Pregel: A System for Large-Scale Graph Processing

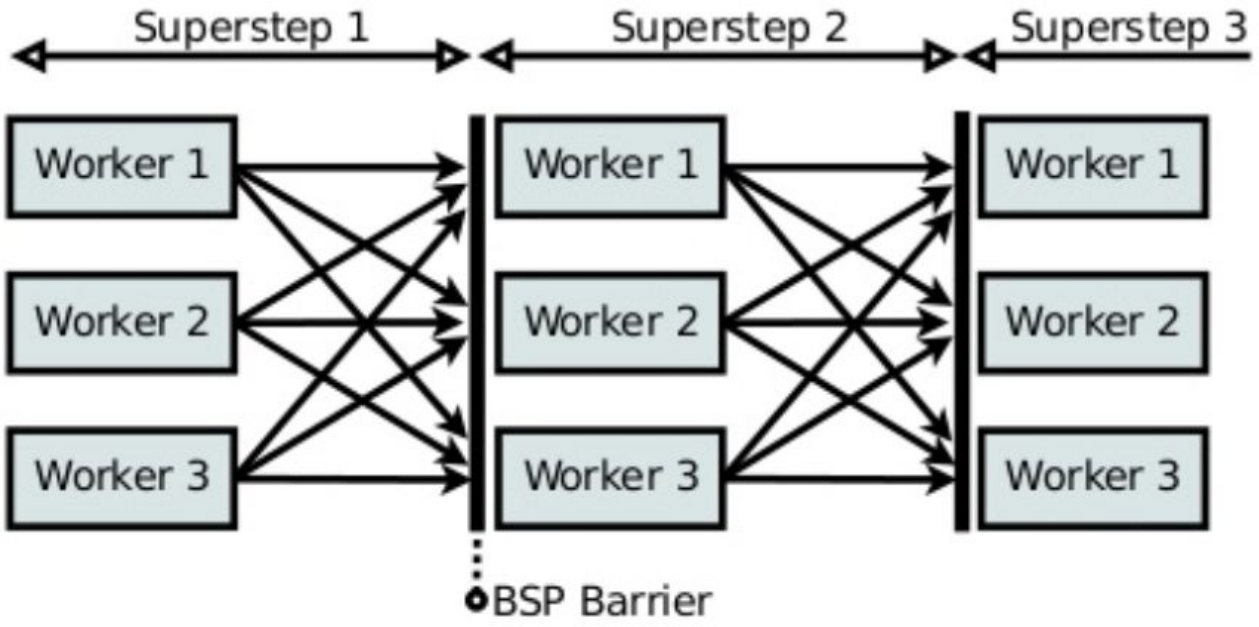
Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik,
James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski
Google, Inc.

R244 Presentation By: Vikash Singh October 24, 2018 Session 3

What is Pregel?

- General purpose system for flexible graph processing
- Efficient, scalable, and fault-tolerant implementation in a large-scale distributed environment

Bulk Synchronous Parallel Model (BSP)[1]



Pros and Cons of BSP for Distributed Graph Processing

- **Pro:** Naturally suited for distributed implementation
  - Order does **NOT** matter within a superstep
  - All communication is **BETWEEN** supersteps
- **Pro:** No deadlocks or data races to worry about
- **Pro:** Capable of balancing the load to minimize latency
- **Con:** As this scales to potentially millions of cores, barriers become expensive!

Termination Mechanism

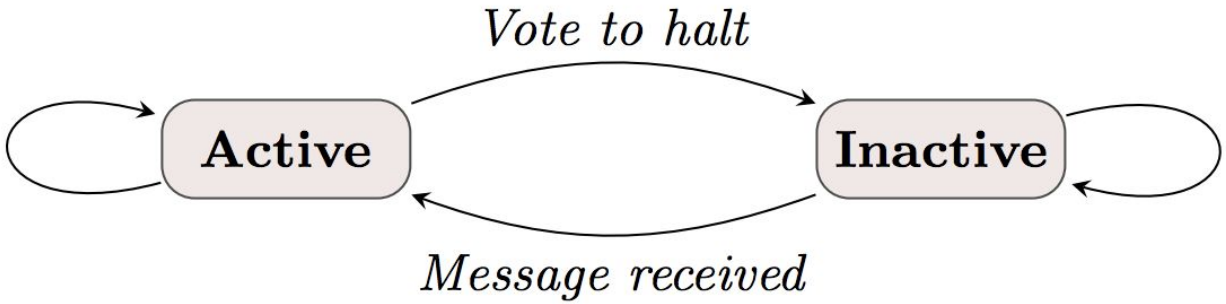


Figure 1: Vertex State Machine

Key Decision: Message Passing vs. Shared Reads

- Message passing expressive enough, especially for graph algorithms
- Remote reads have a high latency
- Message passing can be done asynchronously in batches

