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

Presented by: Ramya Nagarajan
Spring 2019

Agenda

- Motivation
- Related Work
- Model of Computation
- Execution Architecture
- Experiments
- Final Remarks

Motivation

- Graph algorithms don’t lend themselves to scalability and efficiency
  - Poor locality of memory access
  - Changing degree of parallelism over course of execution
- No scalable system for arbitrary graph algorithms
- Need for scalable general-purpose system for executing graph algorithms in large-scale distributed environment

Related/Prior Work

- Existing distributed systems:
  - ie: MapReduce
  - Sub-optimal performance and usability
- Single-computer graph algorithm libraries
  - BGL, LEDA, NetworkX, JDSL, GraphBase, FGL
  - Not scalable
- Existing parallel graph systems
  - Parallel BGL, CGMgraph
  - No fault tolerance
- Valiant’s Bulk Synchronous Parallel Model

Model of Computation

- Input:
    - Directed graph, each vertex labeled with a string identifier

Model of Computation

- Input:
  - Directed graph, each vertex labeled with a string identifier
- Series of supersteps

Model of Computation

- Input:
    - Directed graph, each vertex labeled with a string identifier
- Series of supersteps
    - Superstep 0: all vertices active

Model of Computation

- Input:
  - Directed graph, each vertex labeled with a string identifier
- Series of supersteps
  - Superstep 0: all vertices active
  - Supersteps 1 - n:
    - Vertex computations
    - Message passing between vertices
    - Topology mutations

Model of Computation

- Input:
    - Directed graph, each vertex labeled with a string identifier
- Series of supersteps
    - Superstep 0: all vertices active
    - Supersteps 1 - n:
        - Vertex computations
        - Message passing between vertices
        - Topology Mutations
    - Vertex halting
        - No associated computations or messages

Model of Computation

- Input:
  - Directed graph, each vertex labeled with a string identifier
- Series of supersteps
  - Superstep 0: all vertices active
  - Supersteps 1 - n:
    - Vertex computations
    - Message passing between vertices
    - Topology mutations
  - Vertex halting
    - No associated computations or messages
- Termination Condition
  - All vertices have halted

Model of Computation

- Input:
    - Directed graph, each vertex labeled with a string identifier
- Series of supersteps
    - Superstep 0: all vertices active
    - Supersteps 1 - n:
        - Vertex computations
        - Message passing between vertices
        - Topology Mutations
    - Vertex halting
        - No associated computations or messages
- Termination Condition
    - All vertices have halted
- Output:
    - Set of vertex output values

Message Passing

- Vertices can send any number of messages to any other number of vertices in the graph
  - Send message at superstep S
  - Messages received at superstep S+1

Message Passing

- Vertices can send any number of messages to any other number of vertices in the graph
  - Send message at superstep S
  - Messages received at superstep S+1
- User-defined handlers:
  - Specify behavior if message receiver does not exist in graph

Message Passing

- Combiners
    - Function to combine result of all messages sent to a certain vertex
    - Use by subclassing pre-defined Combiner class

Message Passing

- Combiners
  - Function to combine result of all messages sent to a certain vertex
  - Use by subclassing pre-defined Combiner class
- Aggregators
  - Result made available to all vertices
  - Use by subclassing of pre-defined Aggregator class

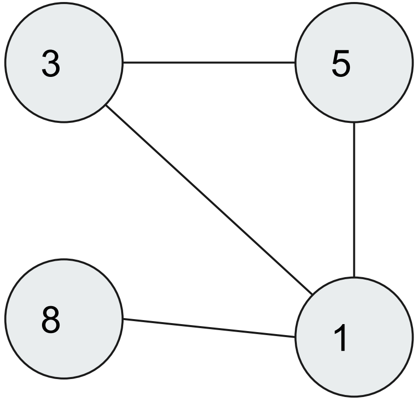
Topology Mutations

- Removals and additions
  - Add/remove edges
  - Add nodes
  - Remove nodes and outgoing edges

