PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

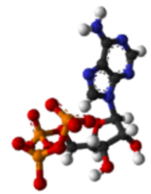
Joseph Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, Carlos Guestrin

CMU, UW

**some figures in the slide deck are borrowed from the official OSDI slides*

What are Natural Graphs?

Social Media | **Science** | **Advertising** | **Web**

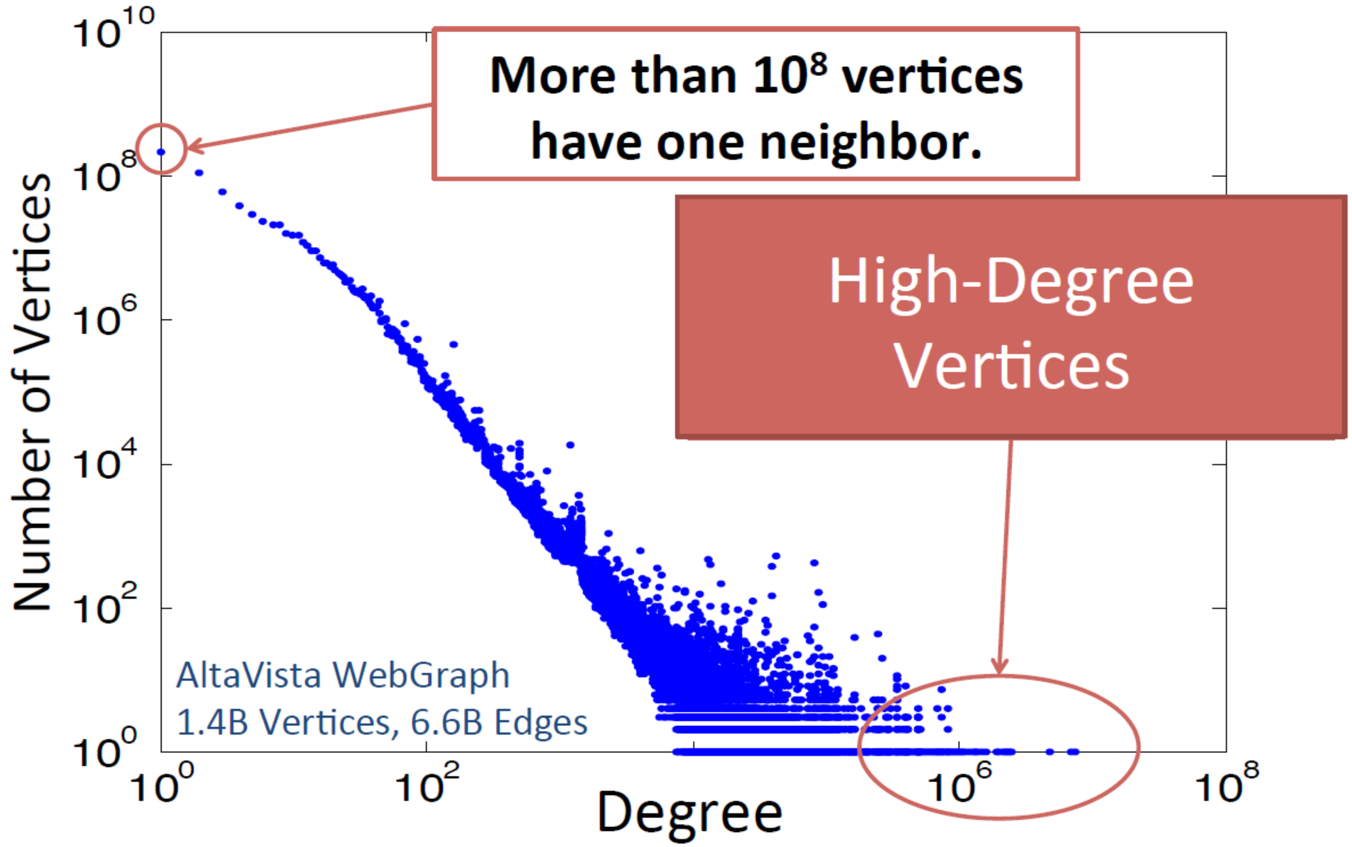


Graphs that are derived from natural phenomena

Such as relationships between:

- People
- Product
- Interests
- Ideas

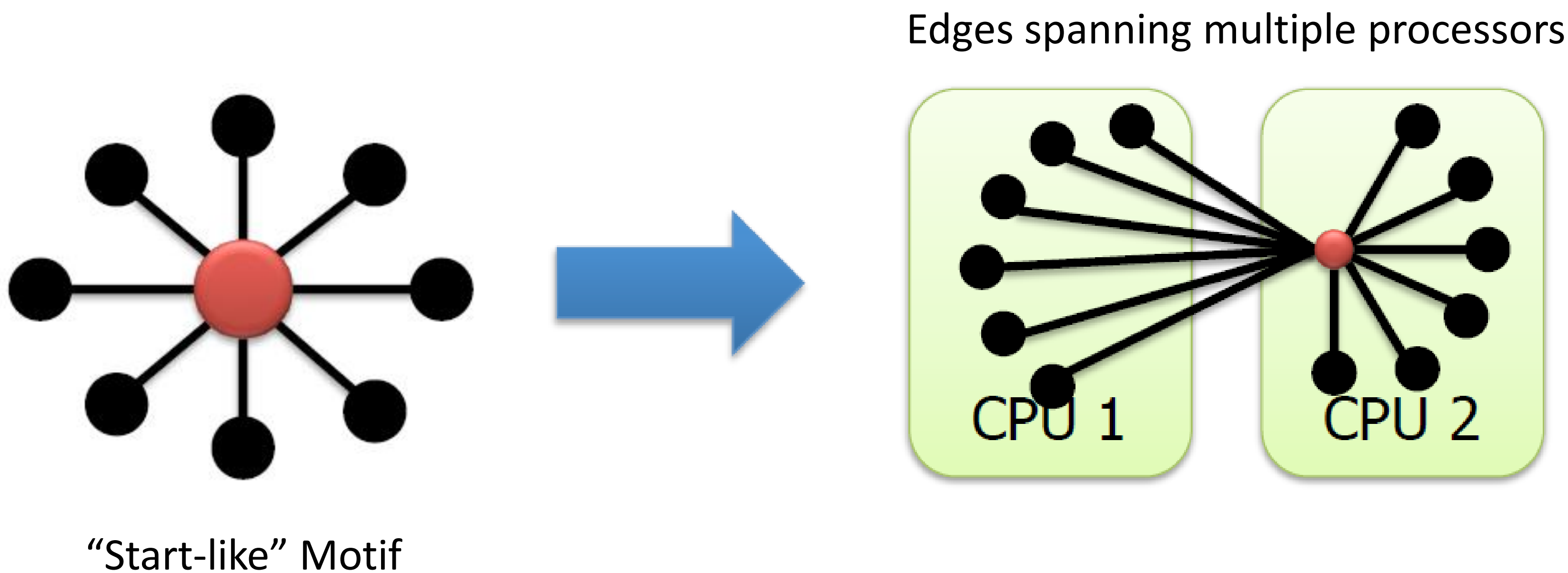
Power-Law Degree Distribution



Most of natural graphs have **skewed power-law degree distribution**

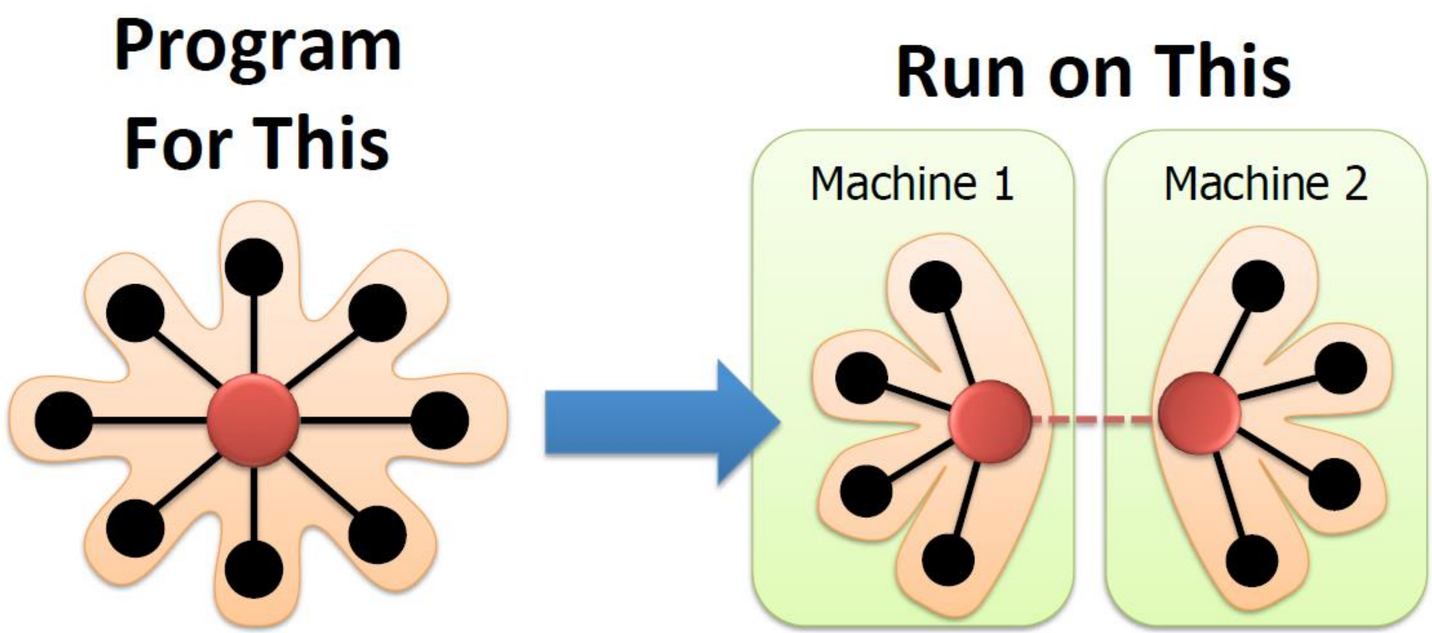
Most vertices have relatively few neighbors, while a few have many neighbors

Problem: Hard to Partition



- Power-law graphs do not have low-cost balanced cuts
- Existing distributed graph computation systems perform poorly on power law graphs

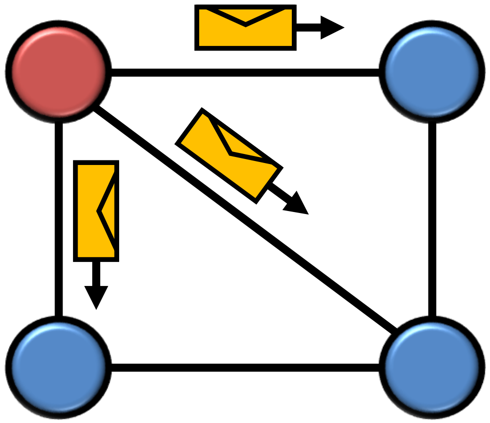
High-Level PowerGrpah Abstraction



- **Split High-Degree Vertices**
- **New abstraction for programming graph computations**

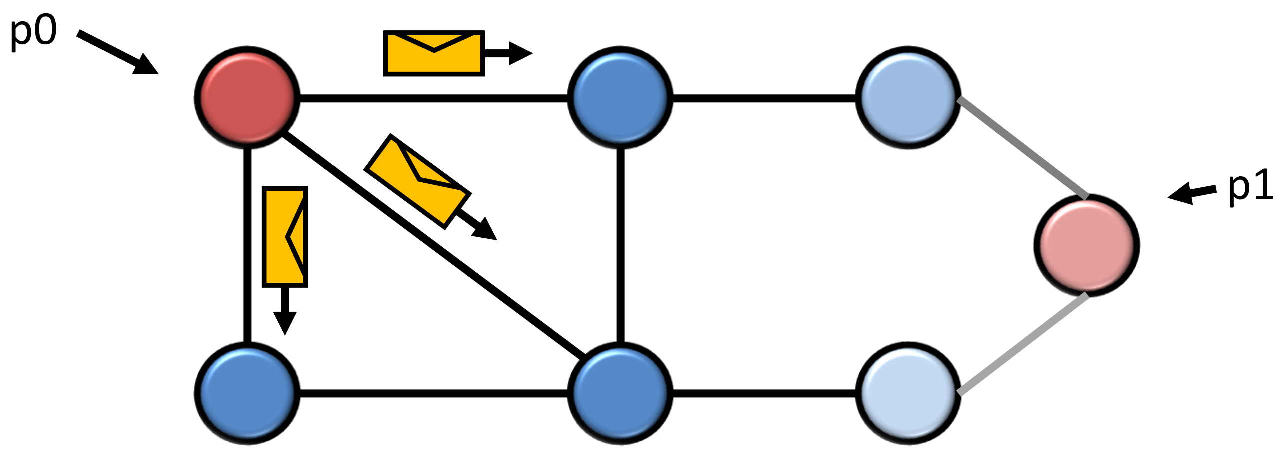
How do we program a graph computation?

