PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

David Johnston

Department of Computer Science
Iowa State University
dwtj@iastate.edu

COMS 641: Data Intensive Languages and Systems - Design and Semantics

Overview

- The Problem: Power Law Graphs are Common
  - An imporant class of natural graphs.
  - A few *very high-degree vertices*.
  - Hard to partition.
  - These vertices cause performance and scalability challenges for existing graph-parallel systems.

Overview

- The Approach: The PowerGraph Abstraction
  - **GAS**: A new 3-phase vertex-program methodology.
  - “Think like a vertex.” (Malewicz et al., 2010, SIGMOD ’10)
  - Graph partitioning via vertex-cut, *not* edge-cut (3 variants).
  - Three execution modes (varying guarantees).
- Evaluation:
  - Evaluate three V-cut graph partitioning methods.
  - Evaluate three execution modes.
  - Compare with other MLDM systems.

Overview

- Benefits:
  - Vertex-oriented graph programming (“Think Like A Vertex”).
  - Handles Large Power Law Graphs.
  - Handles Very High-Degree Vertices.
  - Scalable.
  - Distributable/Parallelizable.
  - Fault Tolerant.

Some Context. . .

- Gonzalez et al., 2012, OSDI '12 Paper
- Gonzalez, 2012a, OSDI '12 Video
- Work connected with larger GraphLab project.
- Work primarily done at CMU, but also UW.
- Tech commercialized by Dato, Inc. (GraphLab, Inc.)
- Current open sourced version of tech: SGraph

The Problem: Power Law Graphs are Common

- An imporant class of natural graphs.
- A few *very high-degree vertices*.
- Hard to partition.
- These vertices cause performance and scalability challenges for existing graph-parallel systems.

Q: What's a Power Law Graph?
A: A graph whose vertex degree distribution is a power law distribution.

Power Law Functions Have A Characteristic Shape

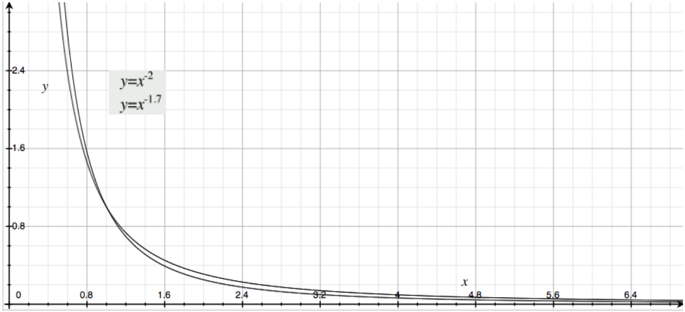
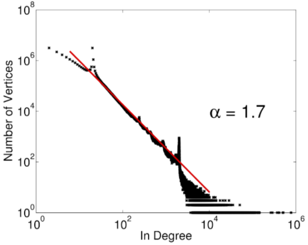
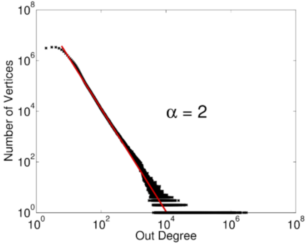


Figure: Two power law functions ($\alpha = 1.7$ and $\alpha = 2$) on a Cartesian coordinate system. A *power law* is a proportionality relation between two values of the form $y \propto x^{-\alpha}$, where α is positive. A power law *distribution* is just a probability distribution of this form.

Many Natural Graphs Are Power Law Graphs





(a) Twitter In-Degree (b) Twitter Out-Degree

Figure 1: The in and out degree distributions of the Twitter follower network plotted in log-log scale.