Comparison to MapReduce

- Graph algorithms can be written as a series of chained MapReduce invocations
- MapReduce would require passing the entire state of the graph from one state to the next, more overhead and communication
- Complexity added that would be taken care of by convenient supersteps in BSP

C++ API Overview

- Vertex class, virtual Compute() function (aka the instructions for each superstep)
- Compute function flexible to change topology
- Combiners/Aggregators available
- Handlers

Master-Worker Architecture

- **Master** assigns **partitions** of vertices to workers
- **Master** coordinates **supersteps** and **checkpoints** (fault tolerance)
- **Workers** execute **compute()** functions for vertices and directly **exchange messages** with each other

Fault Tolerance

- Workers save state of partitions to persistent storage at checkpoint
- Ping messages to check worker availability
- Checkpoint frequency based on mean time to failure model
- Reassign partitions, revert to last checkpoint in failure instance

Master-Worker Implementation

Master

- Maintains list of all living workers (ID, addressing, partition)
- Coordinates supersteps through barrier synchronization/initiates recovery in failure
- Maintains stats on the progress of the graph, runs HTTP server that displays info

Worker

- Maintains the state of graph partition in memory (vertex id, current value, outgoing messages, queue for incoming messages, iterators to outgoing/incoming messages, active flag)
- Optimizations present for vertex message sending within same machine, or else use delivery buffer

How does Pregel Scale with Worker Tasks?

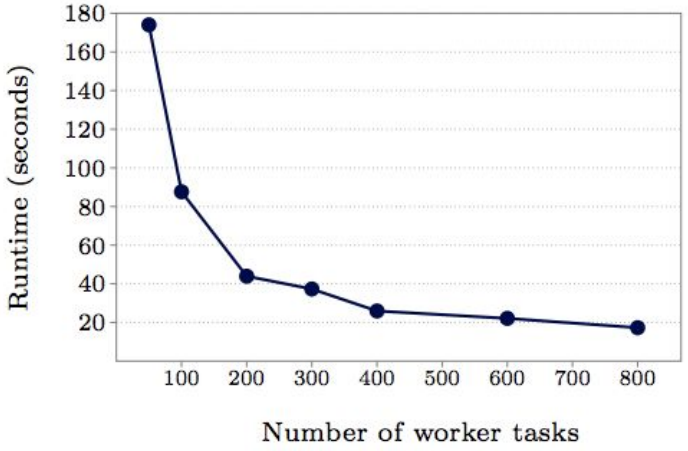


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multicore machines

Experiment Notes (General)

- 300 multicore commodity PCs
- Time for initializing cluster, generating the test graphs in memory, and verifying results not included
- Checkpointing was disabled

How does Pregel Scale with Graph Size (Binary Tree)?

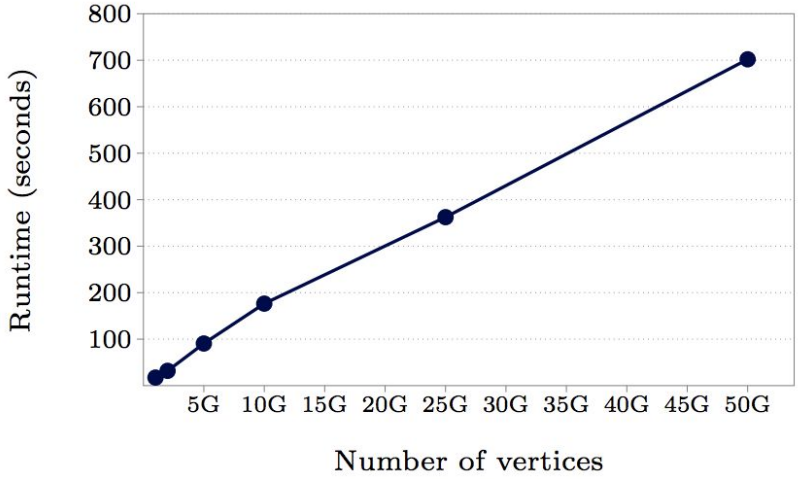


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

How does Pregel Scale with Graph Size (Log Normal Random Graph)?