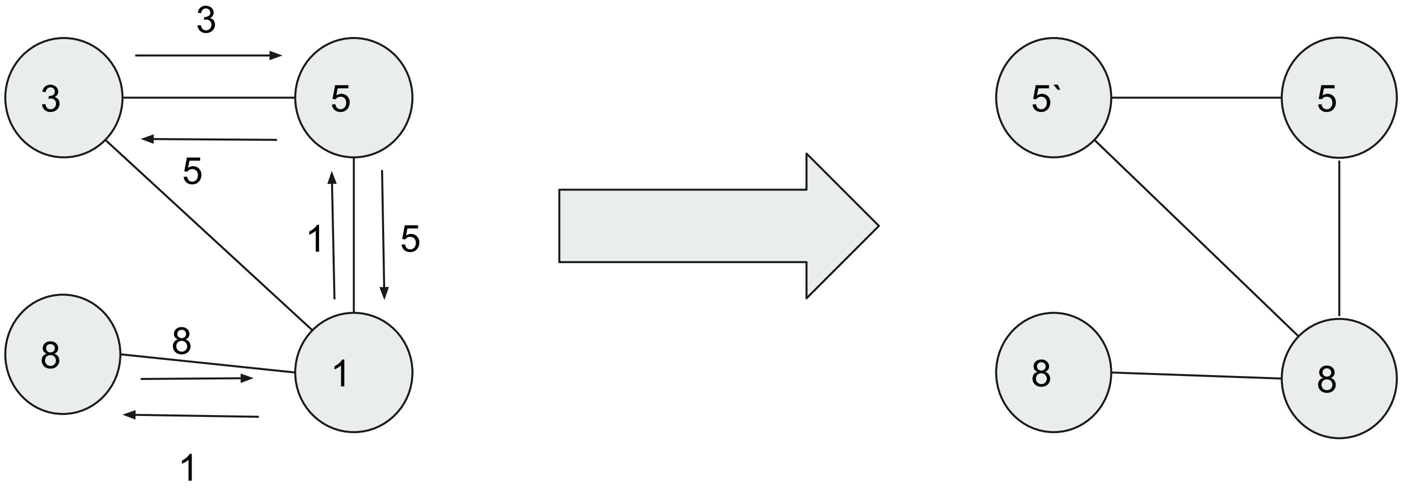
Topology Mutations

- Removals and additions
    - Add/remove edges
    - Add nodes
    - Remove nodes and outgoing edges
- Partial ordering
    - Removals before additions
    - Edge removals, vertex removals, vertex additions, edge additions
- Handlers