Figure: Gonzalez et al., 2012, OSDI '12

A Small Number of Vertices are of Very High Degree

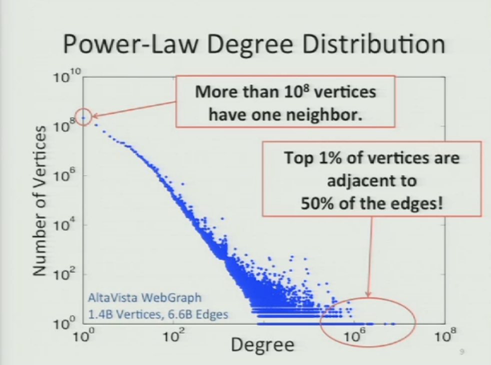
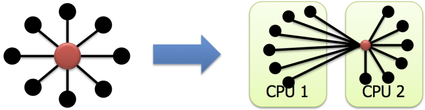


Figure: Gonzalez, 2012b, OSDI '12 Slides

Power Law Graphs Are Hard to Partition



- Power-Law graphs do not have **low-cost** balanced cuts *[Leskovec et al. 08, Lang 04]*
- Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs. *[Abou-Rjeili et al. 06]*

11

Figure: Gonzalez, 2012b, OSDI '12 Slides

Graph-Parallel Abstractions (A Review)

"A graph parallel abstraction consists of a *sparse* graph $G = \{V, E\}$ and a vertex-program Q which is executed in parallel on each vertex $v \in V$ and can interact... with neighboring instances."[1]

[1] Gonzalez et al., 2012, OSDI '12

Graph-Parallel Abstractions (A Review)

"In contrast to more general message passing models, graph-parallel abstractions constrain the interaction of vertex-program (sic) to a graph structure enabling the optimzation of data-layout and communication."[2]

[2] Gonzalez et al., 2012, OSDI '12

Graph-Parallel Abstractions Used For Comparison

User-defined vertex programs run in parallel on many nodes:

- **Pregel:**[3]
    - Programs communicate via message passing along graph.
    - Programs can change graph topology.
    - Vertices own their state and state of their outgoing edges.
    - Consistency via supersteps using a master node.
- **GraphLab:**[4]
    - Programs read/write shared data on a distributed graph.
    - Graph topology is fixed.
    - Good concurrency via smart scheduling of programs.
    - Serializability via locking and inter-node messages.

[3] Malewicz et al., 2010, SIGMOD '10
[4] Low et al., 2012, VLDB '12

Analysis of Previous Vertex Program Abstractions

- The authors analyze—under the assumption of power law graphs–both Pregel and GraphLab.
- They describe some problems (e.g. work imbalance and communication overhead) as a consequence of edge-cuts.
- Only a part of this analysis is discussed here.

Pregel Message Combiners Effective on Fan-In

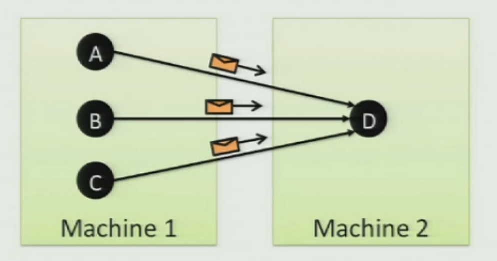


Figure: Gonzalez, 2012b, OSDI '12 Slides

Pregel Message Combiners Effective on Fan-In

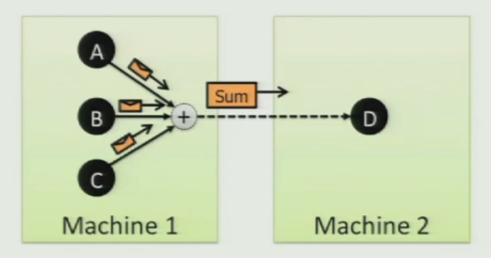
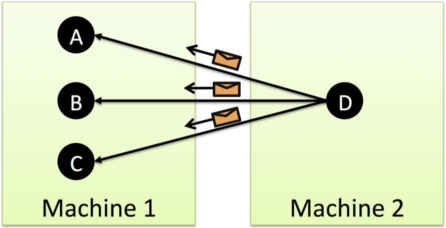
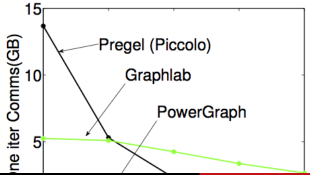


Figure: Gonzalez, 2012b, OSDI '12 Slides

Pregel Struggles with Fan-Out: Comms Overhead

Combiners not applicable on fan-out.





