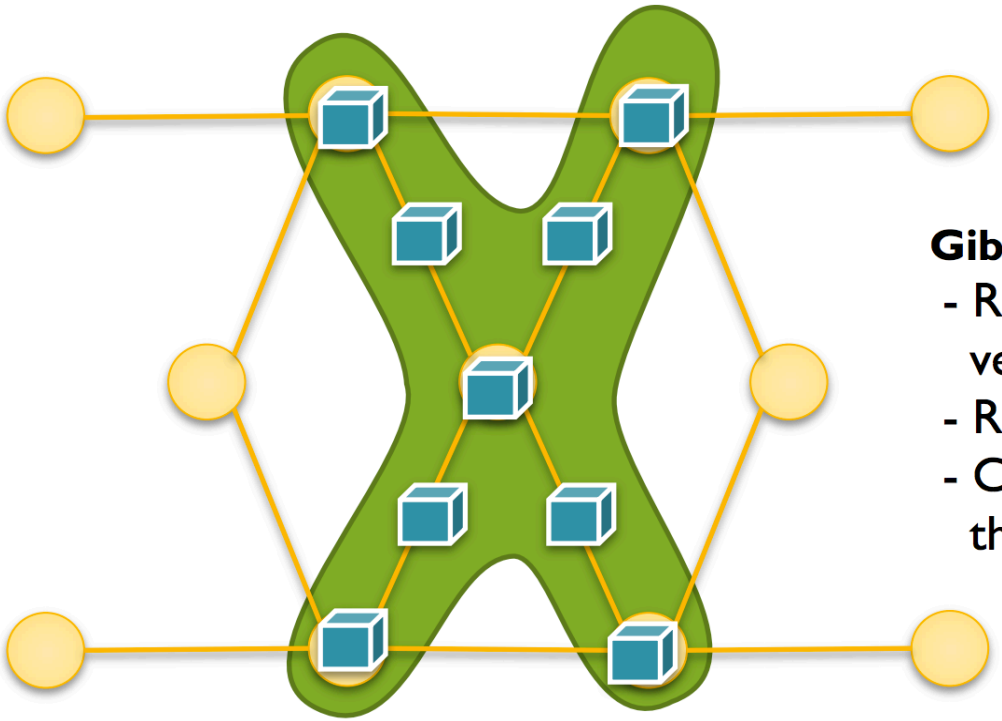


x_3: Sample value
$C(X_3)$: sample counts

$\Phi(X_6,X_9)$: Binary potential

User Update Functions

- Operations applied on a **vertex** that transform data in the **scope** of the vertex



Gibbs Update:
- Read samples on adjacent vertices
- Read edge potentials
- Compute a new sample for the current vertex

Sync Operation

- "Fold" operation aggregates over vertices
  - Analogous to "Reduce" from MapReduce
- "Merge" operation for parallel tree reduction
  - Analogous to "Combine" from MapReduce
- "Apply" finalizes value (e.g. rescaling)

$$
\begin{aligned}
r_k^{(i+1)} &\leftarrow \text{Fold}_k\left(D_v, r_k^{(i)}\right) \\
r_k^{l} &\leftarrow \text{Merge}_k\left(r_k^{i}, r_k^{j}\right) \\
\mathbf{T}\left[k\right] &\leftarrow \text{Apply}_k(r_k^{(|V|)})
\end{aligned}
$$

Let's write Update for Pagerank!