- A user-defined Vertex-Program runs on each vertex
- Graph constrains intersctions along edges
  - Using messages (Pregel[PODC09])
  - Using shared state (GraphLab[VLDB12])

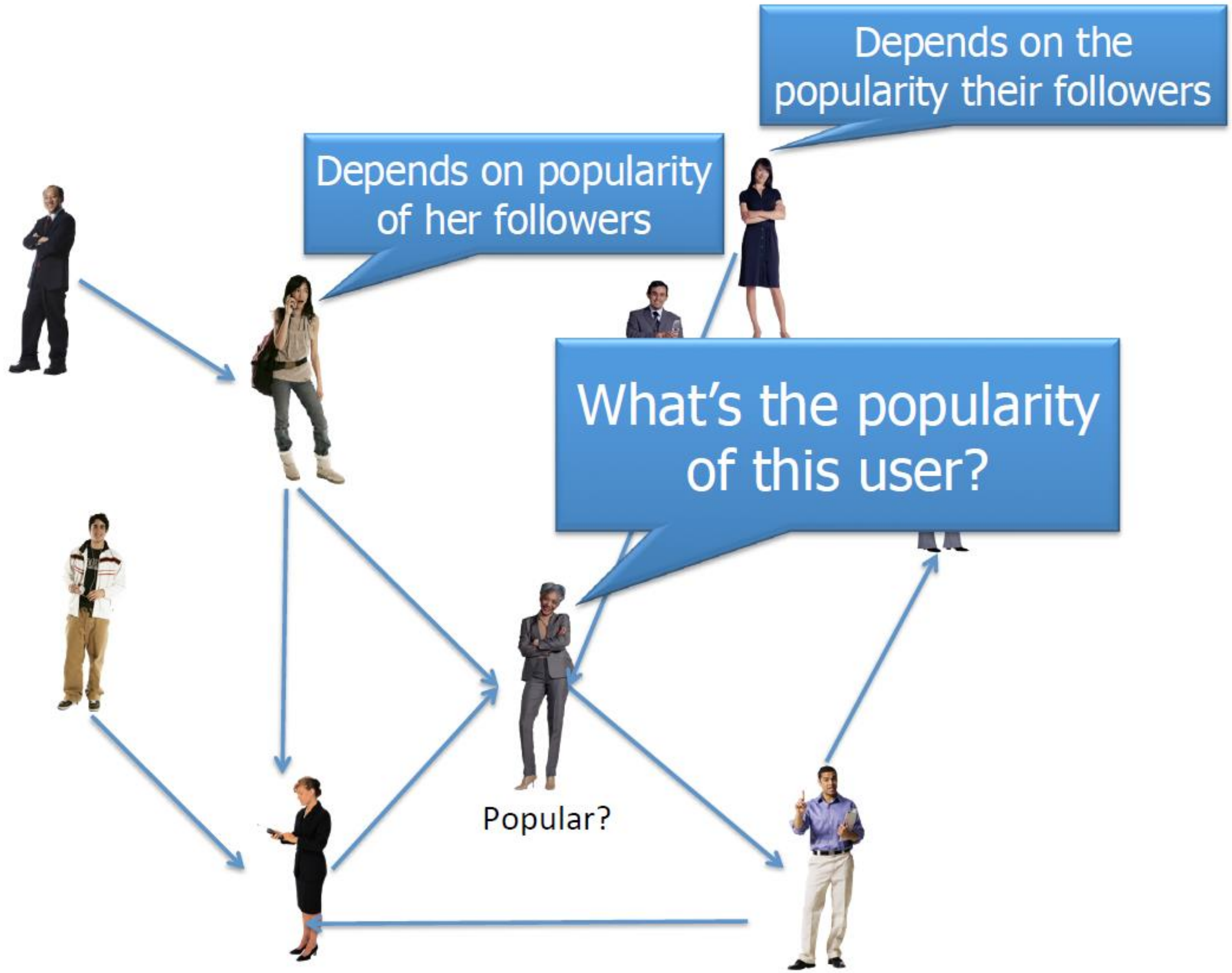


How do we program a graph computation?

- A user-defined Vertex-Program runs on each vertex
- Graph constrains intersctions along edges
  - Using messages (Pregel[PODC09])
  - Using shared state (GraphLab[VLDB12])
- Parallelism: run multiple vertex programs simultaneously



Example Computation: Social Network Popularity



PageRank Algorithm

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$

Rank of user *i*

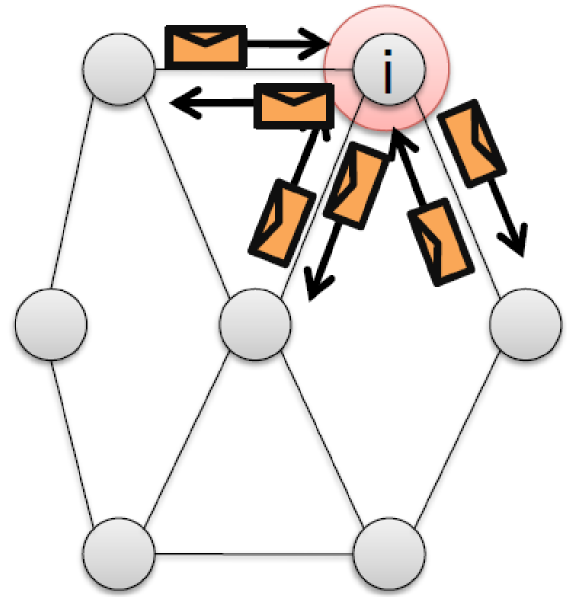
Weighted sum of neighbors' ranks

- Update ranks in parallel
- Iterate until convergence

The Pregel [PODC09] Abstraction

Vertex-Programs interact by sending **messages**.

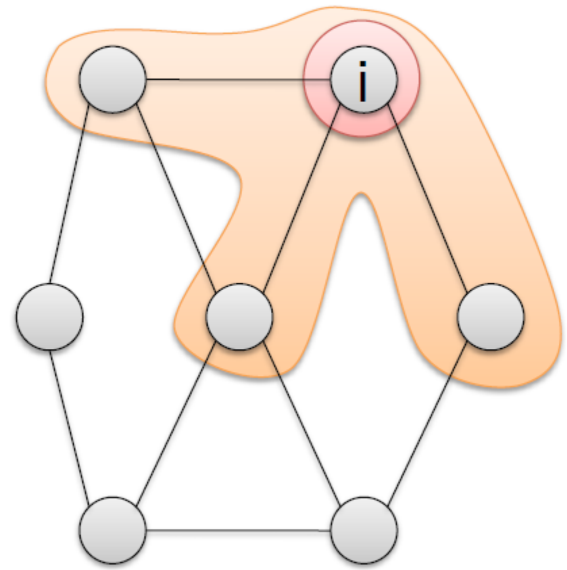
```
Pregel_PageRank(i, messages) :
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // Update the rank of this vertex
  R[i] = 0.15 + total

  // Send new messages to neighbors
  foreach(j in out_neighbors[i]) :
    Send  msg(R[i] * w_ij) to vertex j
```



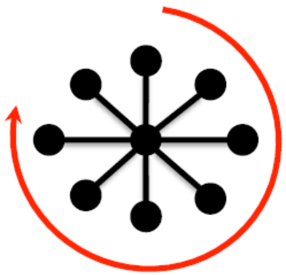
The GraphLab [VLDB12] Abstraction

Vertex-Programs directly **read** the neighbors state

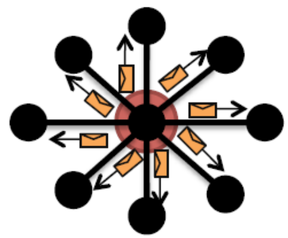
```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in in_neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total

  // Trigger neighbors to run again
  if R[i] not converged then
    foreach( j in out_neighbors(i)):
      signal vertex-program on j
```



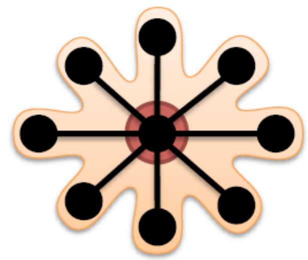
Challenges of High-Degree Vertices



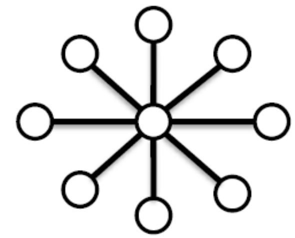
Sequentially process edges



Sends many messages (Pregel)

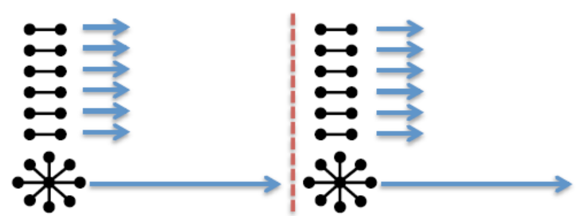


Touches a large fraction of graph (GraphLab)



