PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Gonzalez et al.

Background

Large-scale graph computation: targeted advertising, NLP, etc

Datasets and models have grown beyond the limits of single machine computation

Graph-parallel abstraction:

- vertex-programs run in parallel
- interact with each other through edges

Pregel

Synchronous message-based abstraction

At each step, the vertex-program:

- receives messages from the previous step
- does some computation
- sends messages to neighbors

Advance to next step after all vertices are done

GraphLab

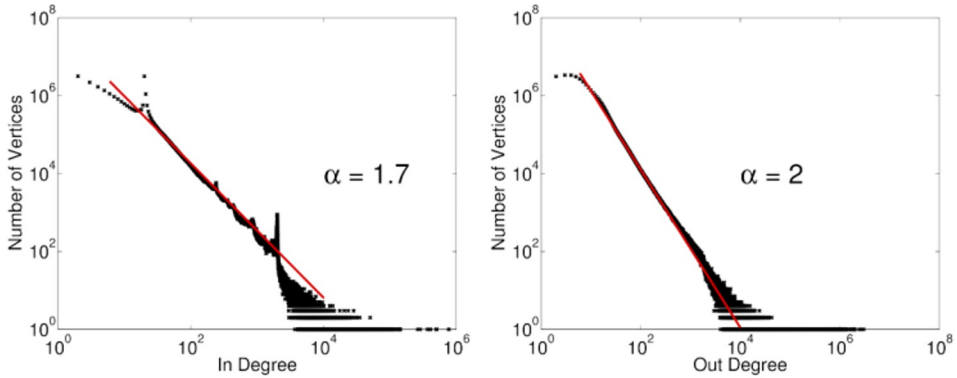
Asynchronous shared-memory abstraction

Each vertex-program can:

- read data from its own vertex, edges, and neighbors
- schedule neighbors to run

Real-word Graphs

Social media and web networks are “scale-free”, degree distribution follows power law (# vertices with degree *d* is proportional to $1/d^{\alpha}$)



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

Real-word Graphs

Scale-free graphs present challenges for existing graph-parallel abstractions:

- Work depends on degree, can vary widely across vertices
- Hard to partition
- Does not parallelize within vertex-programs

Generic Vertex-Program Model

GraphLab and Pregel have a similar overall structure

GAS model for graph computation:

- **Gather**: collect information about adjacent vertices and edges
- **Apply**: update value of central vertex
- **Scatter**: update the data on adjacent edges

Generic Vertex-Program Model

Formally:

$$\Sigma \leftarrow \bigoplus_{v \in \mathbf{Nbr}[u]} g\left(D_u, D_{(u,v)}, D_v\right)$$

$$D_u^{\text{new}} \leftarrow a\left(D_u, \Sigma\right)$$

$$\forall v \in \mathbf{Nbr}[u]: \quad \left(D_{(u,v)}\right) \leftarrow s\left(D_u^{\text{new}}, D_{(u,v)}, D_v\right)$$

PowerGraph

gather in parallel

sum is commutative and associative

scatter in parallel

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

Delta Caching

Vertex-program runs in response to a change in a few neighbors. Normally, we run gather on all neighbors, most of which are unchanged

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

Scatter can optionally return Δa, which is added to a_v

Clear a_v otherwise, recompute gather on the next execution of v

Delta Caching

If we can define an inverse of the accumulation function (e.g. subtraction):

$$\Delta a = g(D_u, D^{\text{new}}_{(u,v)}, D^{\text{new}}_v) - g(D_u, D_{(u,v)}, D_v)$$