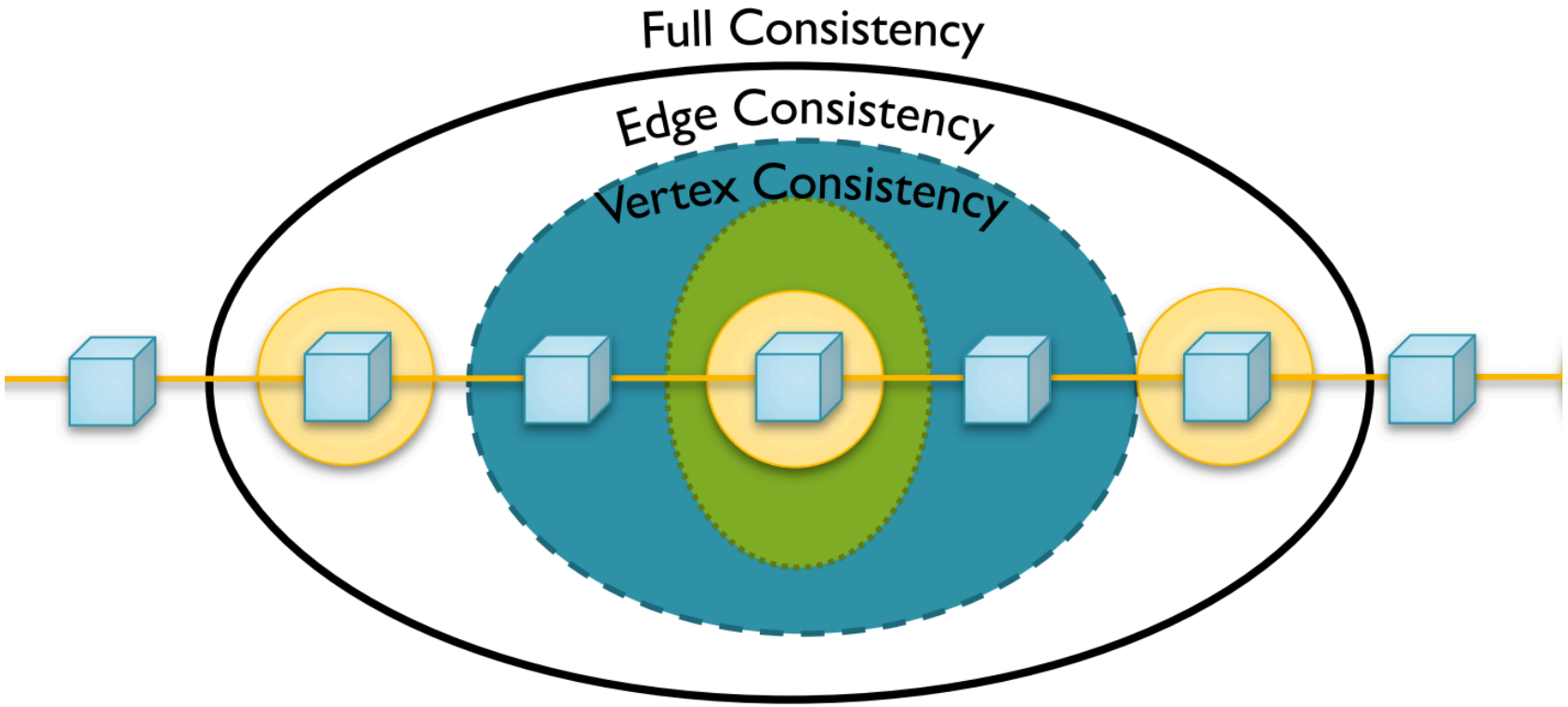
- Reminder: pagerank eqns

$$\boldsymbol{R}(v) = \frac{\alpha}{n} + (1-\alpha) \sum_{u \text{ links to } v} w_{u,v} \times \boldsymbol{R}(u)$$