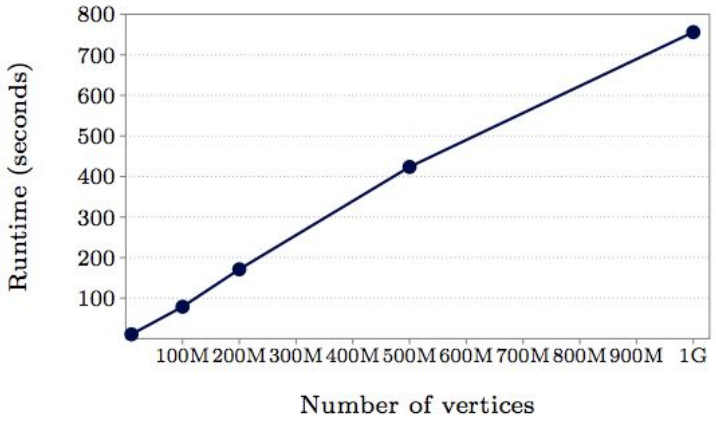


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Criticism

- No legitimate effort to compare to other systems such as MapReduce[3], Parallel BGL[4],CGMGraph[5], Dryad[2],
- No explanation of fault tolerance in case of failure of master
- Inefficient for imbalanced data (no dynamic repartitioning) **PowerGraph to the rescue!**
- Checkpointing disabled in experiments, fault tolerance not experimentally tested
- No experimental analysis of slow down from spill over of data to disk when RAM gets full

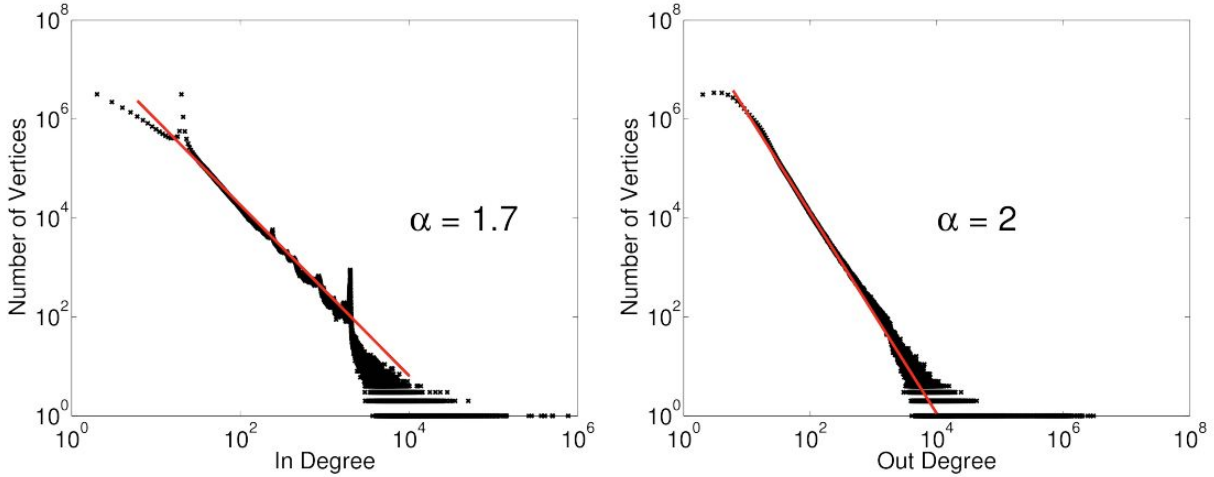
PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin:

Digging into Pregel's Load Imbalance Issue

- Natural graphs often have skewed power-law degree distribution, causes significant imbalance in a vertex-centric system such as Pregel
- Storage, computation, and communication issues
- No parallelization within each vertex

