PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Fan Jun Bo

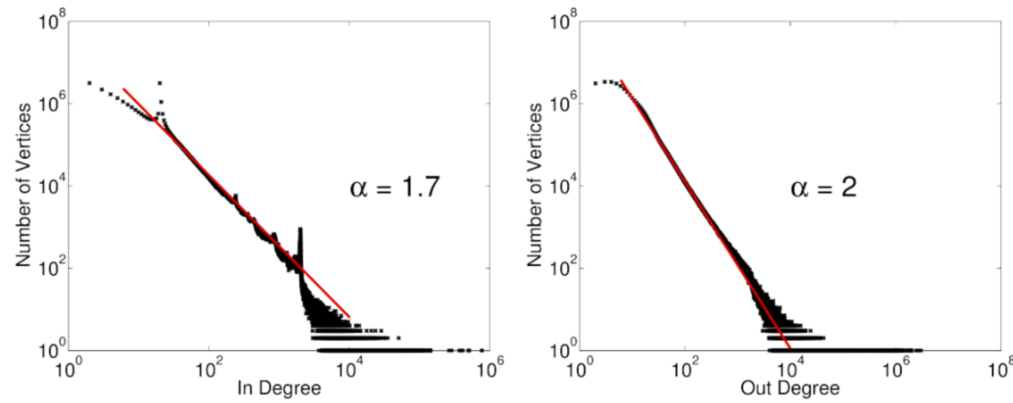
GAS Model: Gather Apply and Scatter

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

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

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

Challenges of Natural Graphs



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

$$\mathbf{P}(d) \propto d^{-\alpha},$$

Work Balance

Partitioning

Communication

Storage

Computation

```
interface GASVertexProgram(u) {
  // Run on gather_nbrs(u)
  gather(D_u, D_(u,v), D_v) → Accum
  sum(Accum left, Accum right) → Accum
  apply(D_u, Accum) → D_u^new
  // Run on scatter_nbrs(u)
  scatter(D_u^new, D_(u,v), D_v) → (D_(u,v)^new, Accum)
}
```

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

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**
    $a_u \leftarrow \mathrm{sum}(a_u, \mathrm{gather}(D_u, D_{(u,v)}, D_v))$
  end
end
$D_u \leftarrow \mathrm{apply}(D_u, a_u)$
foreach *neighbor* v *scatter_nbrs(u)* **do**
  $(D_{(u,v)}, \Delta a) \leftarrow \mathrm{scatter}(D_u, D_{(u,v)}, D_v)$
  if a_v *and* Δa *are not Empty* **then** $a_v \leftarrow \mathrm{sum}(a_v, \Delta a)$
  else $a_v \leftarrow$ Empty
end

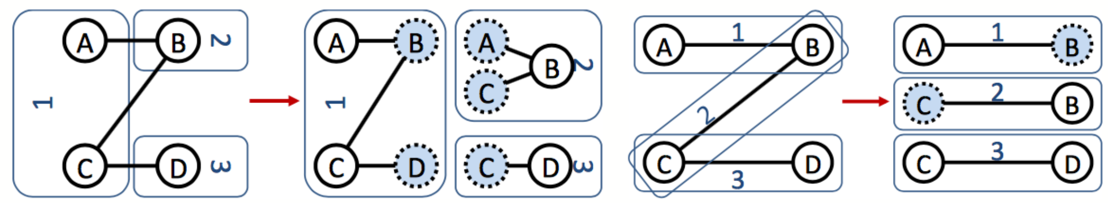
Delta Caching

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

PageRank

```
// 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
```

Distributed Graph Placement



(a) Edge-Cut (b) Vertex-Cut

Balanced p-way edge cut

Theorem 5.1. *If vertices are randomly assigned to p machines then the expected fraction of edges cut is:*

$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|E|}\right] = 1 - \frac{1}{p}. \tag{5.1}$$

For a power-law graph with exponent α, the expected number of edges cut per-vertex is:

$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|V|}\right] = \left(1 - \frac{1}{p}\right)\mathbb{E}\left[\mathbf{D}[v]\right] = \left(1 - \frac{1}{p}\right)\frac{\mathbf{h}_{|V|}(\alpha - 1)}{\mathbf{h}_{|V|}(\alpha)}, \tag{5.2}$$

where the $\mathbf{h}_{|V|}(\alpha) = \sum_{d=1}^{|V|-1} d^{-\alpha}$ is the normalizing constant of the power-law Zipf distribution.