Edge meta-data too large for single machine

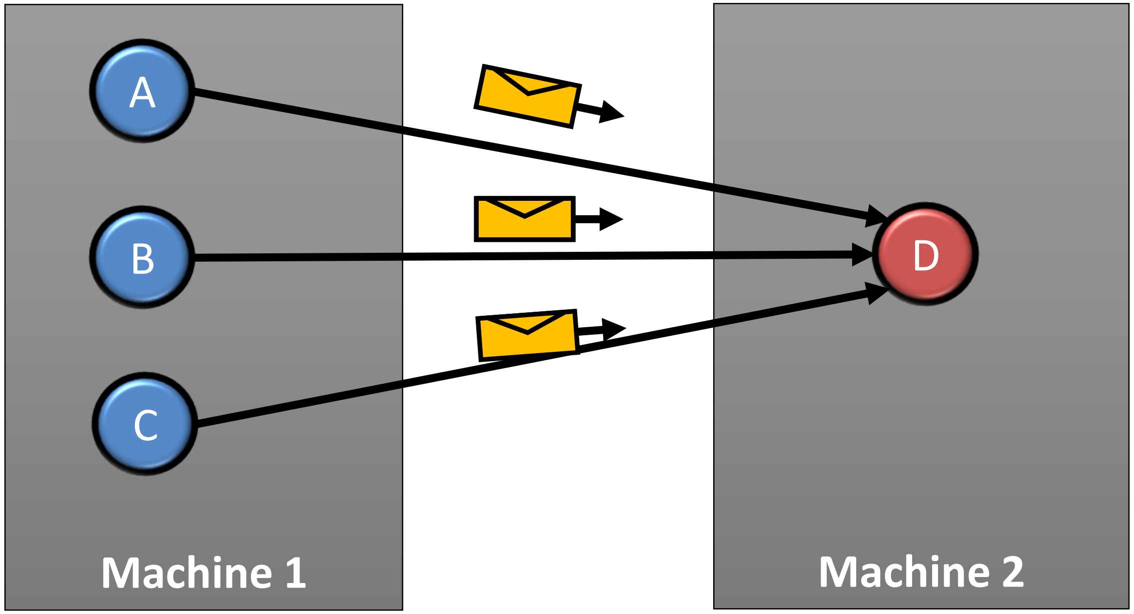
Asynchronous Execution requires heavy locking (GraphLab)



Synchronous Execution prone to stragglers (Pregel)

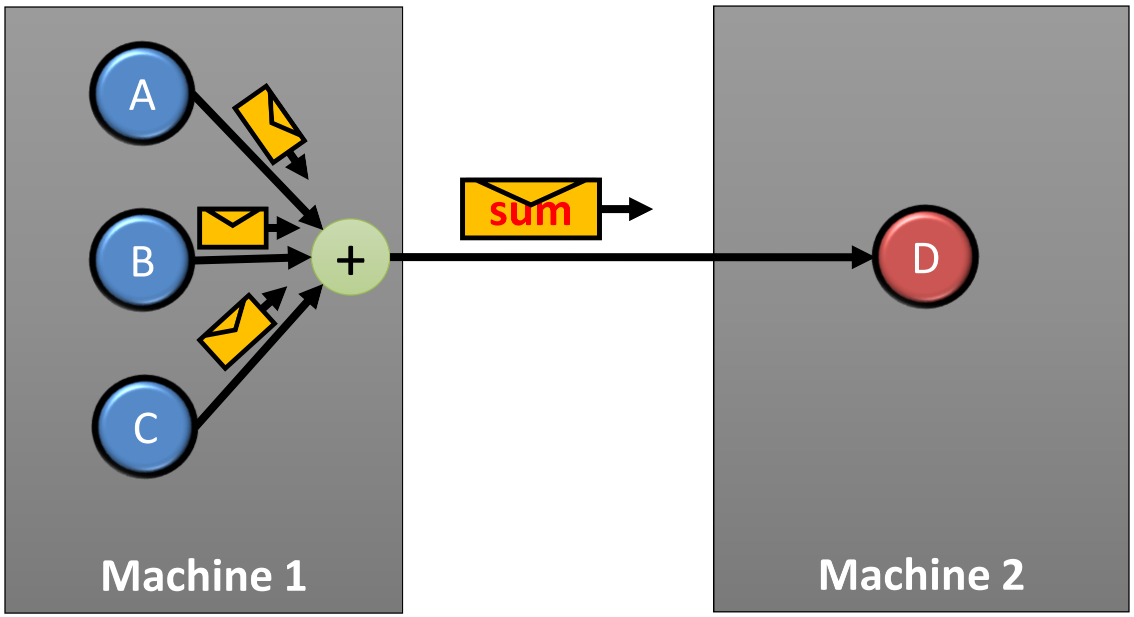
Communication Overhead for High-Degree Vertices is the Most Prominent

Pregel Reduces Fan-In Traffic



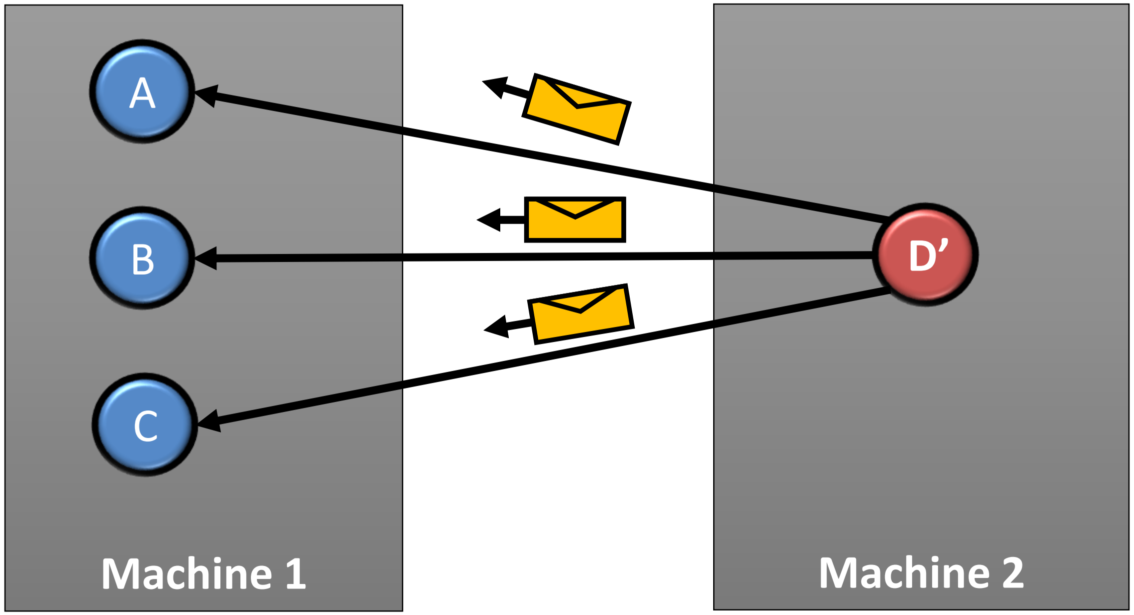
Sending vertex info from neighbors

Pregel Reduces Fan-in Traffic



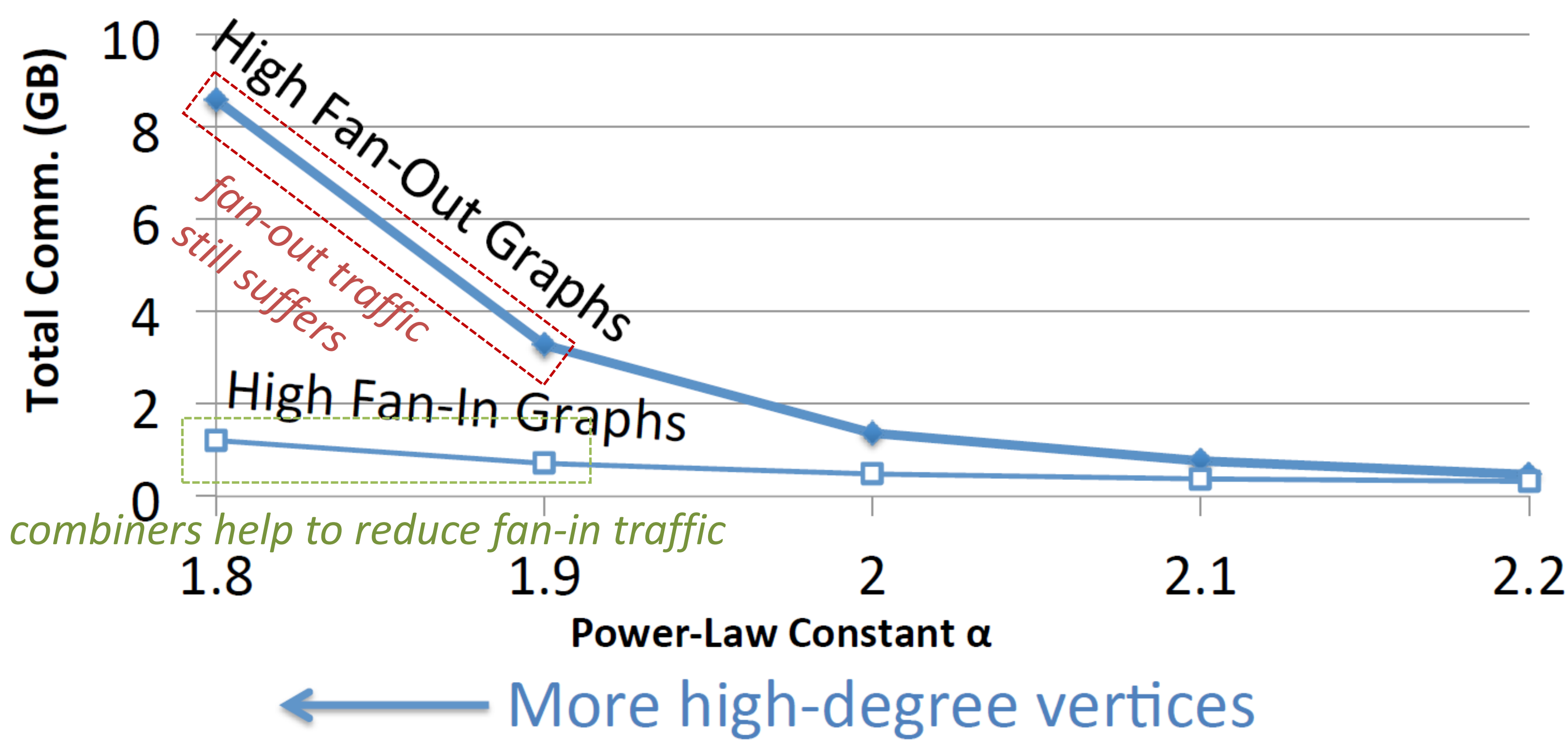
User-defined commutative associative (+) message operation allows preprocessing on the local machine with combiners and reduces the amount of messages transmitted

