PowerGraph
Distributed Graph-Parallel Computation on Natural Graphs

JOSHUA SEND

24/10/2017

LSDPO SESSION 3

Intuition for Graph Processing Systems

Overall goal

- efficiently compute over large graphs of data – key is distributing work

Typical tasks: Single Source Shortest Path, PageRank etc.

Approach

- Define computation graph on the data rather than passing graph through computation steps

Existing Systems – **Pregel** [1]

Input data graph

- Assign computation to each vertex, vertices to instances

Synchronous supersteps

Directed Edges

Existing Systems – **GraphLab** [2]

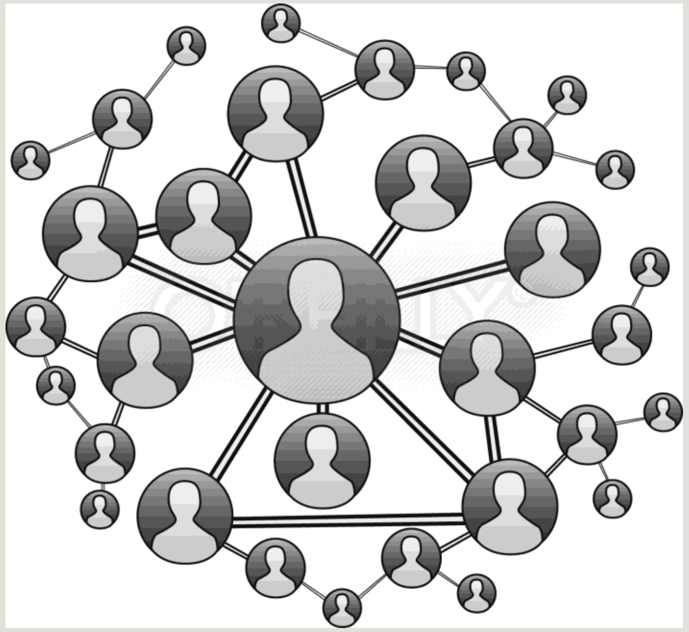
Also facilitates processing large graphs of data and distributes graph vertices to instances