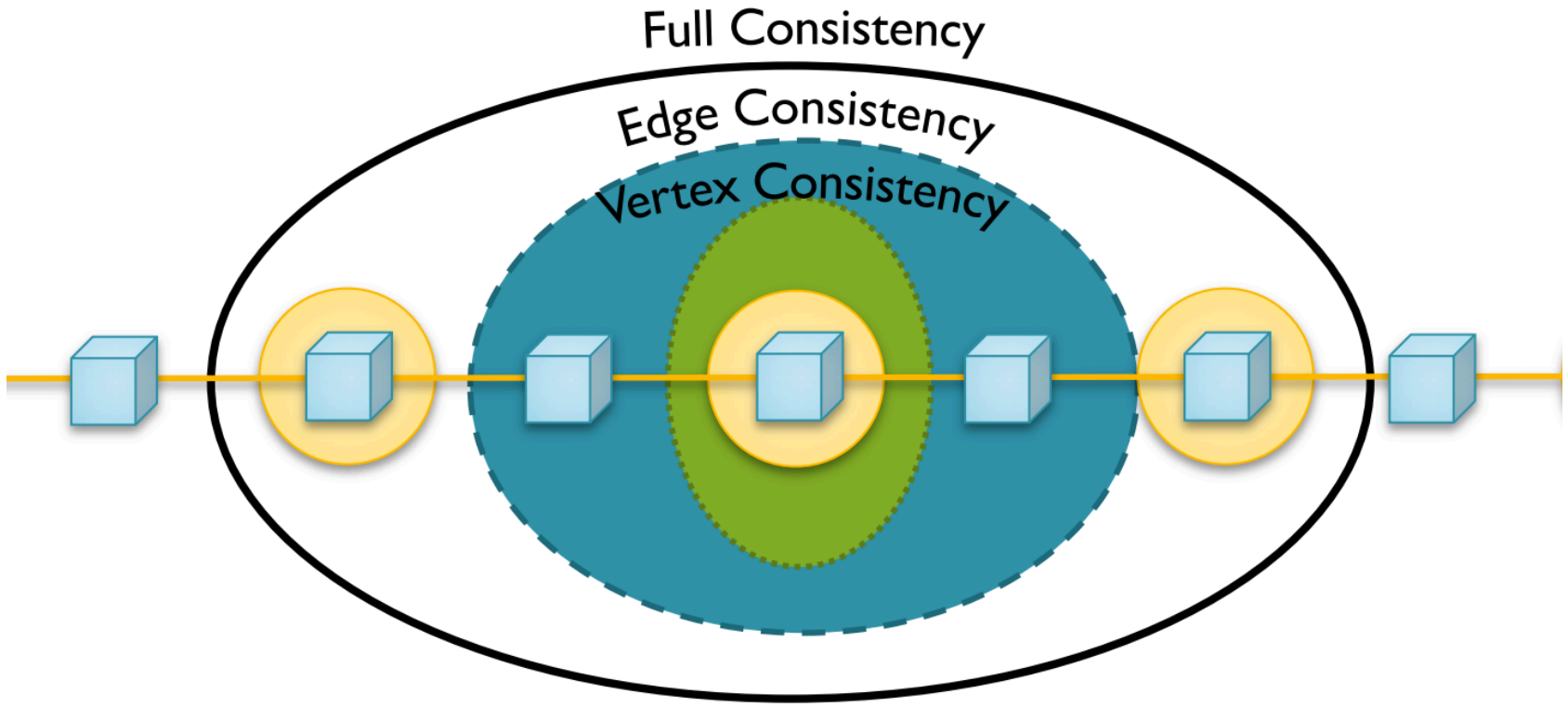
Let's write Update for Pagerank!

Input: Vertex data $\mathbf{R}(v)$ from $\mathcal{S}_v$
Input: Edge data $\{w_{u,v} : u \in \mathbf{N}[v]\}$ from $\mathcal{S}_v$
Input: Neighbor vertex data $\{\mathbf{R}(u) : u \in \mathbf{N}[v]\}$ from $\mathcal{S}_v$
$\mathbf{R}_{\text{old}}(v) \leftarrow \mathbf{R}(v)$ `// Save old PageRank`
$\mathbf{R}(v) \leftarrow \alpha/n$
foreach $u \in \mathbf{N}[v]$ **do** `// Loop over neighbors`
  $\mathbf{R}(v) \leftarrow \mathbf{R}(v) + (1-\alpha) * w_{u,v} * \mathbf{R}(u)$
`// If the PageRank changes sufficiently`
if $|\boldsymbol{R}(v) - \boldsymbol{R}_{old}(v)| > \epsilon$ **then**
  `// Schedule neighbors to be updated`
  return $\{(\mathit{PageRankFun}, u) : u \in \mathbf{N}[v]\}$
Output: Modified scope $\mathcal{S}_v$ with new $\mathbf{R}(v)$