Pregel Struggles with Fan-Out: Work Imbalance

Pregel vertex program execution linear out-edge degree.

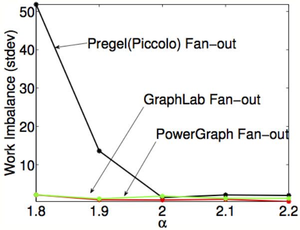


Figure: Gonzalez, 2012b, OSDI '12 Slides; Gonzalez et al., 2012, OSDI '12

Challenges of **High-Degree** Vertices

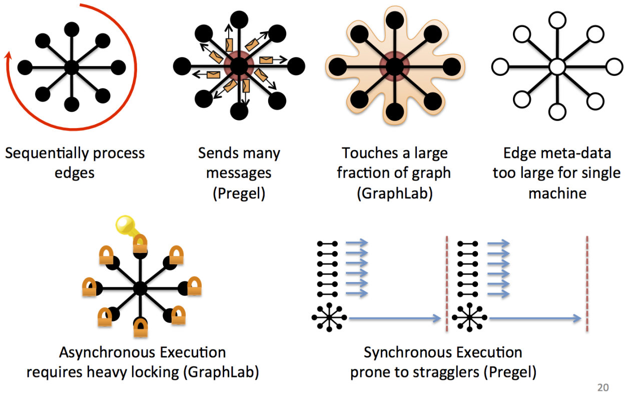


Figure: Gonzalez, 2012b, OSDI '12

Challenges

The authors identify five ways in which these properties of power law graphs create challenges for optimizations within pre-existing graph parallel abstractions:

- Work Balance
- Partitioning
- Communication
- Storage
- Computation

One Important Limitation

Unlike Pregel, graph topology is immutable.

The GAS Decomposition

The authors observed this pattern across many vertex programs.

- **Gather** an accumulated results from neighborhood.
- **Apply** the gathered result on the center node.
- **Scatter** accumulated information across the neighborhood.

Node Partitioning

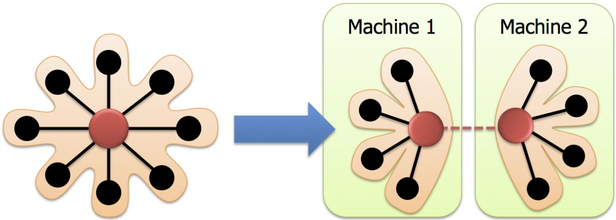


Figure: Gonzalez, 2012b, OSDI '12 Slides

The Key to PowerGraph's Optimizations

Parallelize vertex programs as before, but also parallelize their sub-operations, scatter and gather. *(Smaller critical sections!)*