Pregel Struggles with Fan-Out

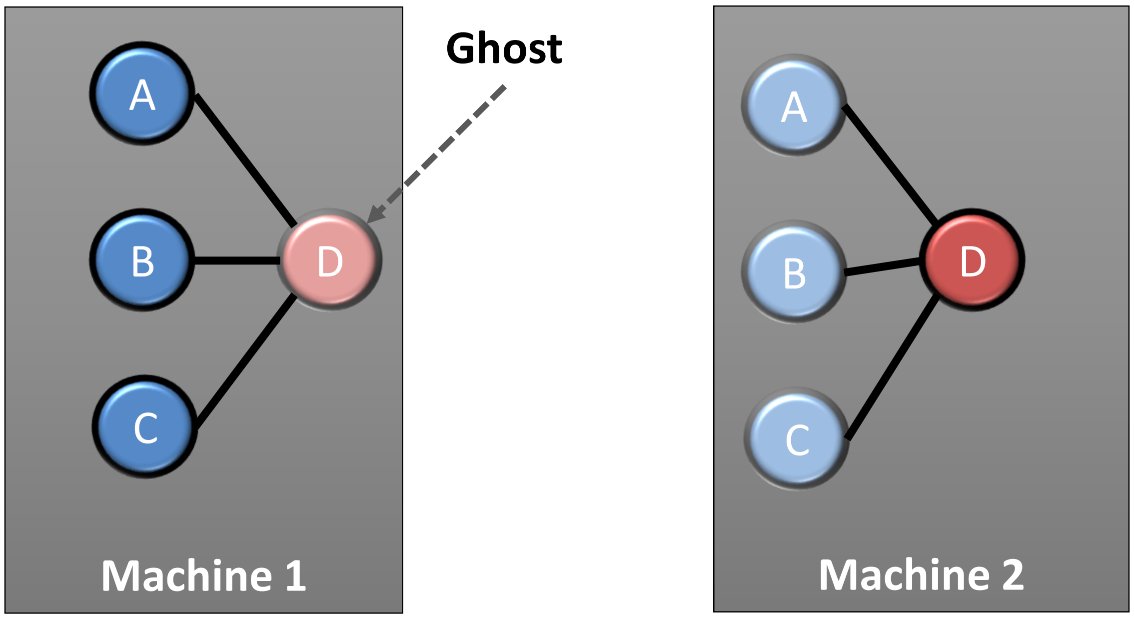


Fan-In and Fan-Out Performance

- PageRank on synthetic Power-law Graphs

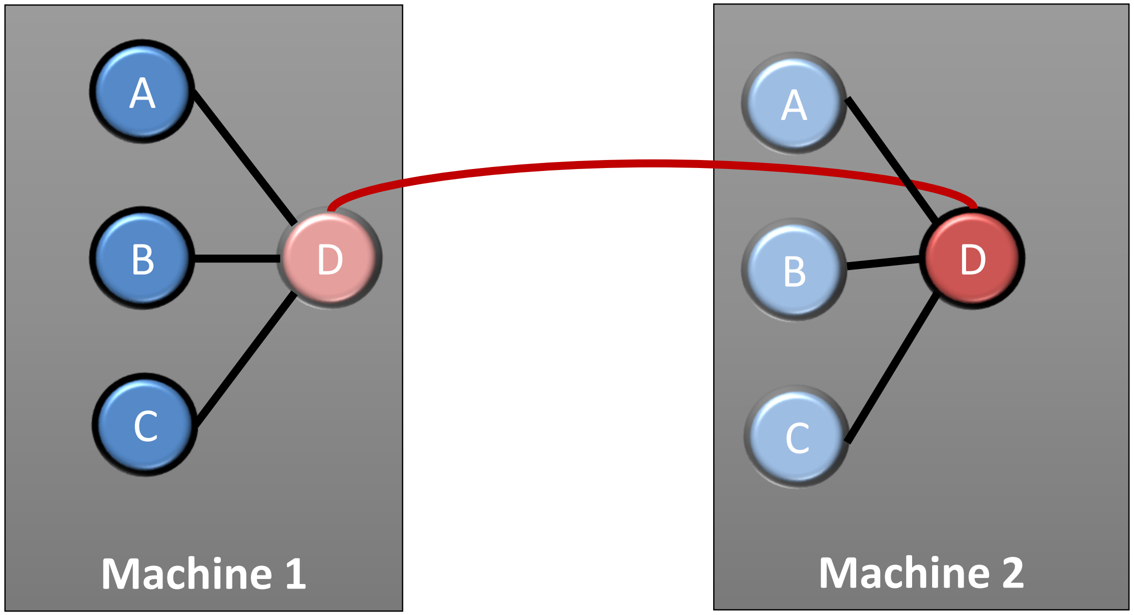


GraphLab Reduces Traffic by Creating Ghost Vertices



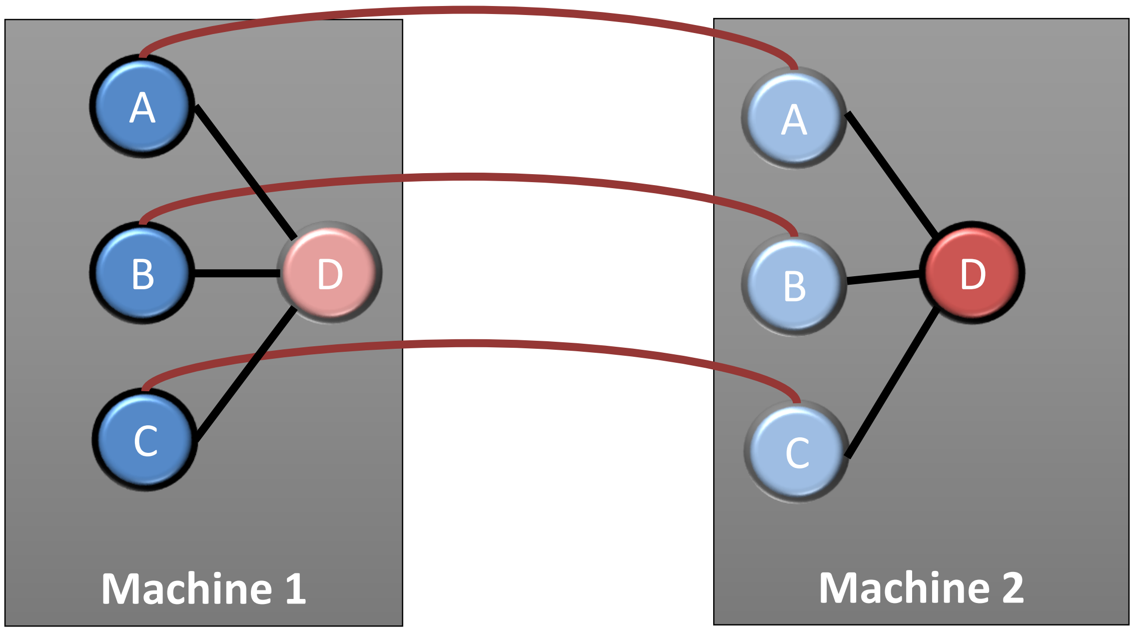
Create "Ghost Nodes" for the neighbors not on the same machine

GraphLab Reduces Broadcast Traffic by Creating Ghost Vertices



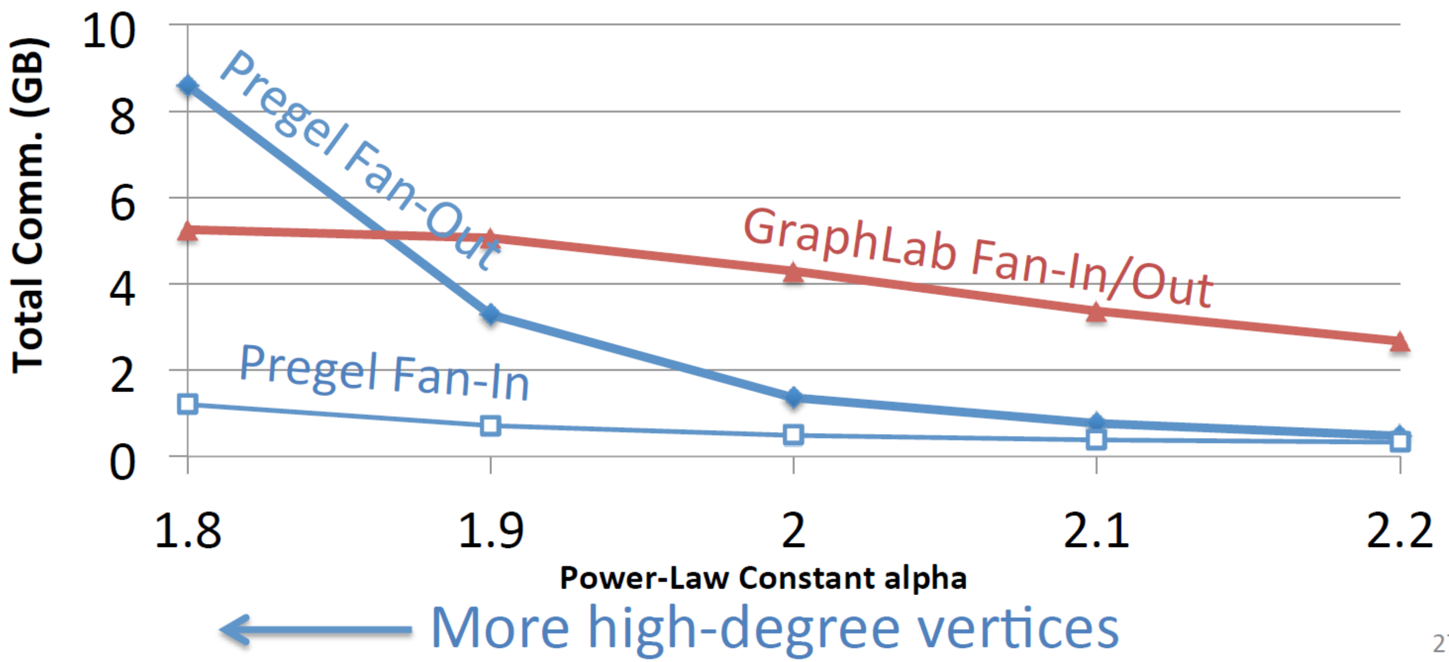
Updates to vertices under evaluation will be sent to another machine via 1 message, and the other machine internally performs transfers

GraphLab Suffers from Neighbors' Changes



Fan-In and Fan-Out Performance

- PageRank on synthetic Power-law Graphs
- GraphLab is undirected



Fan-In and Fan-Out Performance

- PageRank on synthetic Power-law Graphs
- GraphLab is undirected

Pregel and GraphLab are not well suited for natural graphs

- **Challenges to reduce both the fan-in and fan-out traffic for high-degree vertices**
- **Low quality graph partitioning cuts a significant number of edges in the graph (contributing to the significant traffic between different machines)**

1.8 1.9 2 2.1 2.2

Power-Law Constant alpha

← More high-degree vertices

PowerGraph – GAS Decomposition

Gather (Reduce)	**A**pply	**S**catter
Accumulate information about neighborhood	Apply the accumulated value to center vertex	Update adjacent edges and vertices.

GraphLab_PageRank(i)

```
// Compute sum over neighbors
total = 0
foreach( j in in_neighbors(i)):
  total = total + R[j] * w_ji
```

Gather Information About Neighborhood

```
// Update the PageRank
R[i] = 0.1 + total
```

Update Vertex

```
// Trigger neighbors to run again
if R[i] not converged then
  foreach( j in out_neighbors(i))
    signal vertex-program on j
```

Signal Neighbors & Modify Edge Data

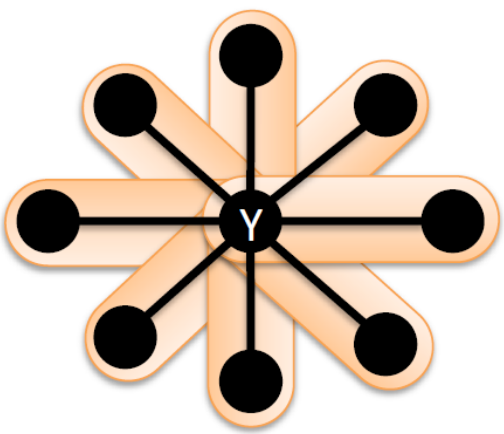
PowerGraph – GAS Decomposition

Gather (Reduce)

Accumulate information about neighborhood

User Defined:

- **Gather**( ) $\rightarrow \Sigma$
- $\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3$



Apply

Apply the accumulated value to center vertex

Scatter

Update adjacent edges and vertices.