Scope Rules



Scope Rules



Q. What consistency model do we need for pagerank?

Scheduling

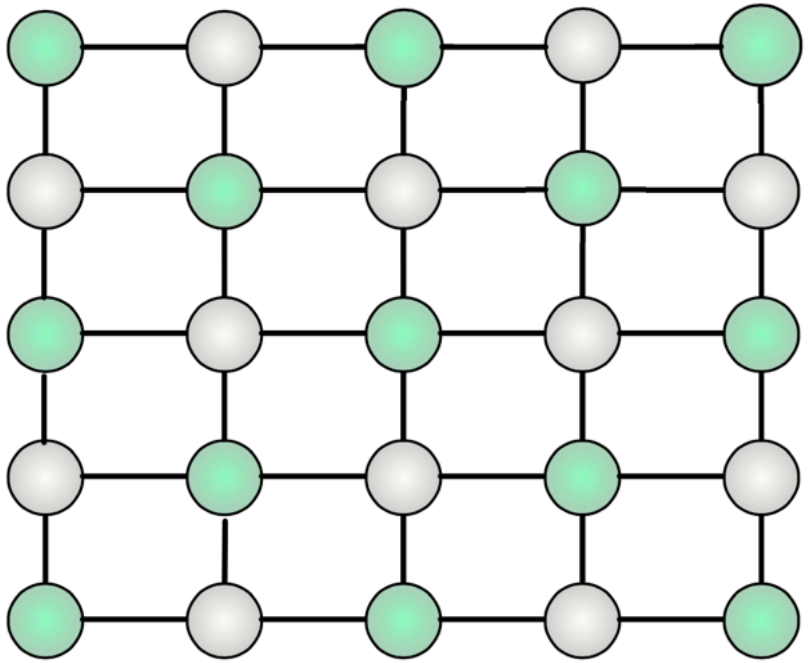
- Synchronous
- Round-robin
- Set scheduler

for $i = 1 \cdots k$ **do**
  Execute f_i on all vertices in S_i in parallel.
  Wait for all updates to complete

- Splash scheduler

Set Scheduling via Graph Coloring

- For certain consistency models (q. which ones?), we can update all vertices of same color in parallel!

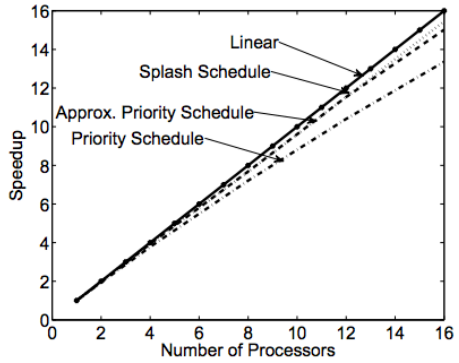


Ex. Gibbs sampling

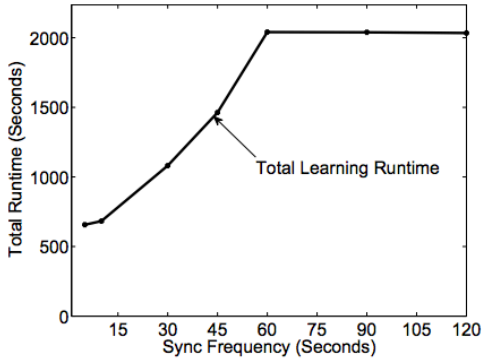
Case Studies

- MRF Parameter learning
- Gibbs sampling
- Co-Em
- Lasso / Shooting
- Compressed Sensing