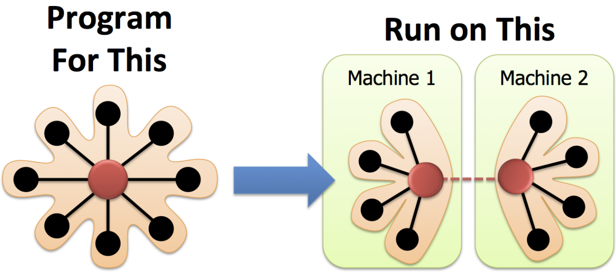


Figure: Gonzalez, 2012b, OSDI '12 Slides

A Taste of the Results

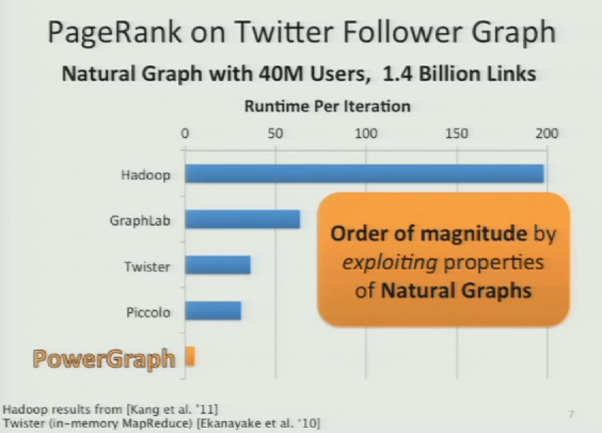


Figure: Gonzalez, 2012b, OSDI'12 Slides

The GAS Decomposition

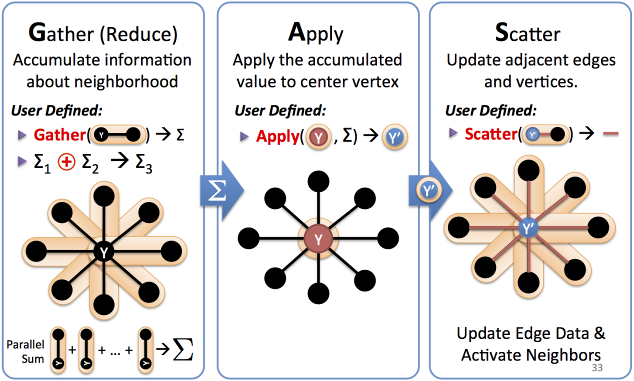


Figure: Gonzalez, 2012b, OSDI '12 Slides

The PowerGraph Abstraction

PowerGraph lifts the GAS decomposition into the framework.
The user implements each of these to make a vertex program.

interface *GASVertexProgram*(u) {

  // Run on gather_nbrs(u)

  gather$(D_u, D_{(u,v)}, D_v) \to Accum$

  sum$(Accum\ \mathrm{left}, Accum\ \mathrm{right}) \to Accum$

  apply$(D_u, Accum) \to D_u^{\mathrm{new}}$

  // Run on scatter_nbrs(u)

  scatter$(D_u^{\mathrm{new}}, D_{(u,v)}, D_v) \to (D_{(u,v)}^{\mathrm{new}}, Accum)$

}

Figure 2: All PowerGraph programs must implement the stateless gather, sum, apply, and scatter functions.

Figure: Gonzalez et al., 2012, OSDI '12

Delta Caching

- "[A] procedure which allows computation state to be dynamically maintained"[5]
- A programmer-directed optimization, useful in some programs.
- **Idea:** When scattering to neighbor, optionally add a correction onto cached copy of gather accumulator.
- Might better be called *cached gather accumulator with corrections.*

[5] Gonzalez et al., 2012, OSDI '12

Node Activation

- Except for initial activation (of all vertices), activation is always explict in user's vertex program.
- A vertex can only be activated by itself or by one of its neighbors.
- **Rule:** A vertex program can activate vertices in **gather()**, **apply()**, or **scatter()**, but only on *visible* vertices (i.e. vertices that are part of args).

Sequential Semantics

The meaning of a vertex program given the user-defined ops:

Algorithm 1: Vertex-Program Execution Semantics

Input: Center vertex u
if *cached accumulator* a_u *is empty* **then**
  foreach *neighbor* v *in gather_nbrs(u)* **do**
    $a_u \leftarrow \text{sum}(a_u, \text{gather}(D_u, D_{(u,v)}, D_v))$
  end
end
$D_u \leftarrow \text{apply}(D_u, a_u)$
foreach *neighbor* v *scatter_nbrs(u)* **do**
  $(D_{(u,v)}, \Delta a) \leftarrow \text{scatter}(D_u, D_{(u,v)}, D_v)$
  if a_v *and* Δa *are not Empty* **then** $a_v \leftarrow \text{sum}(a_v, \Delta a)$
  else $a_v \leftarrow \text{Empty}$
end

Figure: Gonzalez et al., 2012, OSDI ’12

Example: Greedy Graph Coloring

Greedy Graph Coloring

```
// gather_nbrs: ALL_NBRS
gather(D_u, D_(u,v), D_v):
   return set(D_v)
sum(a, b): return union(a, b)
apply(D_u, S):
   D_u = min c where c ∉ S
// scatter_nbrs: ALL_NBRS
scatter(D_u,D_(u,v),D_v):
   // Nbr changed since gather
   if(D_u == D_v)
      Activate(v)
   // Invalidate cached accum
   return NULL
```