Proof. An edge is cut if both vertices are randomly assigned to different machines. The probability that both vertices are assigned to different machines is $1 - 1/p$. $\square$

Distributed Graph Placement

Balanced p-way vertex cut

$$\min_{A} \frac{1}{|V|} \sum_{v \in V} |A(v)| \tag{5.3}$$

$$\text{s.t.} \quad \max_{m} |\{e \in E \mid A(e) = m\}|, < \lambda \frac{|E|}{p} \tag{5.4}$$

Theorem 5.2 (Randomized Vertex Cuts)**.** *A random vertex-cut on p machines has an expected replication:*

$$\mathbb{E}\left[\frac{1}{|V|}\sum_{v \in V}|A(v)|\right] = \frac{p}{|V|}\sum_{v \in V}\left(1 - \left(1 - \frac{1}{p}\right)^{\mathbf{D}[v]}\right). \tag{5.5}$$

where $\mathbf{D}[v]$ *denotes the degree of vertex v. For a power-law graph the expected replication (Fig. 6a) is determined entirely by the power-law constant* α*:*

$$\mathbb{E}\left[\frac{1}{|V|}\sum_{v \in V}|A(v)|\right] = p - \frac{p}{\mathbf{h}_{|V|}(\alpha)}\sum_{d=1}^{|V|-1}\left(\frac{p-1}{p}\right)^{d} d^{-\alpha}, \tag{5.6}$$

where $\mathbf{h}_{|V|}(\alpha) = \sum_{d=1}^{|V|-1} d^{-\alpha}$ *is the normalizing constant of the power-law Zipf distribution.*

Greedy Vertex Cut

$$\arg\min_{k} \mathbb{E}\left[\sum_{v \in V} |A(v)| \,\middle|\, A_i, A(e_{i+1}) = k\right], \tag{5.13}$$

Case 1: If $A(u)$ and $A(v)$ intersect, then the edge should be assigned to a machine in the intersection.

Case 2: If $A(u)$ and $A(v)$ are not empty and do not intersect, then the edge should be assigned to one of the machines from the vertex with the most unassigned edges.

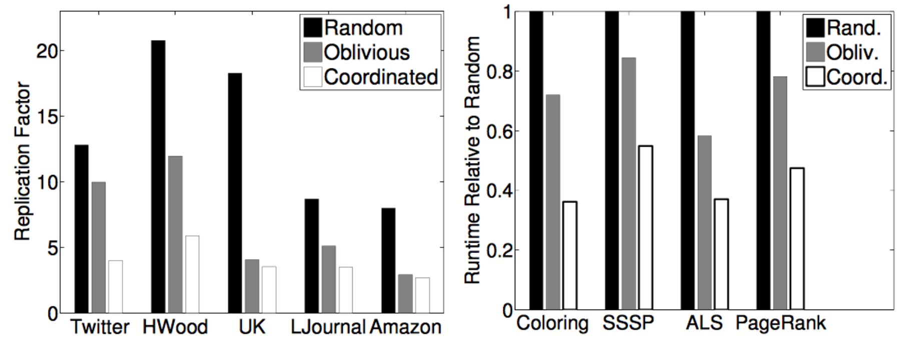
Case 3: If only one of the two vertices has been assigned, then choose a machine from the assigned vertex.

Case 4: If neither vertex has been assigned, then assign the edge to the least loaded machine.

Coordinated: maintains the values of $A_i(v)$ in a distributed table. Then each machine runs the greedy heuristic and periodically updates the distributed table. Local caching is used to reduce communication at the expense of accuracy in the estimate of $A_i(v)$.

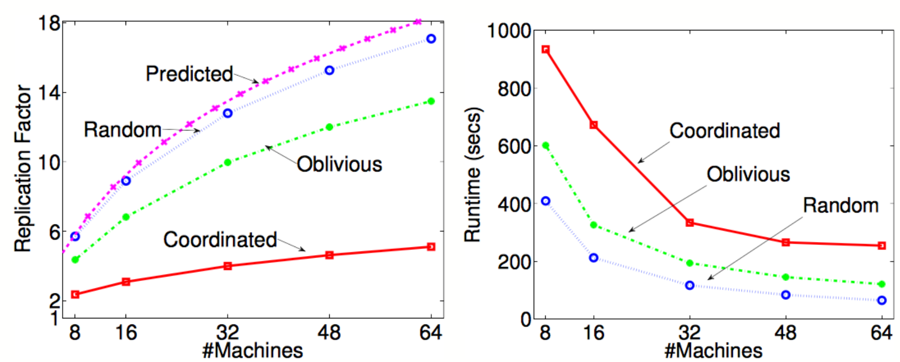
Oblivious: runs the greedy heuristic independently on each machine. Each machine maintains its own estimate of A_i with no additional communication.