We'll see an example using Δa later

PowerGraph Engine

To initialize, user *activates* a specific vertex or all vertices

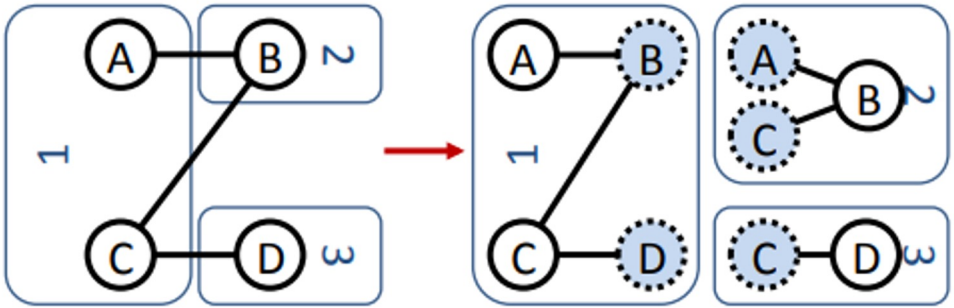
Engine maintains a set of active vertices

- Execute active vertex-programs
- After scatter phase, vertex becomes inactive
- Vertex can activate itself or neighboring vertices

Order of execution is up to engine

Distributed Graph Placement: Edge Cut

Place a graph on *p* machines: construct a *p*-way **edge-cut**

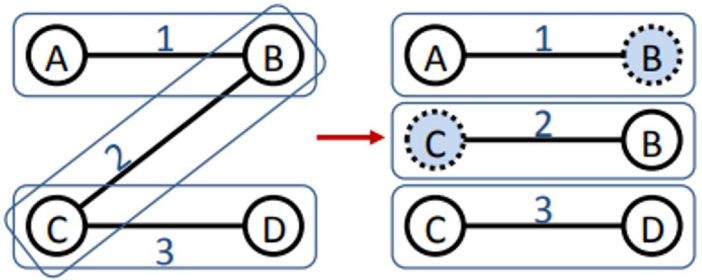


Overhead from every cut edge, and have to synchronize vertex and edge data across the cut

Intuitively: evenly assign vertices to machines, allow edges to span machines

Distributed Graph Placement: Vertex Cut

Evenly assign **edges** to machines, allow **vertices** to span machines



Only have to synchronize vertex data, so we want to minimize the number of machines each vertex spans

Distributed Graph Placement: Vertex Cut

Formally:

$$\min_{A} \frac{1}{|V|} \sum_{v \in V} |A(v)|$$

$$\text{s.t.} \qquad \max_{m} \left| \{ e \in E \mid A(e) = m \} \right|, < \lambda \frac{|E|}{p}$$

$\lambda \geq 1$ is a constant imbalance factor

Randomized Vertex Cut

On p machines, randomly assigning edges to machines has expected replication:

$$\mathbb{E}\left[\frac{1}{|V|}\sum_{v\in V}|A(v)|\right] = \frac{p}{|V|}\sum_{v\in V}\left(1-\left(1-\frac{1}{p}\right)^{\mathbf{D}[v]}\right)$$

Vertex Cut

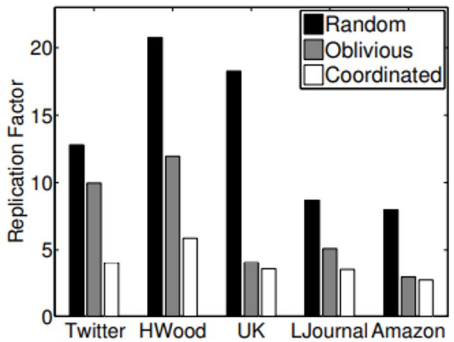
Let $A(x)$ be the set of machines that vertex x is assigned to. Greedy heuristic for edge (u, v):

1. If $A(u)$ and $A(v)$ intersect, assign to a machine in the intersection
2. If $A(u)$ and $A(v)$ nonempty but do not intersect, assign to a machine from the vertex with the most unassigned edges
3. If one of the vertices is assigned, pick a machine from the assigned vertex
4. If neither vertex is assigned, pick the least loaded machine

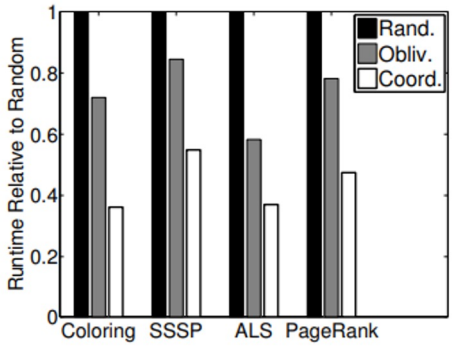
Vertex Cut

Greedy heuristic requires coordination between machines. Two approaches:

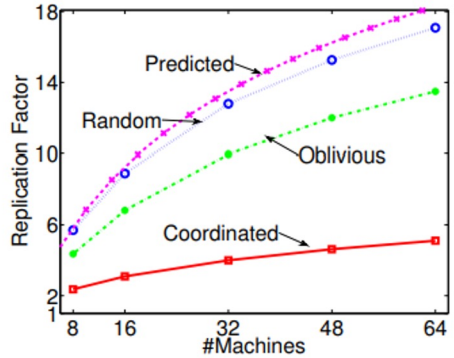
- **Coordinated**: maintain values of *A(x)* in distributed table, which is periodically updated
- **Oblivious**: each machine maintains its own estimate of *A(x)*



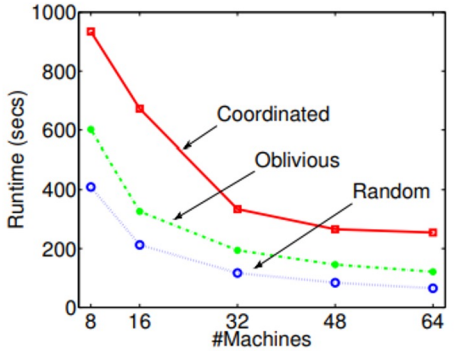
(a) Actual Replication



(b) Effect of Partitioning



(a) Replication Factor (Twitter)

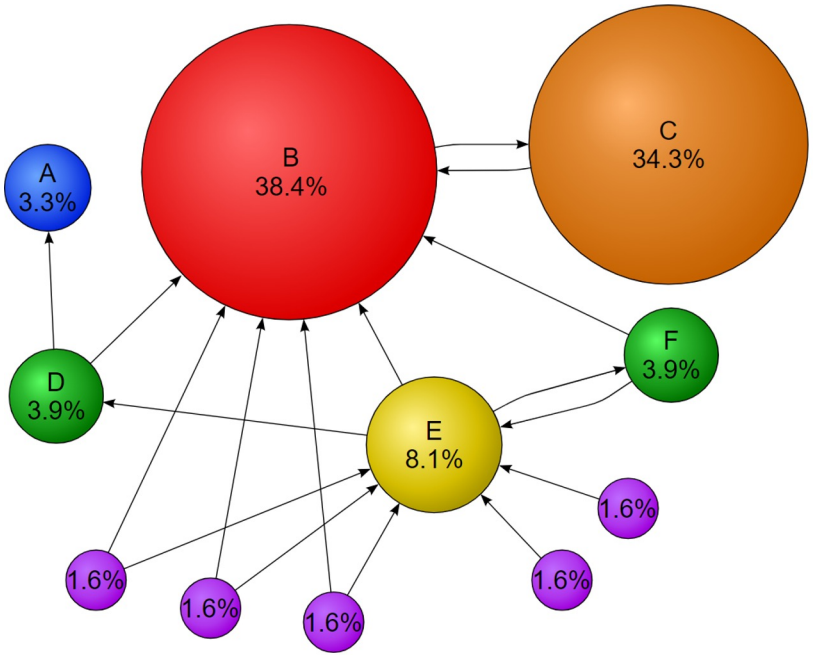


(b) Ingress time (Twitter)

Example: PageRank

Rank websites

- More important websites have more links from other websites
- The links from more important websites have more weight