Algorithm 1: Vertex-Program Execution Semantics

Input: Center vertex u
if *cached accumulator* a_u *is empty* **then**
  foreach *neighbor* v *in gather_nbrs(u)* **do**
    $a_u \leftarrow \text{sum}(a_u, \text{gather}(D_u, D_{(u,v)}, D_v))$
  end
end
$D_u \leftarrow \text{apply}(D_u, a_u)$
foreach *neighbor* v *scatter_nbrs(u)* **do**
  $(D_{(u,v)}, \Delta a) \leftarrow \text{scatter}(D_u, D_{(u,v)}, D_v)$
  if a_v *and* Δa *are not Empty* **then** $a_v \leftarrow \text{sum}(a_v, \Delta a)$
  else $a_v \leftarrow$ Empty
end

Figure: Gonzalez et al., 2012, OSDI '12

Edge-Cut vs Node-Cut Graph Partitioning

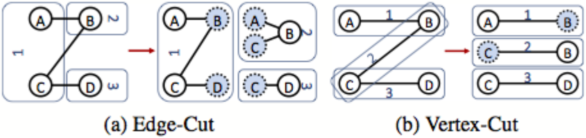


Figure 4: **(a)** An edge-cut and **(b)** vertex-cut of a graph into three parts. Shaded vertices are ghosts and mirrors respectively.

Figure: Gonzalez et al., 2012, OSDI '12

A Vertex Program on a Cut Vertex

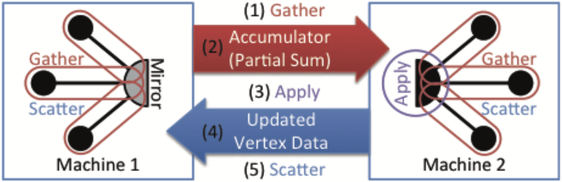


Figure 5: The communication pattern of the PowerGraph abstraction when using a vertex-cut. Gather function runs locally on each machine and then one accumulators is sent from each mirror to the master. The master runs the apply function and then sends the updated vertex data to all mirrors. Finally the scatter phase is run in parallel on mirrors.

Figure: Gonzalez et al., 2012, OSDI '12

Graph Partitioning Formalization

- **Observation:** Decreasing of the average number of replicas decreases *both* the storage overhead and communication overhead.
- **Formalization:** Optimization problem for graph partitioning, balanced $\boldsymbol{p}$-way vertex-cut
- **Formalization:** Parameterized probabilistic models of replication.

$\mathbb{E}$ Replication Varies with α and Number of Machines

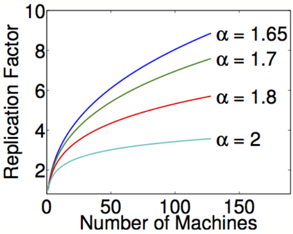


Figure: Gonzalez et al., 2012, OSDI '12

$\mathbb{E}$ Improvement of V-Cut over E-Cut Varies with α and Number of Machines

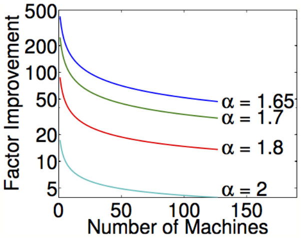


Figure: Gonzalez et al., 2012, OSDI '12

Graph Partitioning Algorithm: Random Heuristic

A simple algorithm to serve as baseline for improvements:

```
Randomly assign edges to nodes.
For each vertex v:
   Replicate v as needed.
   Assign one replica of v as master.
   Other replicas are ghosts.
```

This algorithm will probably produce the same replication rates as predicted by model.

Graph Partitioning Algorithm: Greedy Heuristic

Loosely speaking, place each edge on that node which minimizes the expected replication given previous edge/vertex assignments.