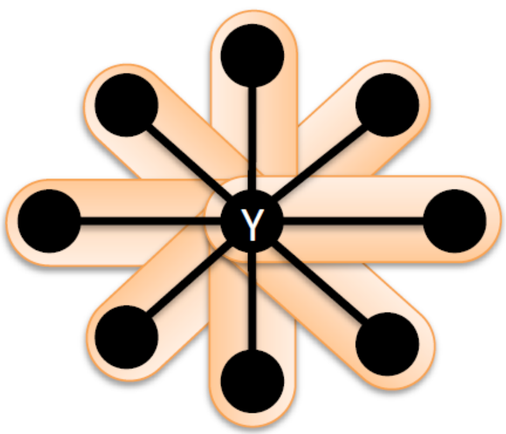
PowerGraph – GAS Decomposition

Gather (Reduce)

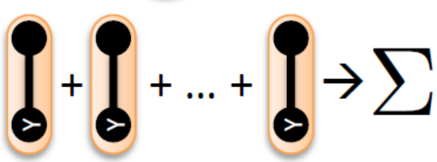
Accumulate information about neighborhood

User Defined:

- **Gather**(Y—●) → Σ
- $\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3$



Parallel Sum

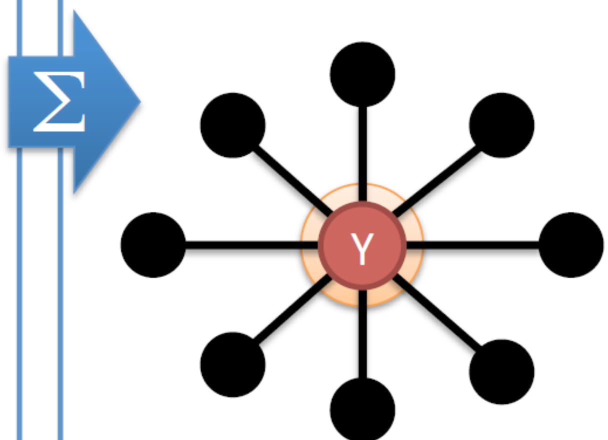


Apply

Apply the accumulated value to center vertex

User Defined:

- **Apply**(Y , Σ) → Y’



Scatter

Update adjacent edges and vertices.

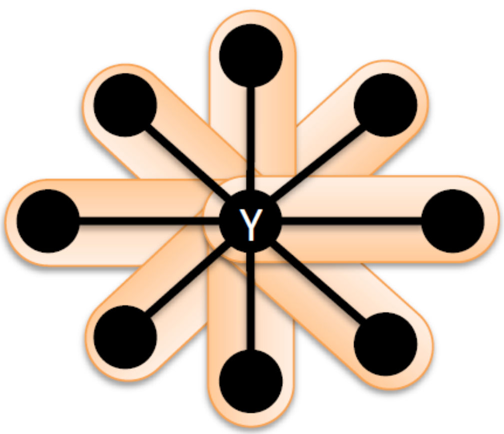
PowerGraph – GAS Decomposition

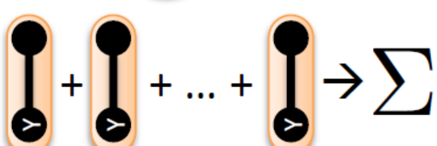
Gather (Reduce)

Accumulate information about neighborhood

User Defined:

- **Gather**(Y) $\rightarrow \Sigma$
- $\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3$



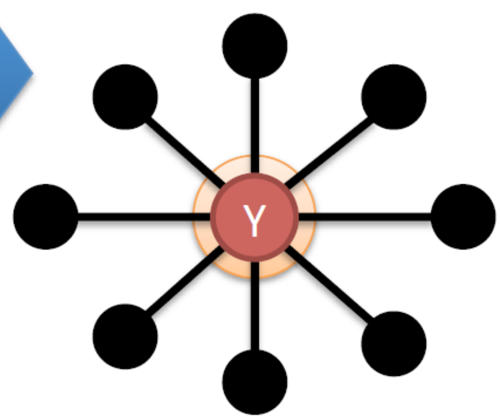


Apply

Apply the accumulated value to center vertex

User Defined:

- **Apply**(Y, Σ) $\rightarrow$ Y'

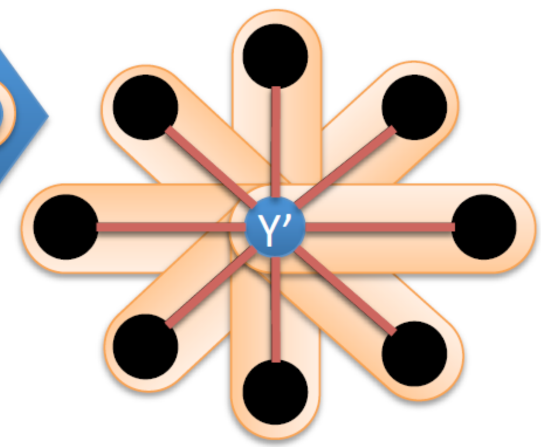


Scatter

Update adjacent edges and vertices.

User Defined:

- **Scatter**(Y') $\rightarrow$



Update Edge Data & Activate Neighbors

PageRank in PowerGraph

$$R[i] = 0.15 + \boxed{\sum_{j \in \text{Nbrs}(i)} w_{ji} R[j]}$$

PowerGraph_PageRank(i)

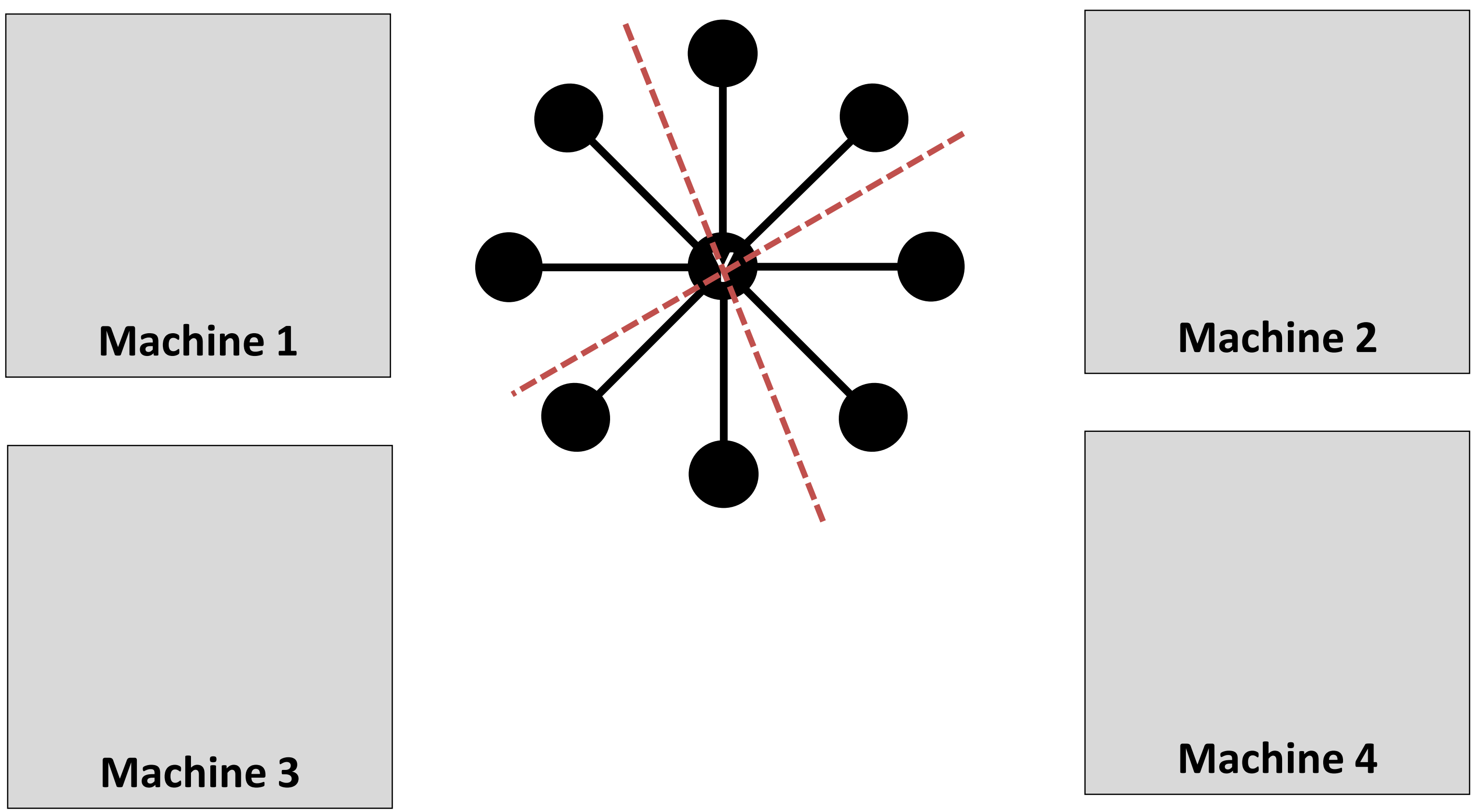
Gather(j → i) : return $w_{ji} * R[j]$
sum(a, b) : return a + b;

Apply(i, Σ) : $R[i] = 0.15 + \Sigma$

Scatter(i → j) :
if *R[i]* changed then trigger *j* to be **recomputed**

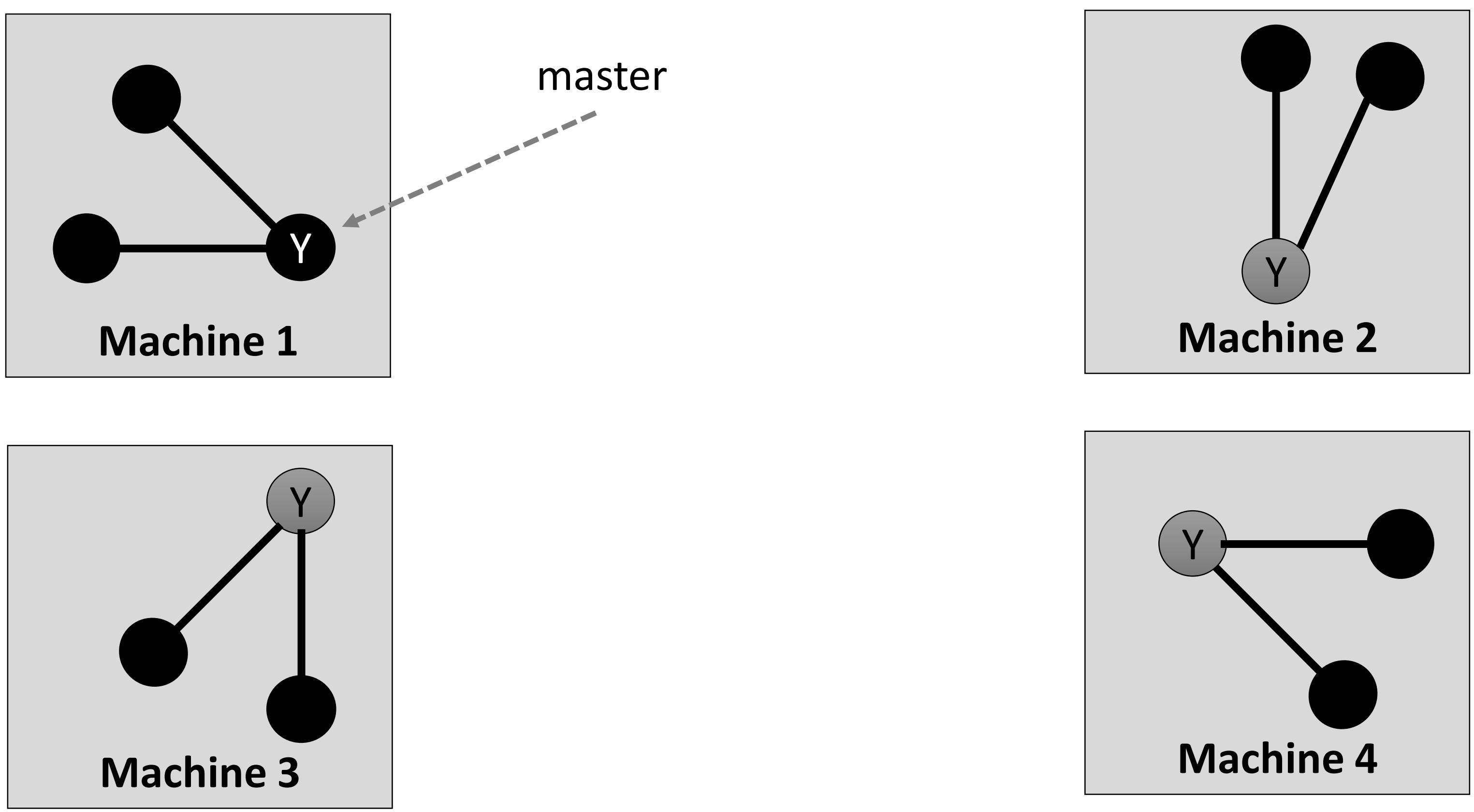
Distributed Execution of a PowerGrpah Vertex-Program