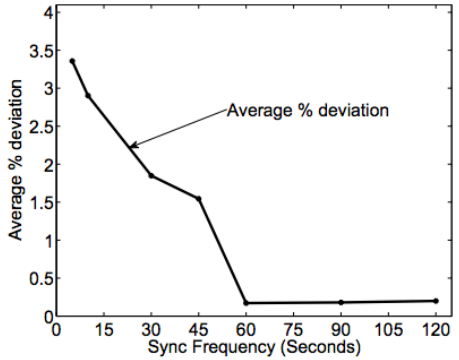
Results (MRF parameter learning)



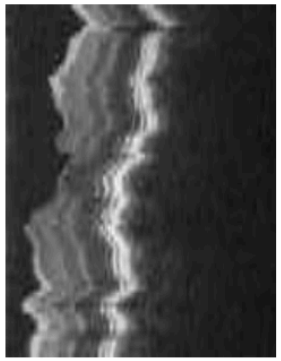
(a) Speedup



(b) Bkgnd Sync. Runtime



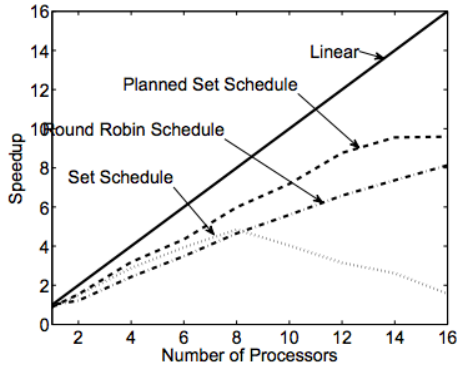
(c) Bkgnd Sync. Error



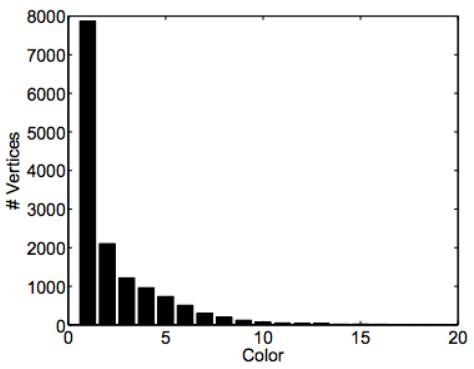
(d) Original

(e) Denoised

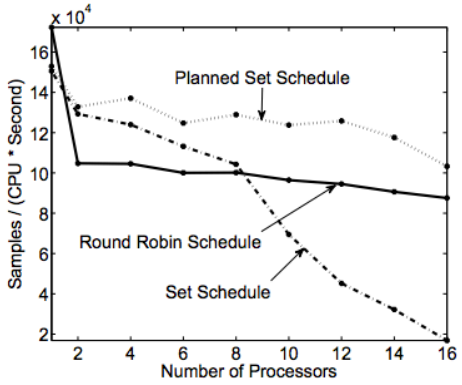
Results (Gibbs sampling, LBP)



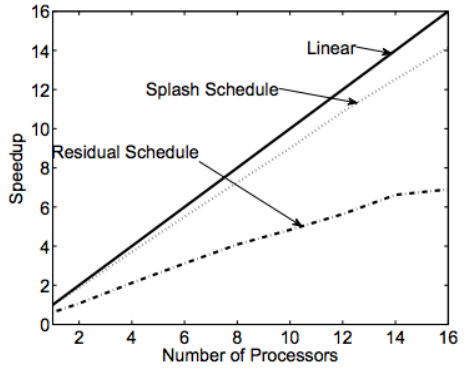
(a) Gibbs Speedup



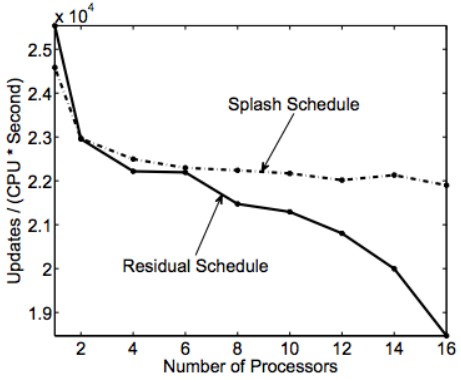
(b) Gibbs Color



(c) Gibbs Eff.