Visualizing Power-Law Degree Distribution

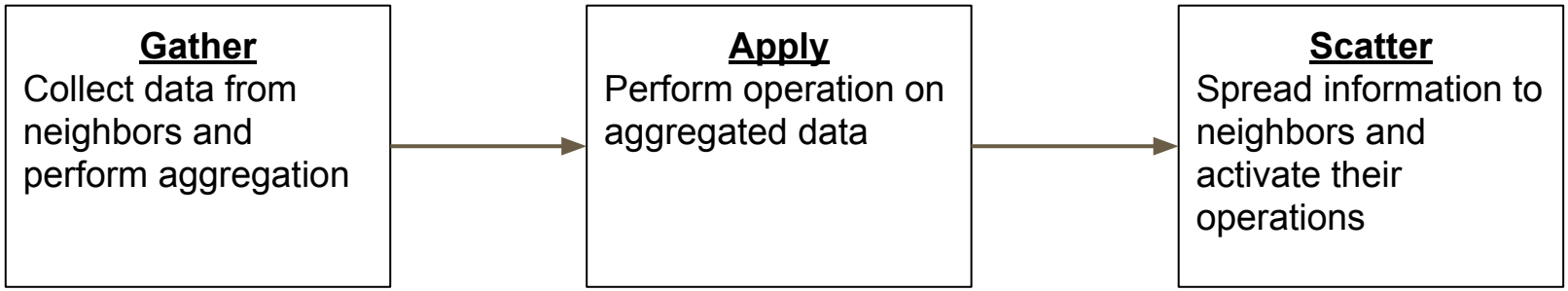


(a) Twitter In-Degree (b) Twitter Out-Degree

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

Powergraph Solution

- Distribute edges rather than vertices, allowing for parallelization of huge vertices (vertex-cut)
- Execution of vertex program, using Gather, Apply, Scatter (GAS) model



Vertex-Cut Communication

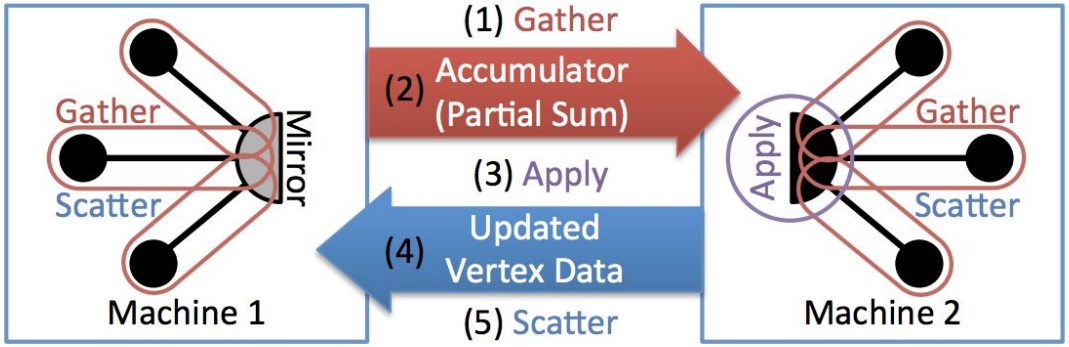
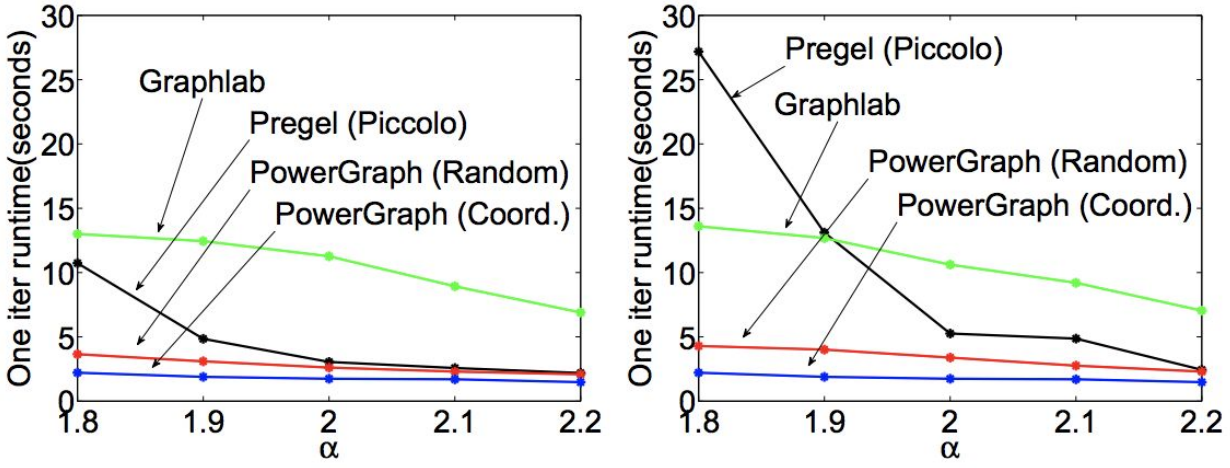