No explicit message passing and directed edges

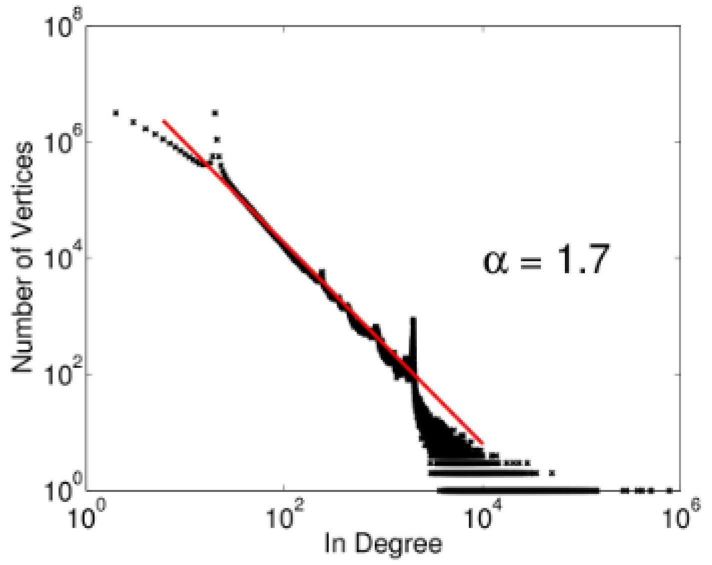
Asynchronous execution – no supersteps

Motivation

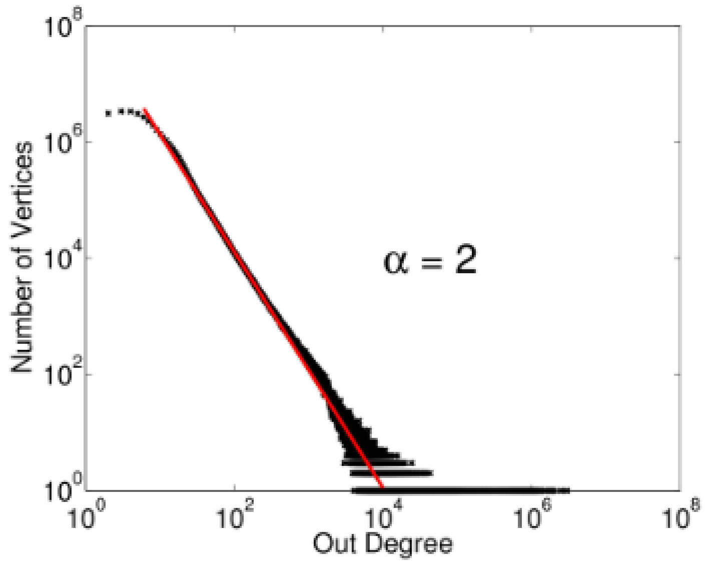
- Power Law connectivity: $P(d) \propto d^{-\alpha}$
- Eg. Social networks, internet ($\alpha \approx 2$)



Natural Graphs



(a) Twitter In-Degree



(b) Twitter Out-Degree

Contributions

1. Generalized “vertex program”
2. Distribute graph edge-by-edge rather than vertex-by-vertex
3. Practical parallel locking

Generalized Vertex Program

Gather

- Collect data and aggregate
- Commutative, associative aggregator

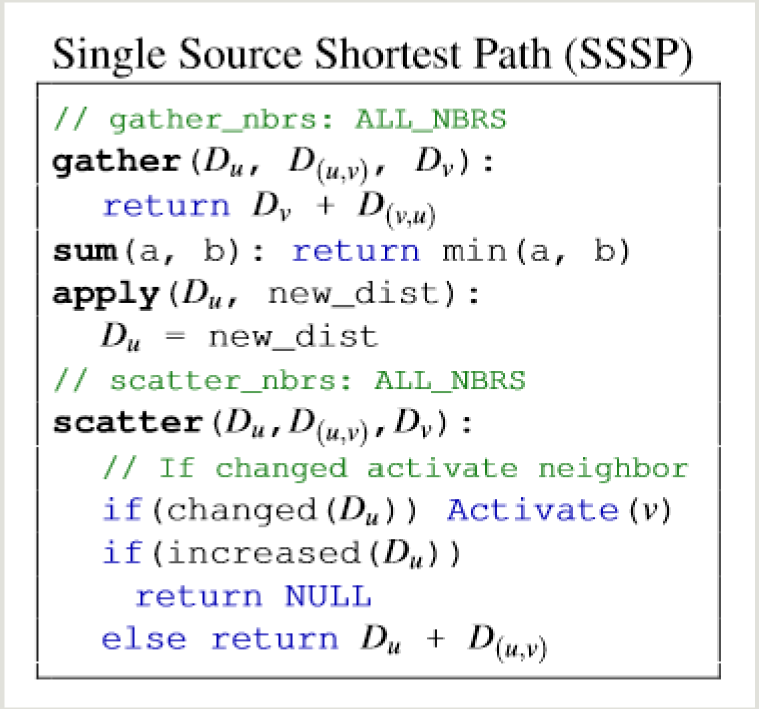
Apply

- Perform operation on gathered data

Scatter

- Disseminate to neighbors
- Activate their operation

SSSP



Single Source Shortest Path (SSSP)

```
// gather_nbrs: ALL_NBRS
gather($D_u$, $D_{(u,v)}$, $D_v$):
  return $D_v$ + $D_{(v,u)}$
sum(a, b): return min(a, b)
apply($D_u$, new_dist):
  $D_u$ = new_dist
// scatter_nbrs: ALL_NBRS
scatter($D_u$,$D_{(u,v)}$,$D_v$):
  // If changed activate neighbor
  if(changed($D_u$)) Activate($v$)
  if(increased($D_u$))
    return NULL
  else return $D_u$ + $D_{(u,v)}$
```

Vertex Splitting

Standard approach – assign each vertex of graph to an instance – often requires ‘ghosts’