- Sequential
- Coordinated
- Oblivious

Three Variants of PowerGraph's Runtime

- **Bulk Synchronous:** Vertex program progress synchronized at both *minor-steps* and *super-steps*.
- **Asynchronous:** Mutations to graph are immediately visible by subsequent adjacent vertex programs.
- **Asynchronous Serializable:** . . .

Asynchronous Serializable

Def: A guarantee of *serializability* is a guarantee that every possible parallel/distributed execution of vertex programs has a corresponding sequential execution.[6]

[6] Gonzalez et al., 2012, OSDI '12

Asynchronous Serializable

Async+S is equivalent to GraphLab's *edge consistency*.[7]

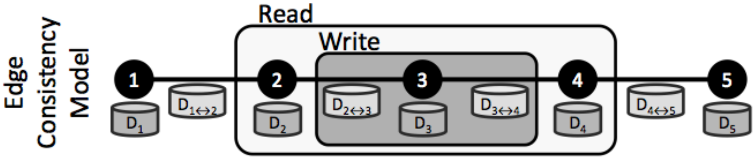


Figure: Low et al., 2012, VLDB '12

[7] Gonzalez et al., 2012, OSDI '12

Evaluation

- **Evaluated:** Three implemented graph partitioning algorithms.
  - Random
  - Oblivious
  - Coordinated
- **Evaluated:** Three implemented PowerGraph abstraction runtimes.
  - Bulk Synchronous
  - Asynchronous
  - Asynchronous Serializable
- **Not Evaluated:** Performance relative to other similar frameworks/languages.

Do Improved V-Cut Methods Reduce Replication on Big Graph Datasets?

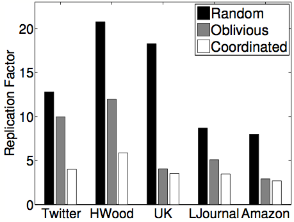


Figure: Gonzalez et al., 2012, OSDI '12

Do Improved V-Cut Methods Reduce Runtime on Big Graph Analysis Tasks?

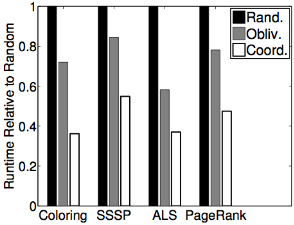


Figure: Gonzalez et al., 2012, OSDI '12

Do Improved V-Cut Methods Improve Scaling of Replication Rates? (Twitter Dataset)

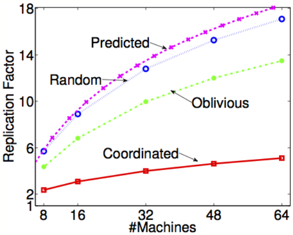


Figure: Gonzalez et al., 2012, OSDI '12

Do V-Cut Algorithms *Themselves* Scale Well? (Twitter Dataset)

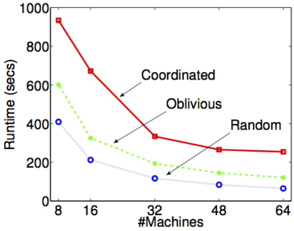


Figure: Gonzalez et al., 2012, OSDI '12

Do Improved V-Cut Methods Improve Scaling of User-Op Rates? (Twitter PageRank)

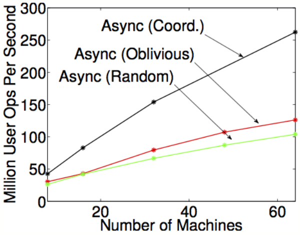


Figure: Gonzalez et al., 2012, OSDI ’12

Do Improved V-Cut Methods Improve Performance of Synchronous Execution?

Do Improvements Remain With Scaling?