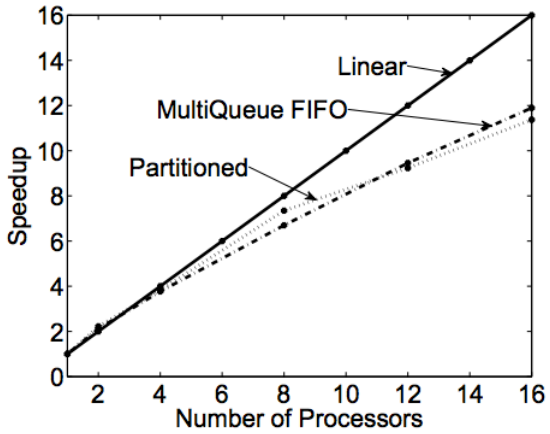


(d) BP Speedup

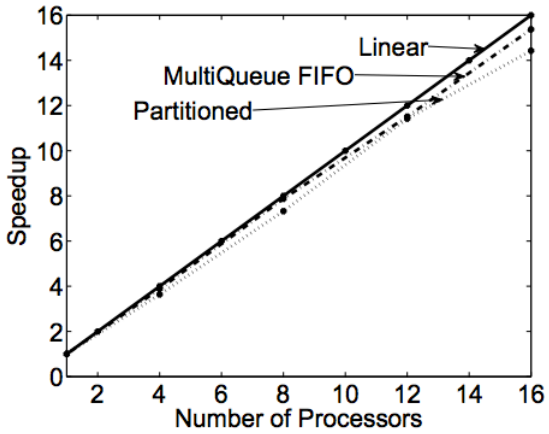


(e) BP Eff.

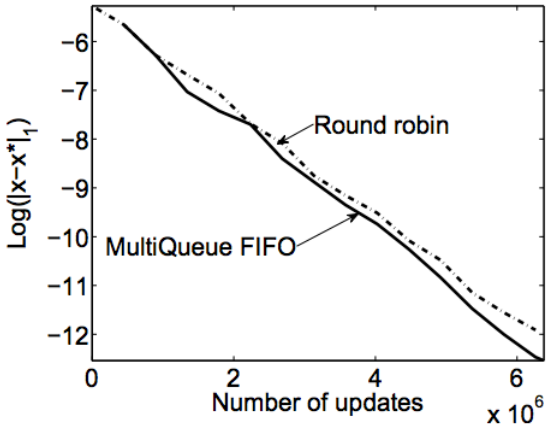
Results (Co-Em)