Figure 5: The communication pattern of the PowerGraph abstraction when using a vertex-cut. Gather function runs locally on each machine and then one accumulators is sent from each mirror to the master. The master runs the apply function and then sends the updated vertex data to all mirrors. Finally the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

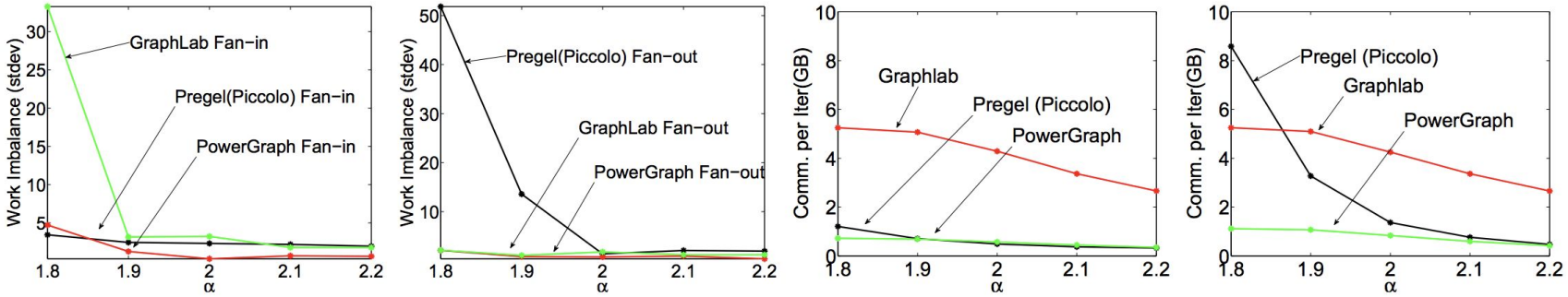
Runtime Comparison



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

Figure 10: **Synthetic Experiments Runtime. (a, b)** Per iteration runtime of each abstraction on synthetic power-law graphs.

Worker Imbalance and Communication Comparison



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

Figure 9: **Synthetic Experiments: Work Imbalance and Communication. (a, b)** Standard deviation of worker computation time across 8 distributed workers for each abstraction on power-law fan-in and fan-out graphs. **(b, c)** Bytes communicated per iteration for each abstraction on power-law fan-in and fan-out graphs.

Final Thoughts

- Pregel mostly achieved its main goal: a flexible distributed framework for graph processing
- Weak experimental data and comparisons, however it is in production on multiple systems at Google so we have some degree of faith
- Powergraph solves issue of load imbalance in Pregel's method of distributed graph processing

References

1. Leslie G. Valiant, A Bridging Model for Parallel Computation. Comm. ACM 33(8), 1990, 103–111.
2. Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly, Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. in Proc. European Conf. on Computer Syst., 2007, 59–72.
3. Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters. in Proc. 6th USENIX Symp. on Operating Syst. Design and Impl., 2004, 137–150
4. Douglas Gregor and Andrew Lumsdaine, The Parallel BGL: A Generic Library for Distributed Graph Computations. Proc. of Parallel Object-Oriented Scientific Computing (POOSC), July 2005.
5. Albert Chan and Frank Dehne, CGMGRAPH/CGMLIB: Implementing and Testing CGM Graph Algorithms on PC Clusters and Shared Memory Machines. Intl. J. of High Performance Computing Applications 19(1), 2005, 81–97.