Experiments



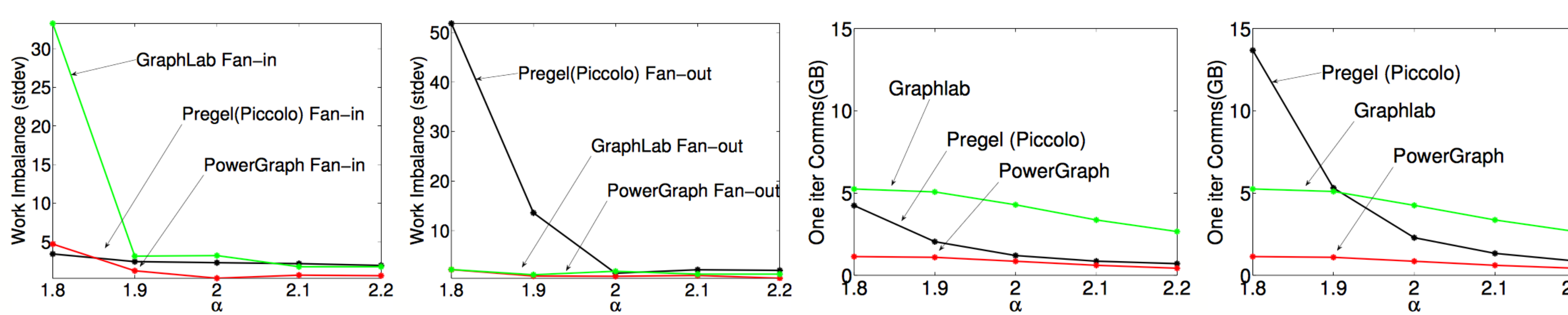
(a) Actual Replication (b) Effect of Partitioning

Figure 7: **(a)** The actual replication factor on 32 machines. **(b)** The effect of partitioning on runtime.



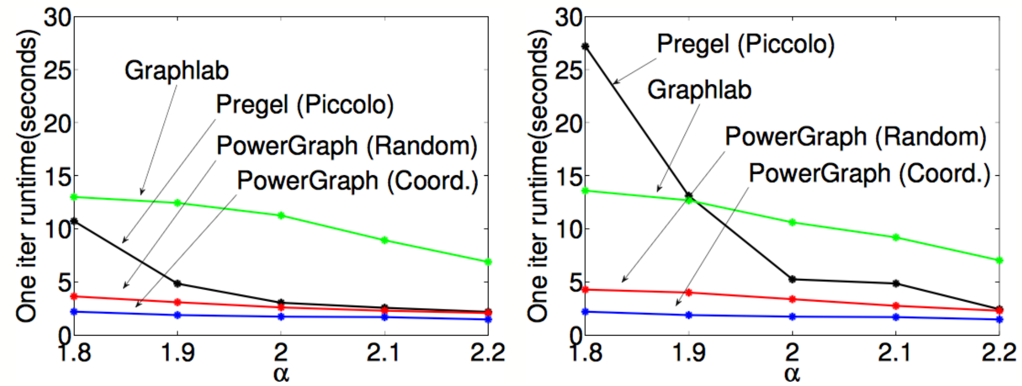
(a) Replication Factor (Twitter) (b) Ingress time (Twitter)

Figure 8: **(a,b)** Replication factor and runtime of graph ingress for the Twitter follower network as a function of the number of machines for random, oblivious, and coordinated vertex-cuts.

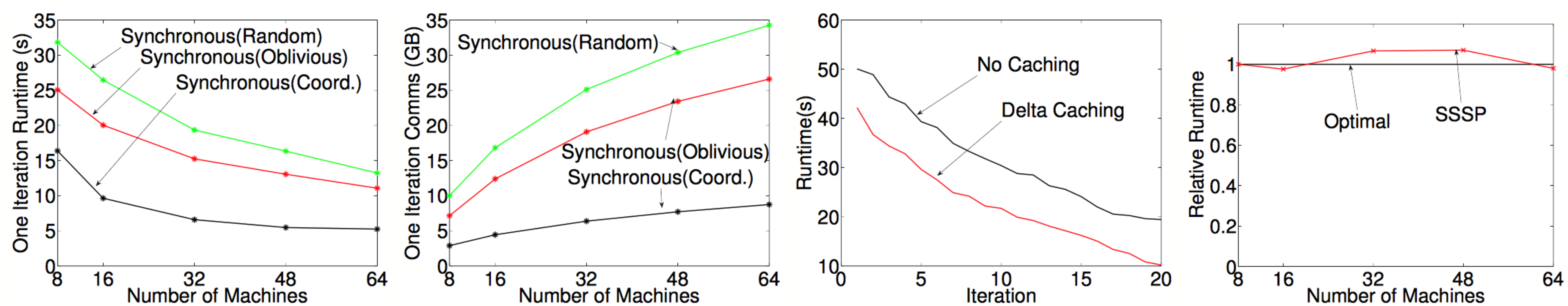


(a) Power-law Fan-In Balance (b) Power-law Fan-Out Balance (c) Power-law Fan-In Comm. (d) Power-law Fan-Out Comm.