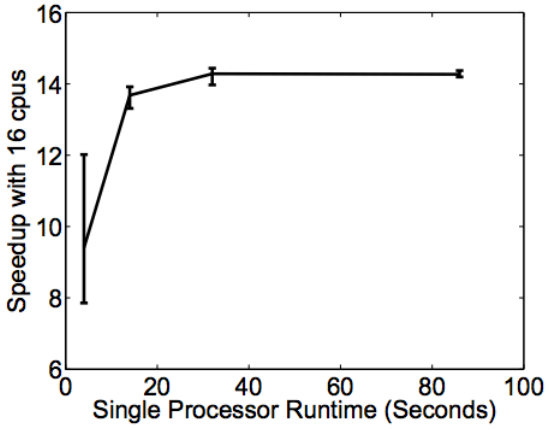
(a) CoEM Speedup Small



(b) CoEM Speedup Large

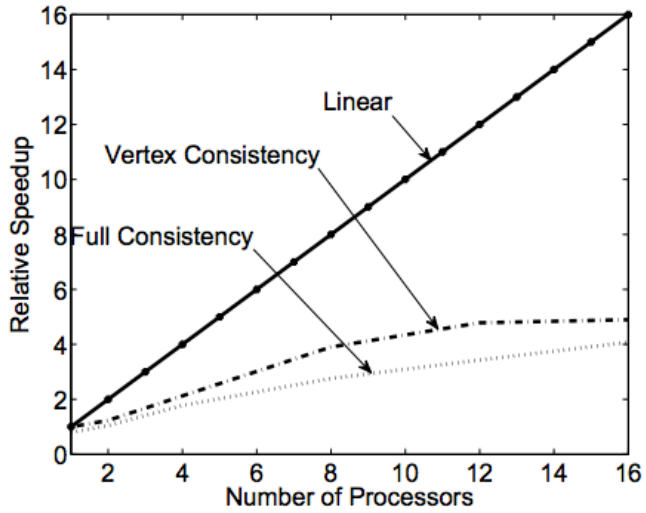


(c) Convergence

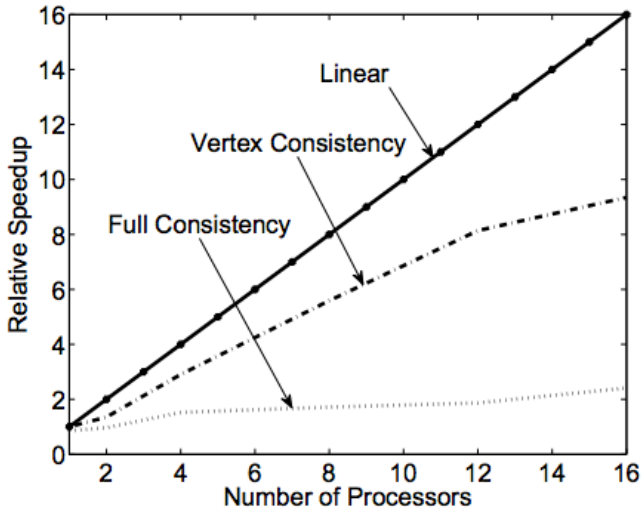


(d) Speedup with Problem Size

Results (Lasso)

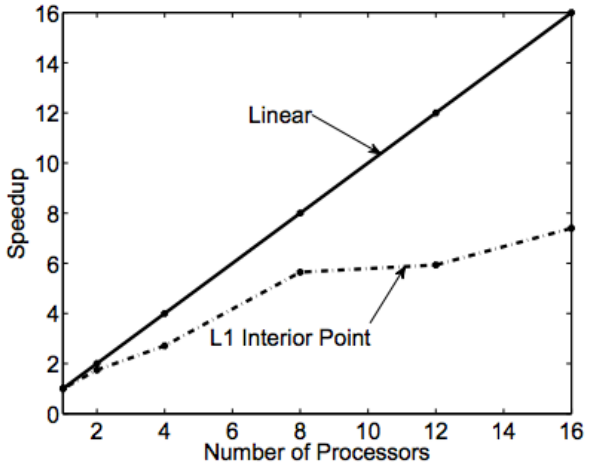


(a) Sparser Dataset Speedup



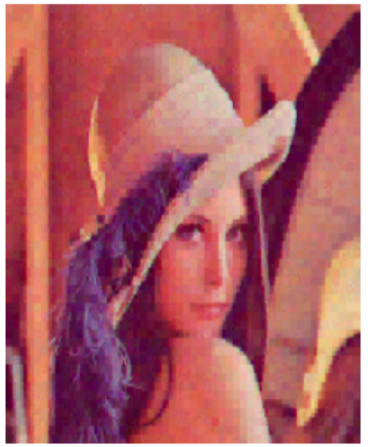
(b) Denser Dataset Speedup

Results (Compressed Sensing)



(a) Speedup

(b) Lenna



(c) Lenna 50%

Conclusion

- Does not say anything new about parallel programming fundamentals
- New interesting abstraction for certain types of problems
  - Shows need for rethinking existing abstractions (e.g. MR)
  - (particularly for iterative / ML algorithms)
- Commercialized as GraphLab Create
  - But mostly for toolkit / viz purposes
  - Actual mucking with update / sync functions somewhat niche
    - Open sourced in PowerGraph, GraphChi, etc.

More Gibbs Stuff