Cutting graphs from vertices instead of cutting from edges



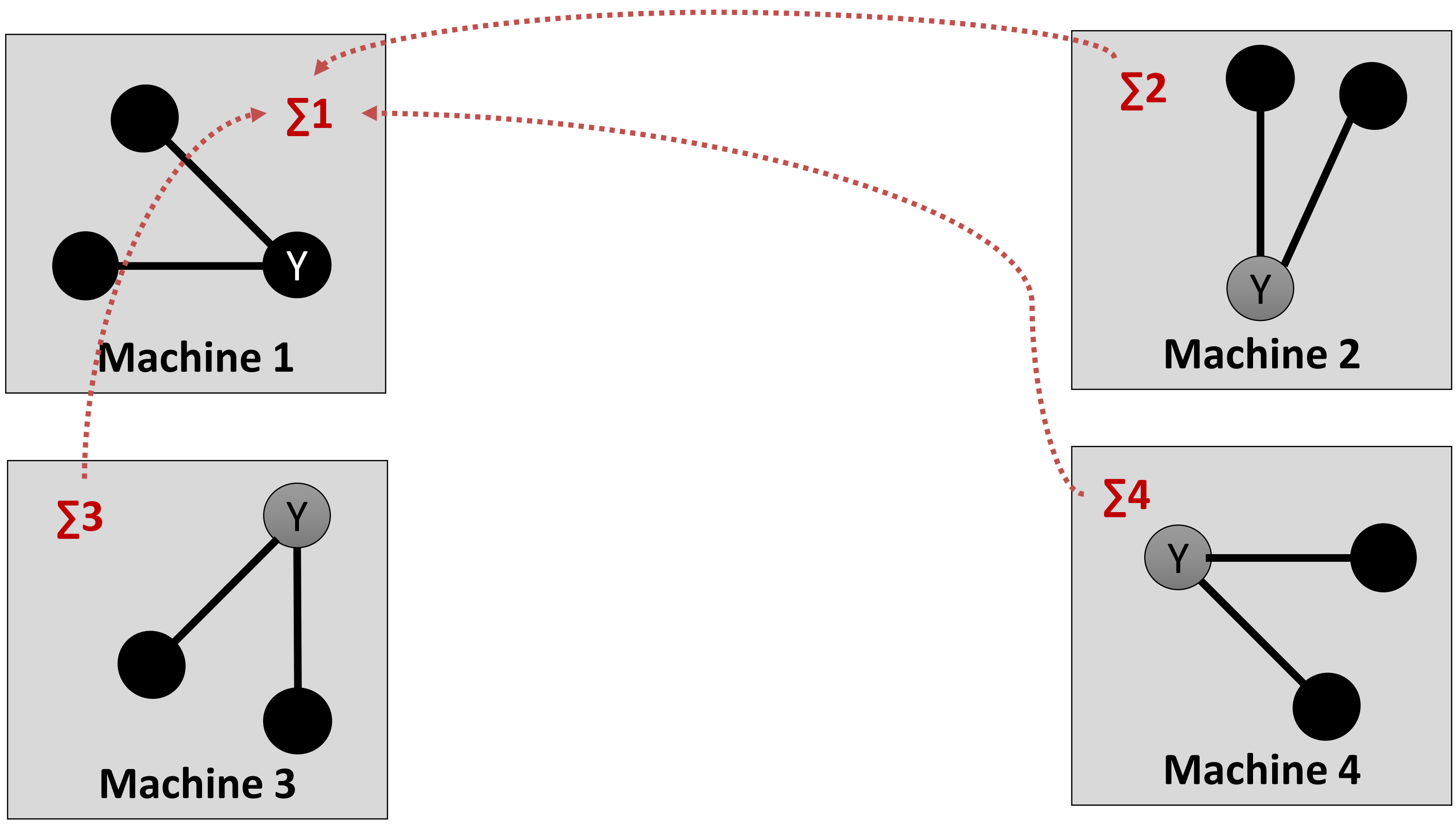
Distributed Execution of a PowerGrpah Vertex-Program

- Assign each portion of edges to a different machine
  - Select a master machine
  - Create shadow vertices on auxiliary machines



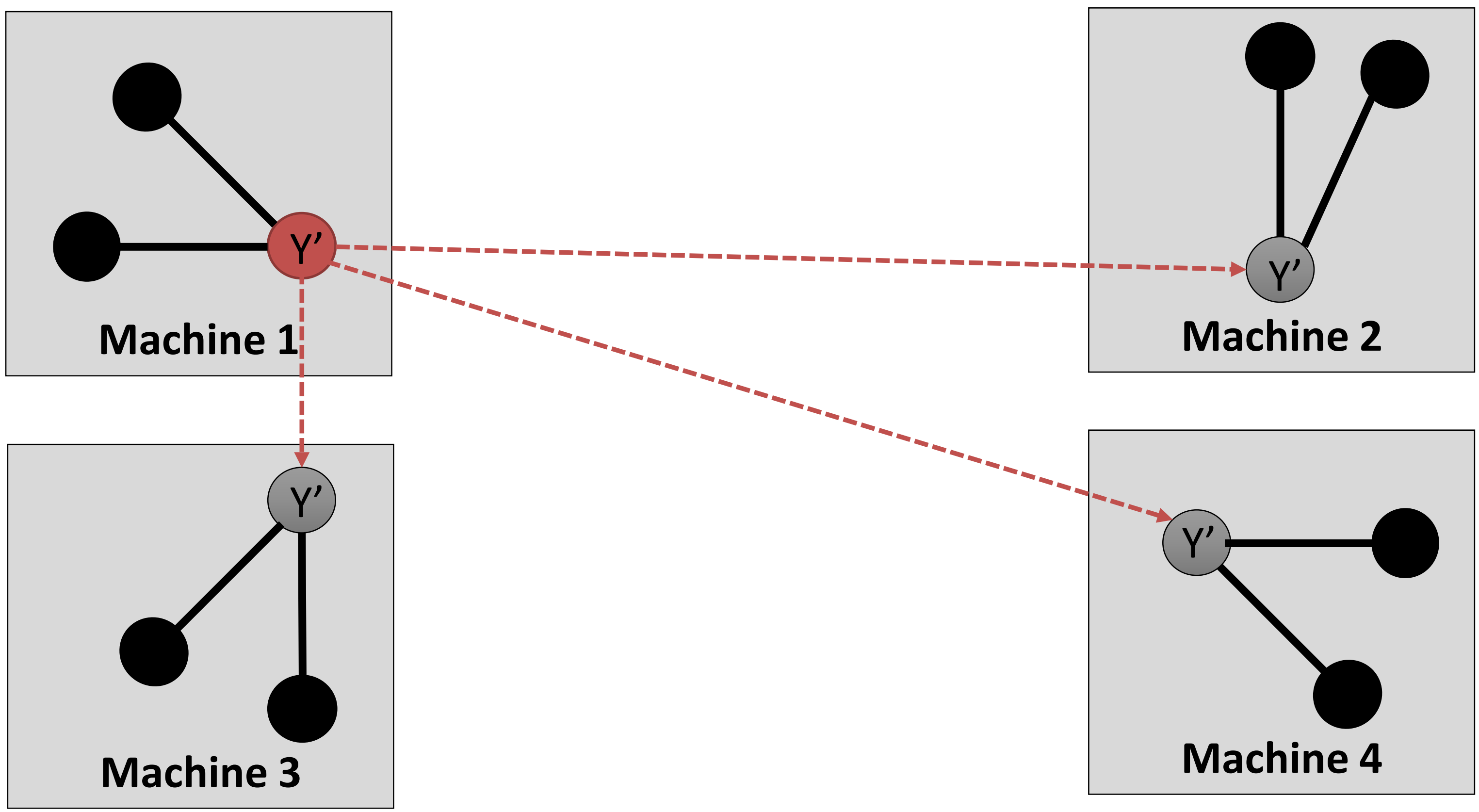
Distributed Execution of a PowerGrpah Vertex-Program

- Gather:
  - Each vertices shadow gathers on local machine (parallel)
  - Send the sum to the master machine



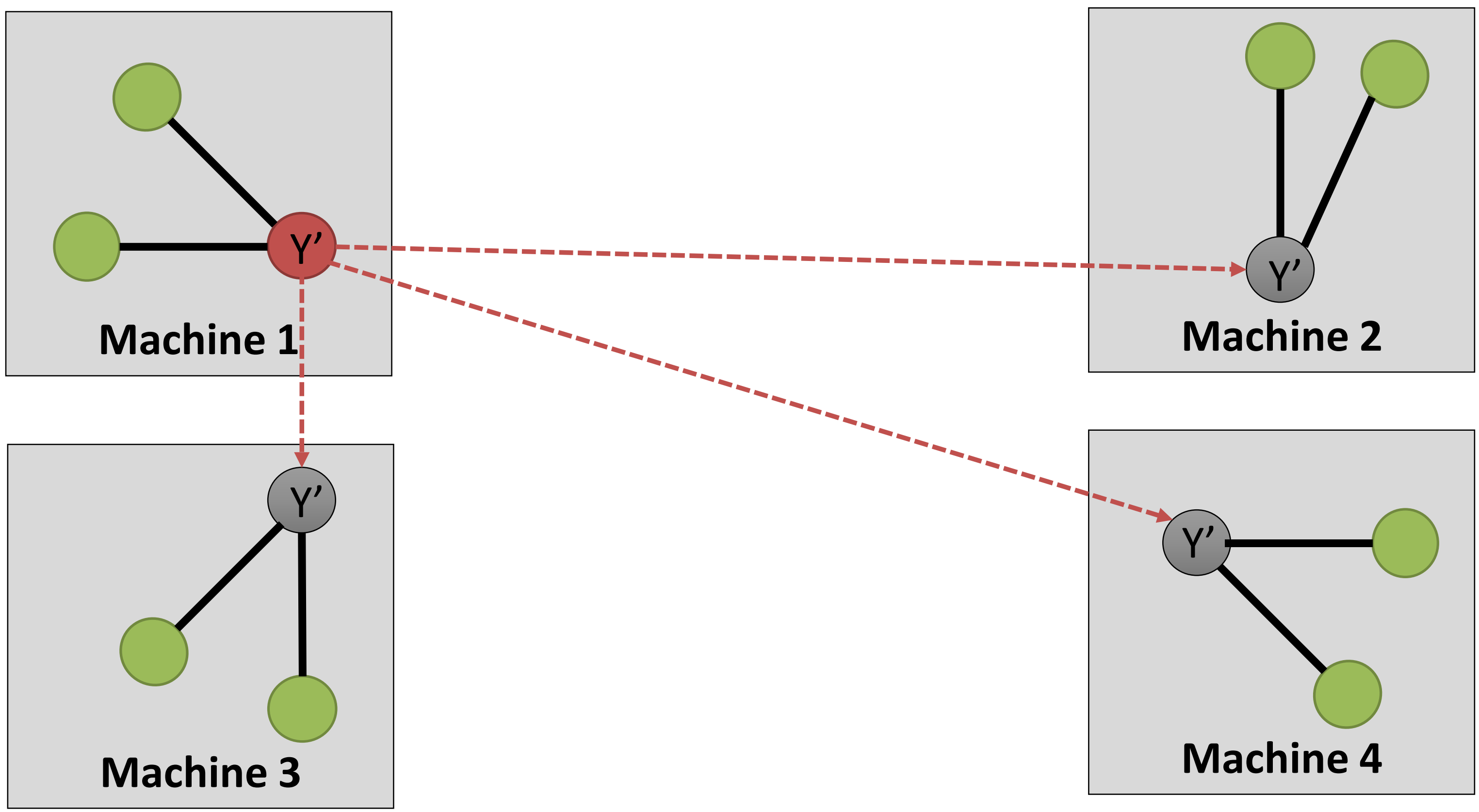
Distributed Execution of a PowerGrpah Vertex-Program