Experiments

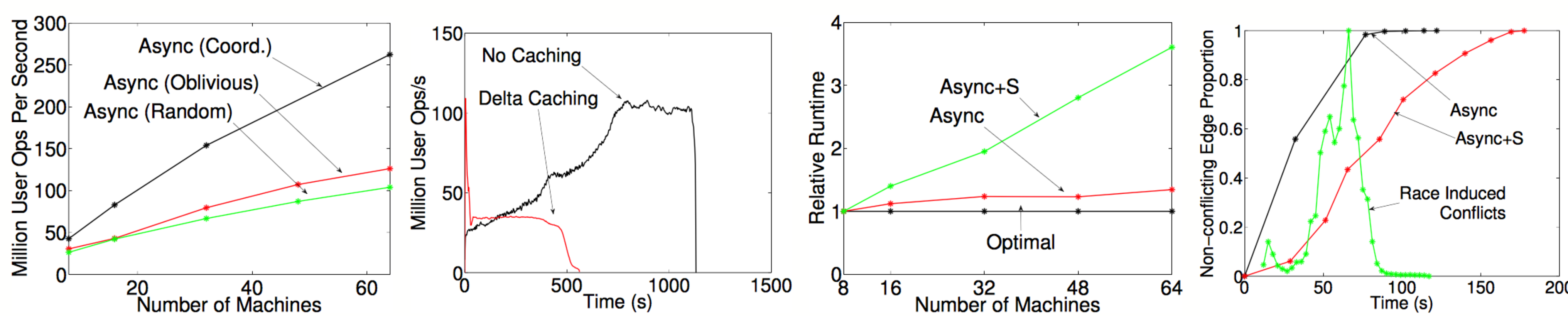


(a) Power-law Fan-In Runtime (b) Power-law Fan-Out Runtime



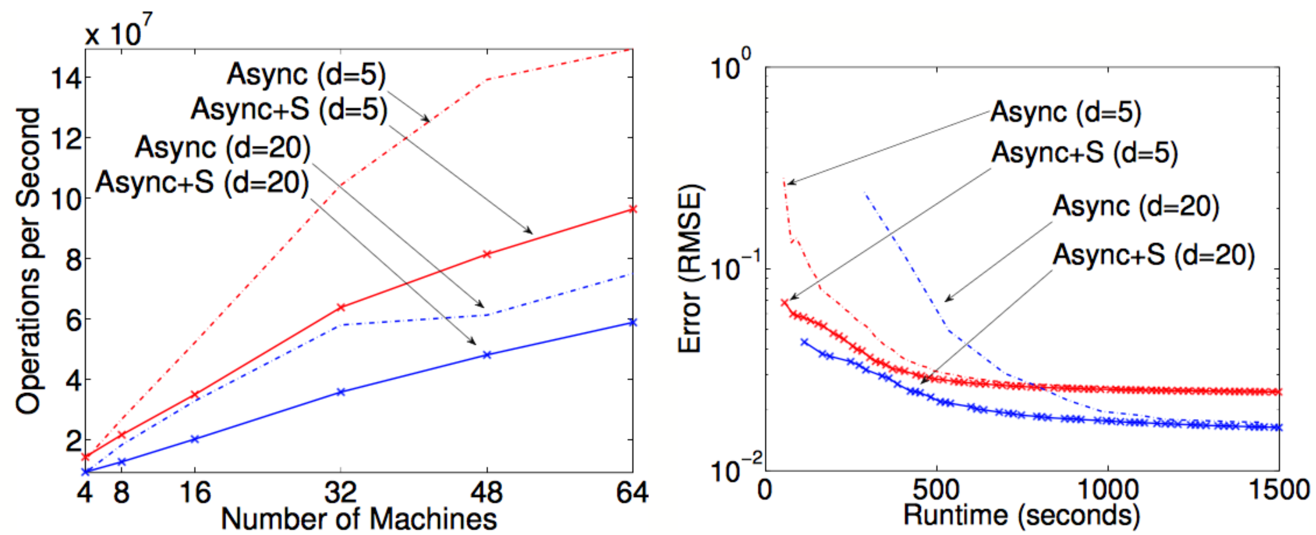
(a) Twitter PageRank Runtime (b) Twitter PageRank Comms (c) Twitter PageRank Delta Cache (d) SSSP Weak Scaling

Figure 11: **Synchronous Experiments (a,b)** Synchronous PageRank Scaling on Twitter graph. **(c)** The PageRank per iteration runtime on the Twitter graph with and without delta caching. **(d)** Weak scaling of SSSP on synthetic graphs.



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

Experiments



(a) ALS Throughput (b) ALS Convergence

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

Table 2: Relative performance of PageRank, triangle counting, and LDA on similar graphs. PageRank runtime is measured per iteration. Both PageRank and triangle counting were run on the Twitter follower network and LDA was run on Wikipedia. The systems are reported as number of nodes by number of cores.