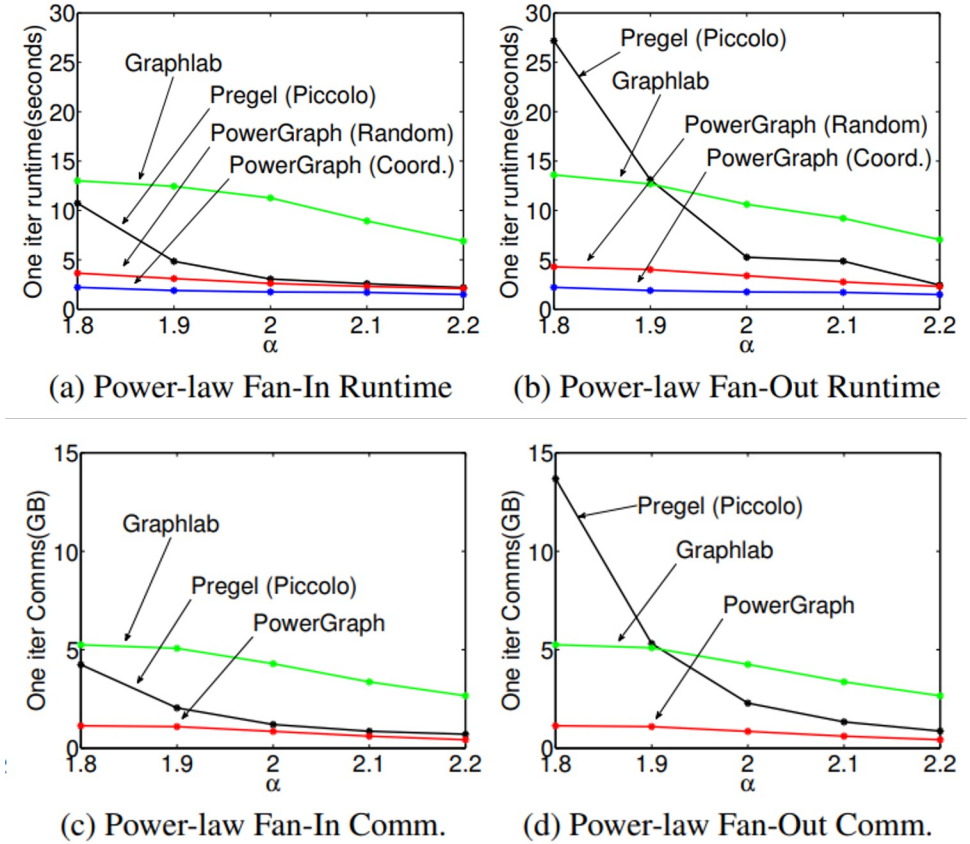


PageRank

The accumulation is addition, so we can use Δa

If node value changes enough, recalculate ranking for neighbors

```
// gather_nbrs: IN_NBRS
gather(D_u, D_(u,v), D_v):
  return D_v.rank / #outNbrs(v)
sum(a, b): return a + b
apply(D_u, acc):
  rnew = 0.15 + 0.85 * acc
  D_u.delta = (rnew - D_u.rank)/
              #outNbrs(u)
  D_u.rank = rnew
// scatter_nbrs: OUT_NBRS
scatter(D_u, D_(u,v), D_v):
  if(|D_u.delta|>ε) Activate(v)
  return delta
```



PowerGraph vs GraphLab and Pregel

Synchronous Execution (similar to Pregel)

Super-step consists of three **minor**-steps:

- gather on all active vertices
- apply on all active vertices
- scatter on all active vertices

Changes to vertex and edge data are committed after each minor-step

Newly-activated vertices are executed in the next super-step

Asynchronous Execution (similar to GraphLab)

Engine runs active vertices as resources become available

Changes made to vertex and edge data committed immediately

Pros:

- More effective usage of resources
- Algorithm can converge faster

Cons:

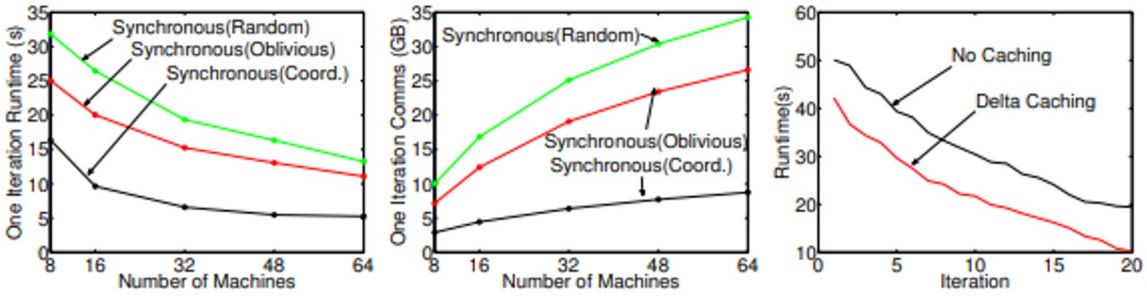
- Non-determinism

Asynchronous Execution (similar to GraphLab)

How to address non-determinism?

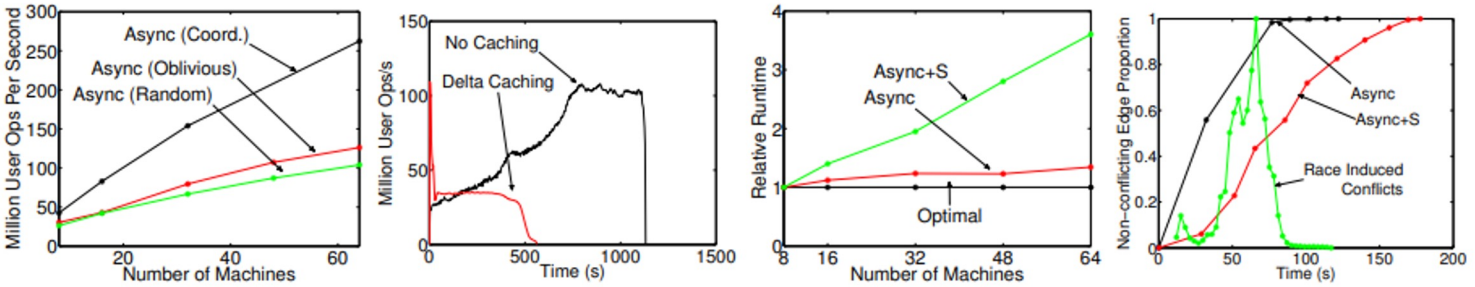
- **serializability**: every parallel execution has a corresponding sequential execution
- Prevent adjacent vertex-programs from running concurrently

PowerGraph has **Async+S**



(a) Twitter PageRank Runtime (b) Twitter PageRank Comms (c) Twitter PageRank Delta Cache

Synchronous Experiments with PowerGraph



(a) Twitter PageRank Throughput (b) Twitter PageRank Delta Cache (c) Coloring Weak Scaling (d) Coloring Conflict Rate

Asynchronous Experiments with PowerGraph

Fault Tolerance

Similar to Pregel and GraphLab, save snapshots of the graph

- **Synchronous** engine: snapshot between super-steps
- **Asynchronous** engine: suspend execution to construct snapshot

Overhead is small relative to total runtime

Related Work

Vertex cuts:

- CATALYUREK, U., AND AYKANAT, C. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication
- DEVINE, K. D., BOMAN, E. G., HEAPHY, R. T., BISSELING, R. H., AND CATALYUREK, U. V. Parallel hypergraph partitioning for scientific computing

Graph-parallel abstractions:

- GREGOR, D., AND LUMSDAINE, A. The parallel BGL: A generic library for distributed graph computations
- CHENG, R., HONG, J., KYROLA, A., MIAO, Y., WENG, X., WU, M., YANG, F., ZHOU, L., ZHAO, F., AND CHEN, E. Kineograph: taking the pulse of a fast-changing and connected world
- PUJOL, J. M., ERRAMILLI, V., SIGANOS, G., YANG, X., LAOUTARIS, N., CHHABRA, P., AND RODRIGUEZ, P. The littleengine(s) that could: scaling online social networks
- KYROLA, A., BLELLOCH, G., AND GUESTRIN, C. GraphChi: Large-scale graph computation on just a PC.