- Apply:
  - Apply the aggregated sum in a user defined way
  - Send the updated value to all machines



Distributed Execution of a PowerGrpah Vertex-Program

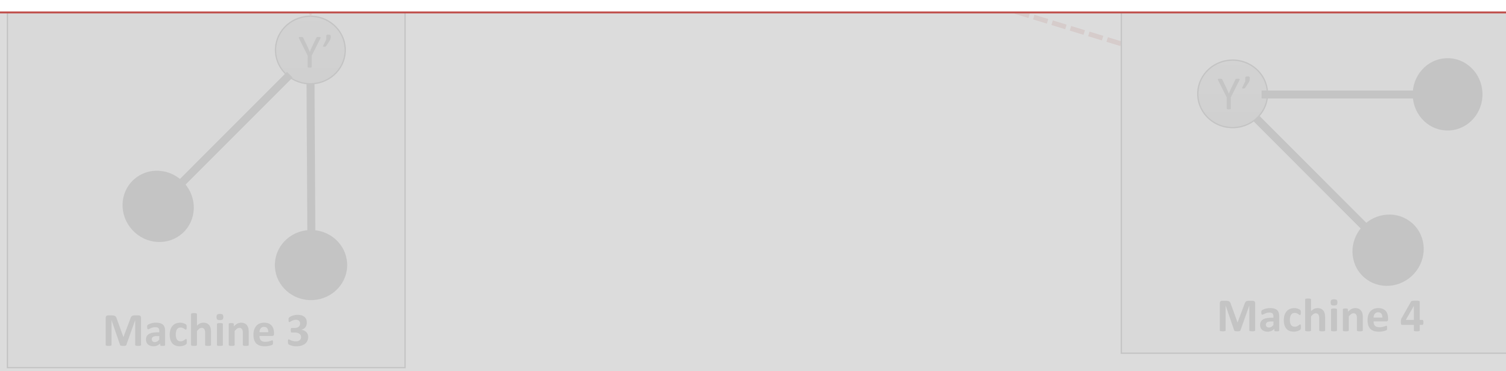
- Scatter:
  - Scatter locally (parallel)



Distributed Execution of a PowerGrpah Vertex-Program

- Scatter:
  - Scatter locally (parallel)

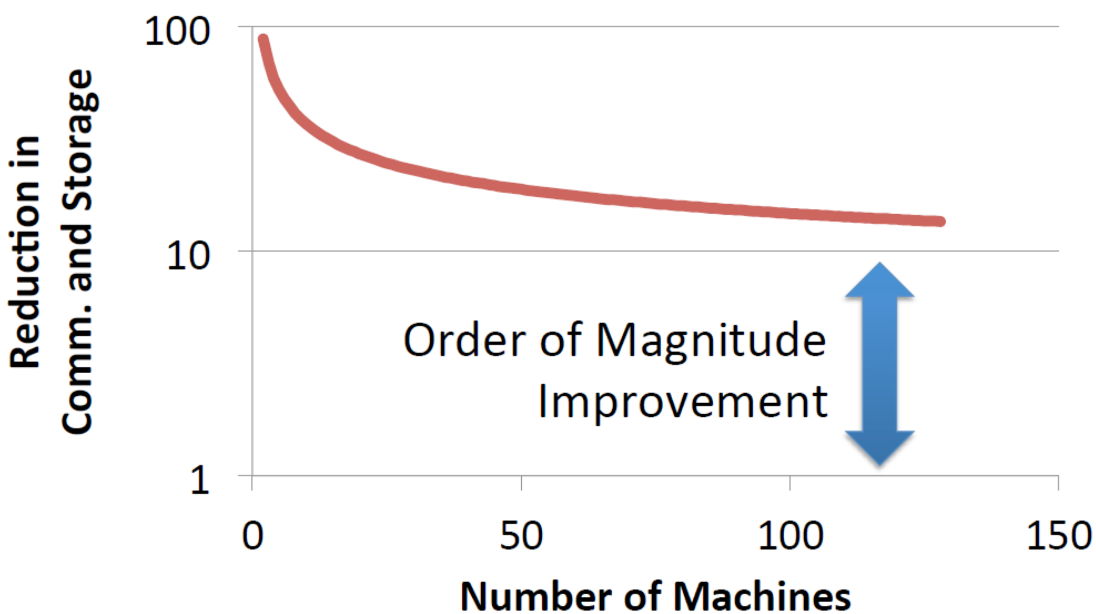
- **Communication is linear in the number of machines each vertex spans**
- **Percolation theory suggests that power law graphs have good vertex cuts**
- **<u>Theorem</u>: For any edge-cut we can directly construct a vertex-cut which requires less communication and storage**



How to perform vertex cuts?

- Random partitioning
  - Pick the lightest loaded machine when edges come in
  - No coordination overhead

Random vertex cut communication improvements

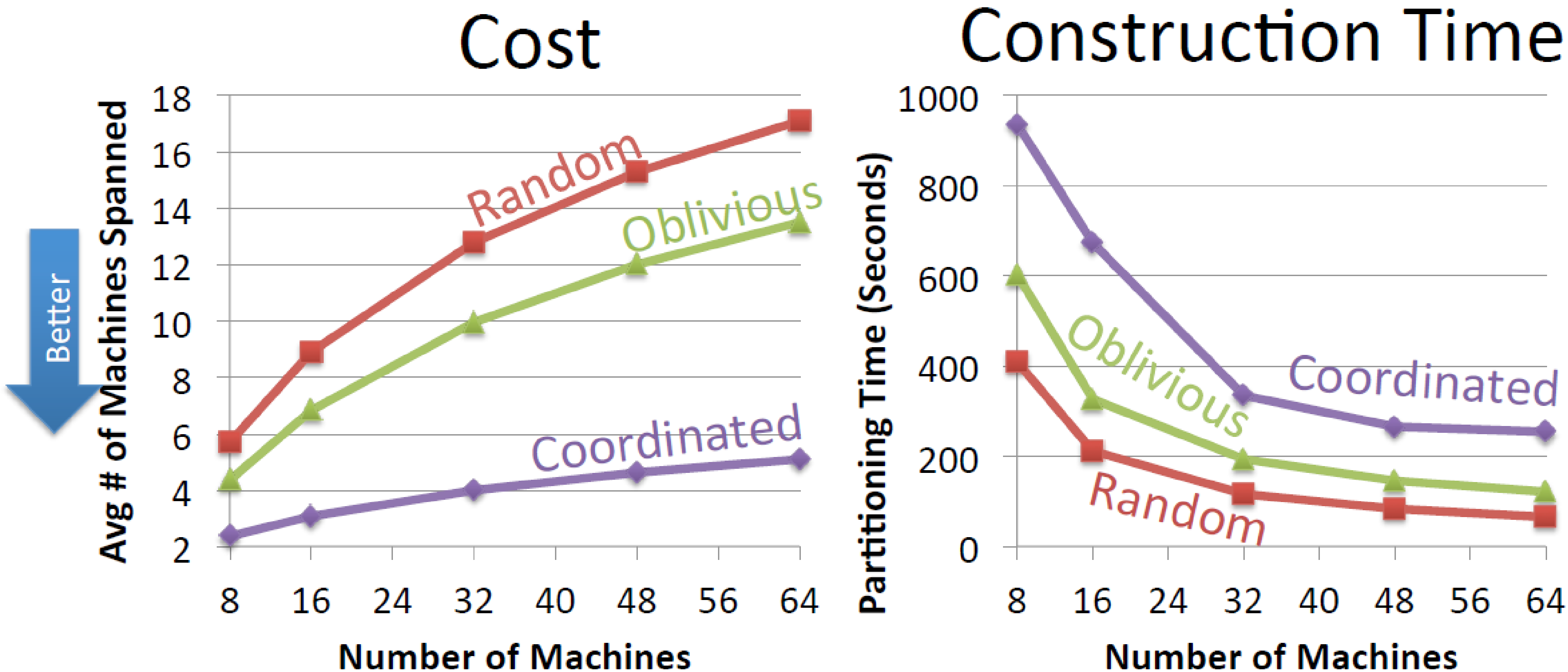


How to perform vertex cuts?

- Random partitioning
  - Pick the lightest loaded machine when edges come in
  - No coordination overhead
- Greedy partitioning
  - Globally tracks which vertex is placed to which machine and try to place the edges for the same vertex on the same machine in a workload-balanced way
  - High coordination overhead
- Oblivious partitioning
  - Locally tracks the per-vertex info, and place the edges in a workload-balanced way
  - Medium coordinate overhead

Comparing Vertex Cut Algorithms

Twitter Graph: 41M vertices, 1.4B edges



Oblivious balances cost and partitioning time.