Idea – assign each edge to an instance

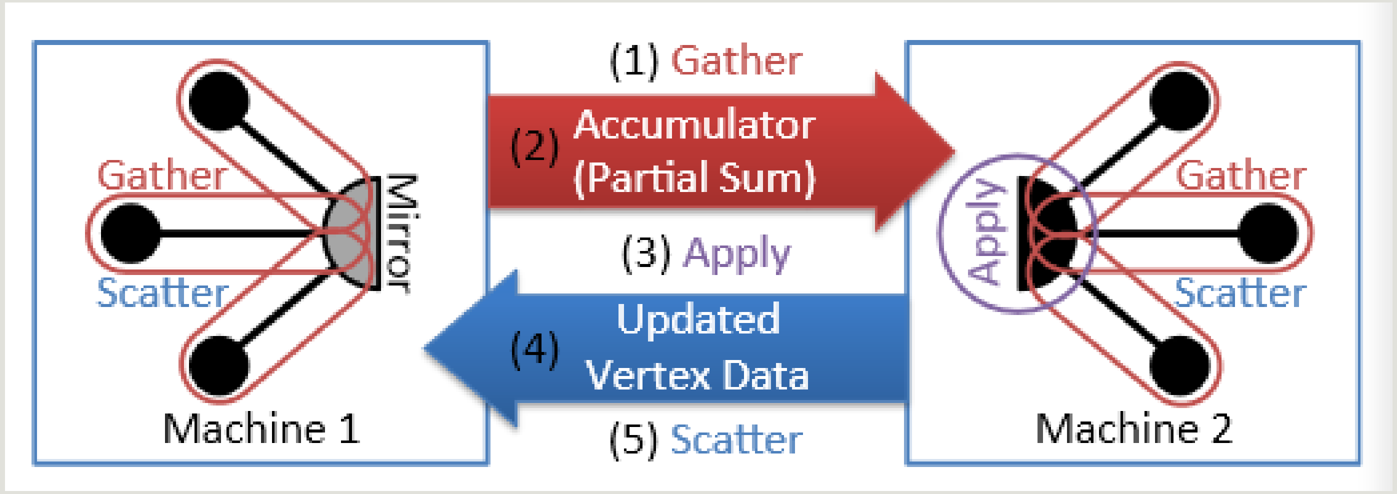
Leads to vertices appearing on different instances

Parallelization of data gathering and scattering “within” one vertex as edges may be in different instances

Set of instances containing a particular vertex called *replicas* and randomly assign a master, rest are called mirrors

Master receives partial aggregations, applies vertex operation, sends changes to edges to scatter

Master, Mirrors



How to actually distribute Edges

3 different strategies

1. Random
   - Deploy edge to instance based on hash
2. Greedy Heuristic
   - Reduce number of replicas per vertex
   - Requires estimate of sets of replicas per vertex

Heuristic Distribution

1. Oblivious
   - Estimate sets from local information only
   - Paper unclear on how exactly this works
2. Coordinated
   - Keep distributed table of sets replicas per vertex

Tradeoff space: longer load time vs. fewer replicas & faster execution

Execution Stategies

Supports:

- Synchronized supersteps (à la Pregel),
- Asynchronous
- Asynchronous + serializable utilizing parallel locking

Tradeoff space: predictability/determinism vs throughput vs runtime/convergence speed

Miscellaneous

Delta Caching

- Update edges with deltas rather than rewriting values. If delta is 0, neighbor may not have to recompute

Fault Tolerance

- Checkpointing

Results

Partitioning scheme

- Random > oblivious > coordinated in terms of replication factor
- All faster than Pregel/Piccolo and GraphLab for synthetic natural graphs

Execution Strategy

- Synchronized: 3-8x faster implementing PageRank than on Spark per iteration
- Async: Even faster (authors don’t provide a direct comparison?)
- Async + Serializable: less throughput, converges faster (less recomputation)

Remarks

Paper’s details are hard to understand

Evaluation is a bit sloppy – missing some direct comparisons between execution strategies and combinations of partitioning and execution

Large tradeoff space, hard to navigate

- Eg. Coordinated distribution can increase load times 4x
- Authors highlight 60s vs 240s for random vs coordinated partitioning
- Meanwhile, SSSP on 6.5B edges takes 65s to run

Remarks

Solid theoretical foundation for partitioning heuristic

Very solid gains over prior systems, especially in tasks with natural graphs!

References

1. G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski: Pregel: A System for Large-Scale Graph Processing, SIGMOD, 2010.
2. Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. Hellerstein: Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud, VLDB, 2012.
3. J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin: Powergraph: distributed graph-parallel computation on natural graphs. OSDI, 2012.