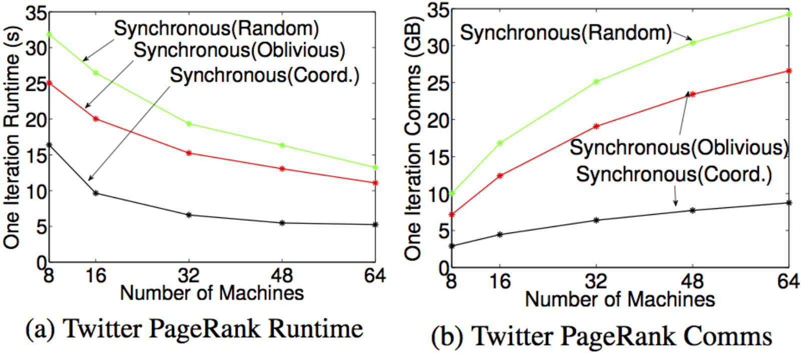
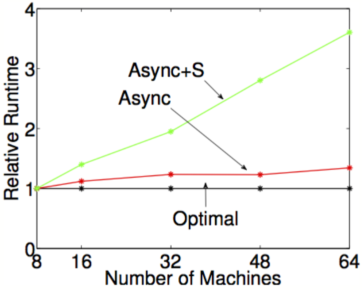


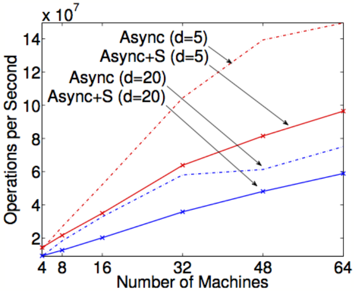
Figure: Gonzalez et al., 2012, OSDI '12

Does Serializability Guarantee Cause Weak Scaling?

Note: ALS Algorithm is $\boldsymbol{O(d^3)}$



(c) Coloring Weak Scaling



(a) ALS Throughput

Figure: Gonzalez et al., 2012, OSDI '12

PowerGraph Performance vs Other MLDM Systems

PageRank	Runtime	$\|V\|$	$\|E\|$	System
Hadoop [22]	198s	–	1.1B	50x8
Spark [37]	97.4s	40M	1.5B	50x2
Twister [15]	36s	50M	1.4B	64x4
PowerGraph (Sync)	3.6s	40M	1.5B	64x8

Triangle Count	Runtime	$\|V\|$	$\|E\|$	System
Hadoop [36]	423m	40M	1.4B	1636x?
PowerGraph (Sync)	1.5m	40M	1.4B	64x16

LDA	Tok/sec	Topics	System
Smola et al. [34]	150M	1000	100x8
PowerGraph (Async)	110M	1000	64x16

Figure: Gonzalez et al., 2012, OSDI '12

Overview

- The Problem: Power Law Graphs are Common
  - An imporant class of natural graphs.
  - A few *very high-degree vertices*.
  - Hard to partition.
  - These vertices cause performance and scalability challenges for existing graph-parallel systems.

Overview

- The Approach: The PowerGraph Abstraction
  - **GAS**: A new 3-phase vertex-program methodology.
  - “Think like a vertex.” (Malewicz et al., 2010, SIGMOD ’10)
  - Graph partitioning via vertex-cut, *not* edge-cut (3 variants).
  - Three execution modes (varying guarantees).
- Evaluation:
  - Evaluate three V-cut graph partitioning methods.
  - Evaluate three execution modes.
  - Compare with other MLDM systems.

Questions?

`http://www.cs.iastate.edu/~dwtj`

[Gon+12] Joseph E Gonzalez et al. "Powergraph: Distributed graph-parallel computation on natural graphs". In: *The 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 2012, pp. 17–30.

[Gon12a] Joseph E Gonzalez. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs*. 2012. URL: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/gonzalez.

[Gon12b] Joseph E Gonzalez. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs*. 2012. URL: https://www.usenix.org/sites/default/files/conference/protected-files/gonzalez_osdi12_slides.pdf.

[Low+12] Yucheng Low et al. "Distributed GraphLab: a framework for machine learning and data mining in the cloud". In: *Proceedings of the VLDB Endowment* 5.8 (2012), pp. 716–727.

[Mal+10] Grzegorz Malewicz et al. "Pregel: a system for large-scale graph processing". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 135–146.

Hidden Slide 1

Hidden Slide 2