Delta-Caching Optimization

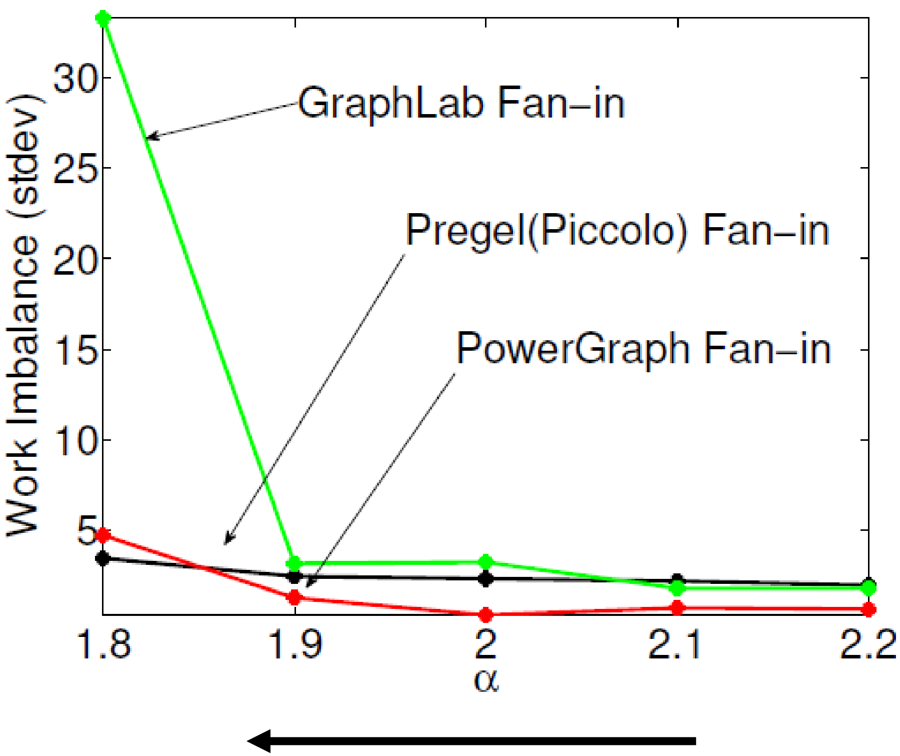
- Most of time, only a few of the neighboring vertices change their values
- Oppututnities to reduces the necessary gathering
- Keep a local copy of the gathered neighboring value from the last iteration
- Calculate delta during scatter to update the local cached value as well

Results – Algorithm Implementations

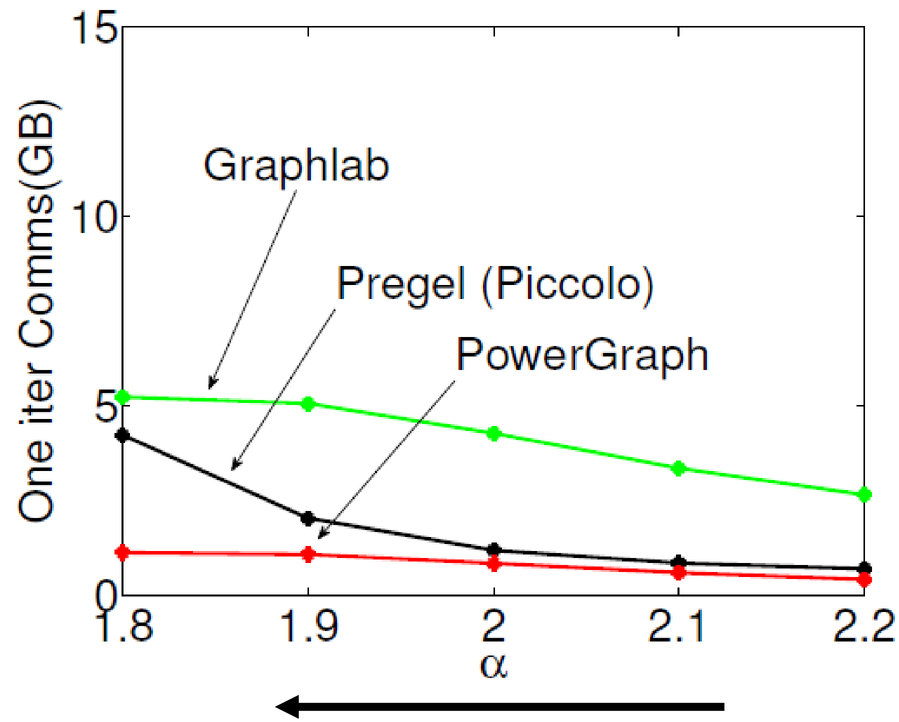
- **Collaborative Filtering**
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - SVD
  - Non-negative MF
- **Statistical Inference**
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- **Graph Analytics**
  - PageRank
  - Triangle Counting
  - Shortest Path
  - Graph Coloring
  - K-core Decomposition
- **Computer Vision**
  - Image stitching
- **Language Modeling**
  - LDA

Results – Compare to GraphLab & Pregel

- Running PageRank on Synthetic Power-Law Graphs



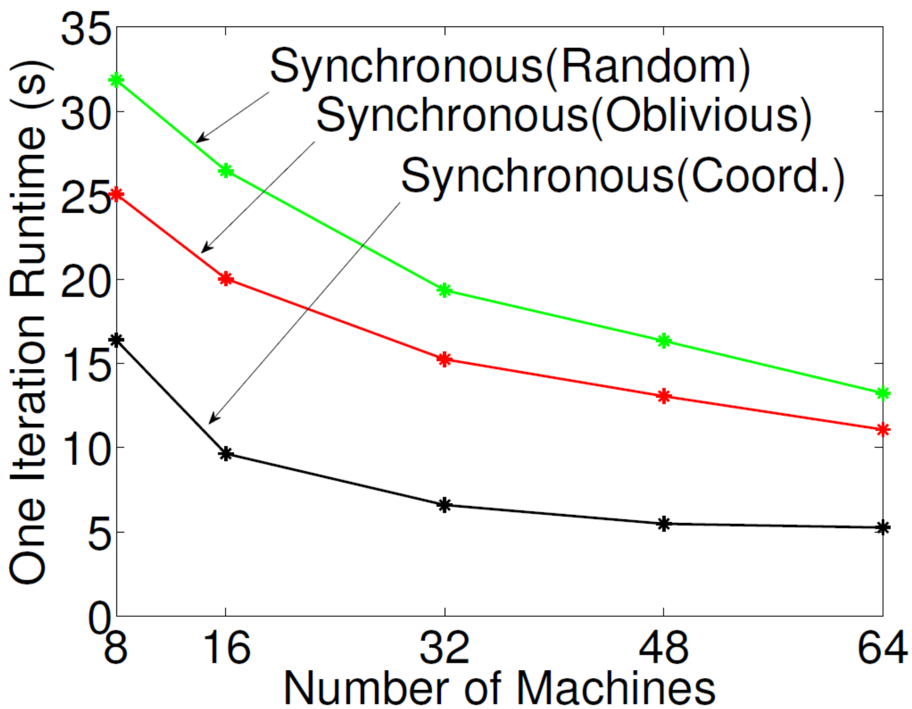
High-Degree Vertices

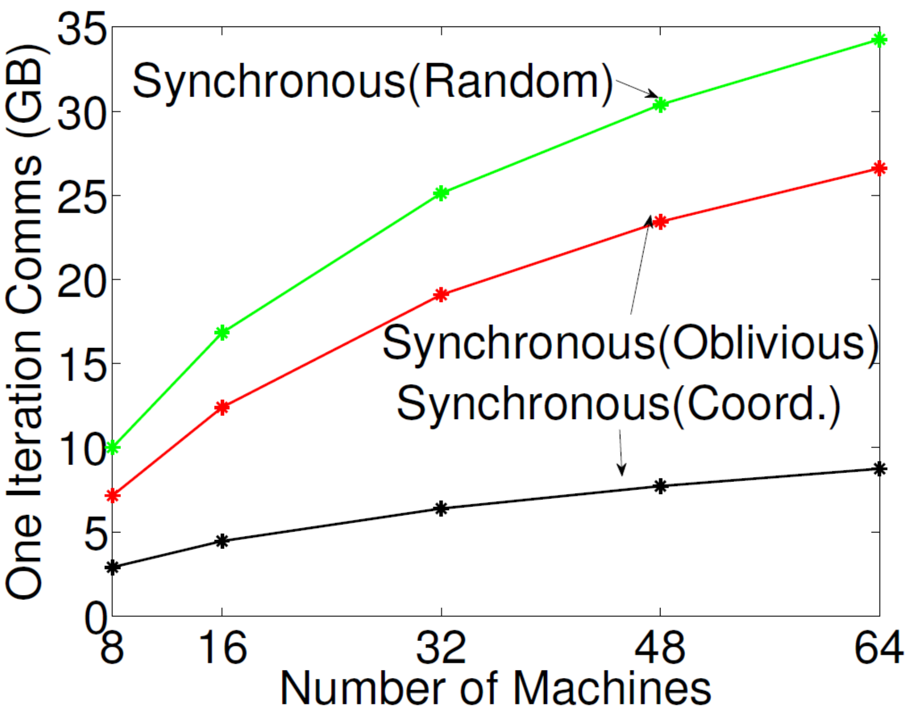


High-Degree Vertices

Results – Scaling

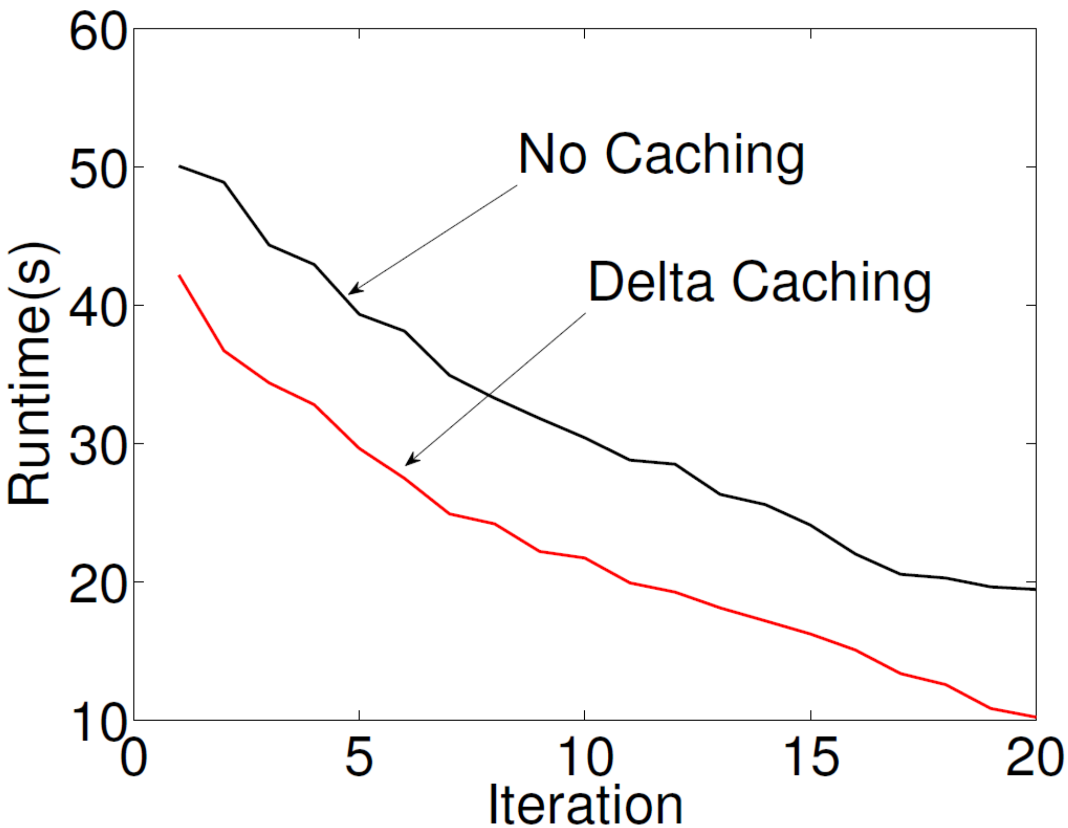
- Running PageRank on Twitter graph





Results – Delta Cache Improvements

- Running PageRank on Twitter graph



Strengths

+ Paper is well-motivated by the concern of efficiently processing power-law natural graphs

+ Paper clearly presents the challenges of the problems and the issues of the existing work

+ Paper shows a comprehensive study of the performance of the proposed abstraction

  - application algorithms
  - communication overhead
  - scaling

Weakness

- Paper does not show how well the abstraction performs if the application workload is not as power-law in nature. A good abstraction should still have reasonable performance even if the workload is not the target workload
- Paper did not show results with fewer than 8 machines and do not compare against sequential algorithm

Discussions

- Is the GAS abstraction general enough to represent all commonly know algorithms?
- Can we apply the vertex cut ideas to other framework for performance improvements?
- How will PowerGraph perform if the application workloads are not natural graphs?