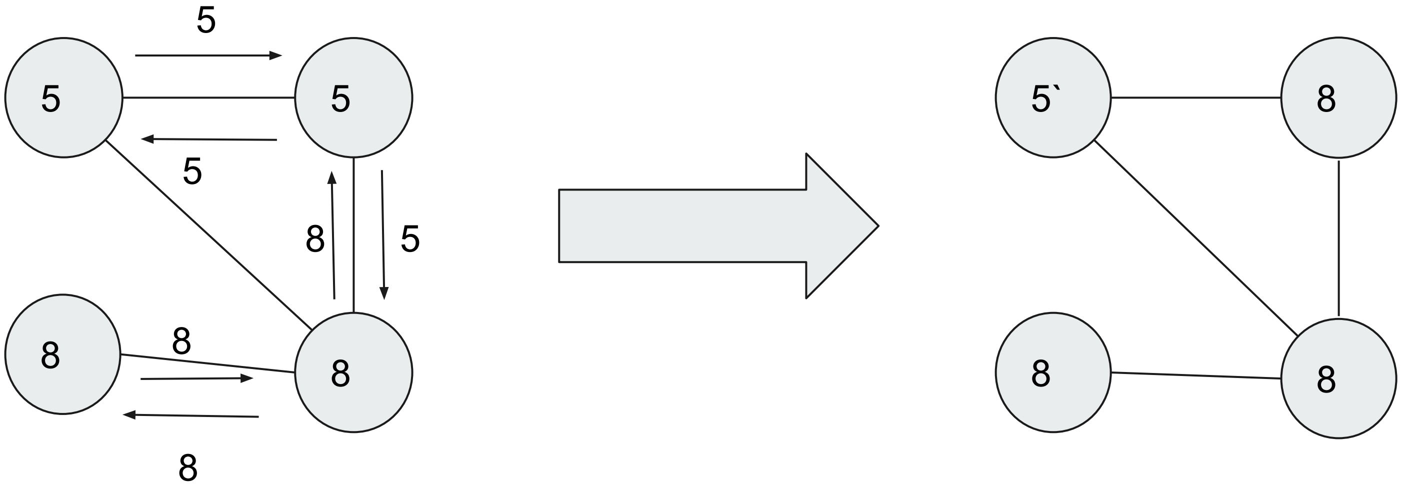
Maximum Value Calculation using Pregel



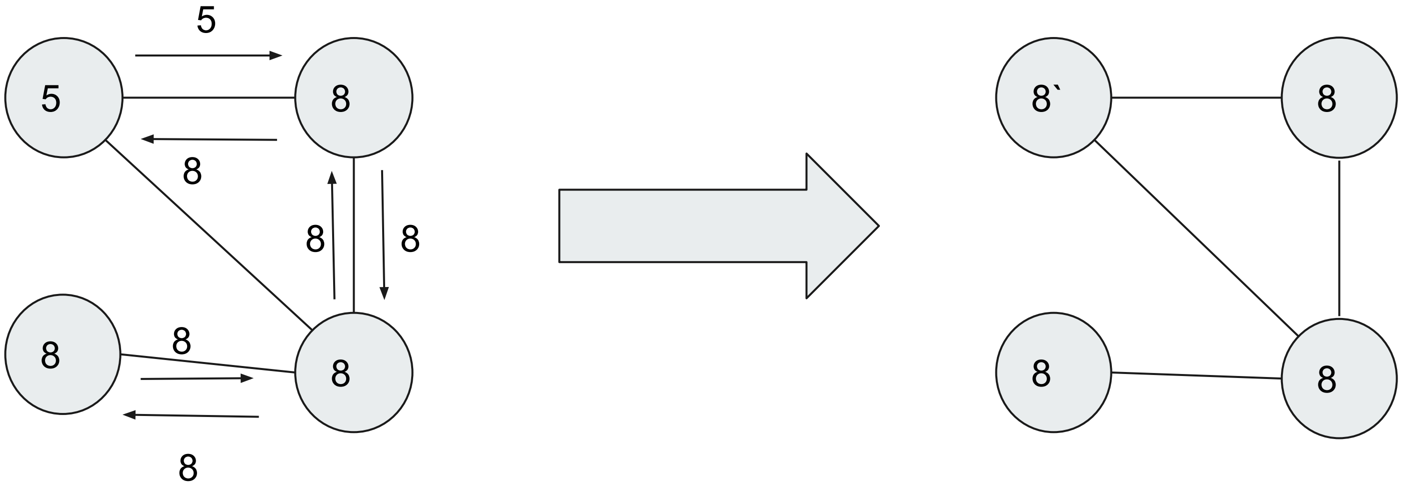
Maximum Value Calculation using Pregel



Maximum Value Calculation using Pregel



Maximum Value Calculation using Pregel



Applications

- PageRank
- Shortest Paths
- Bipartite Matching
- Semi-clustering

Example: Shortest Paths

- Implemented for single-source shortest path
- All vertices initialized to INF
- Superstep 0:
  - Source vertex updates value to 0, broadcasts to neighbors
- Subsequent supersteps:
  - Broadcast new minimum values
- Terminates when no remaining updates

Basic Architecture

- Graph partitioned into sets:
    - Each set contains a group of vertices + outgoing edges from vertices
- Worker and master machine separation

Basic Architecture

- Graph partitioned into sets:
  - Each set contains a group of vertices + outgoing edges from vertices
- Worker and master machine separation
- Execution Steps:
  - Many copies of the input graph run
  - Master determines number of partitions
  - Master assigns partitions to workers
  - Master orchestrates superstep start

Worker and Master Implementations

- Worker:
  - Loops through all vertices, performs Compute() step
  - Runs two threads:
    - Thread to process vertices
    - Thread to receive messages
- Master:
  - Coordinating worker activities
  - Barrier Synchronization

Fault Tolerance

- Checkpointing
  - Worker state saved
  - Frequency selected based on mean time to failure model
- “Ping” messages for failure detection
- Confined recovery

Experiments

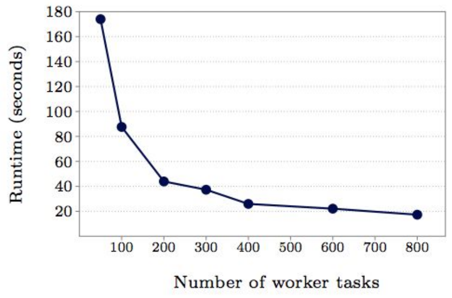


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multicore machines

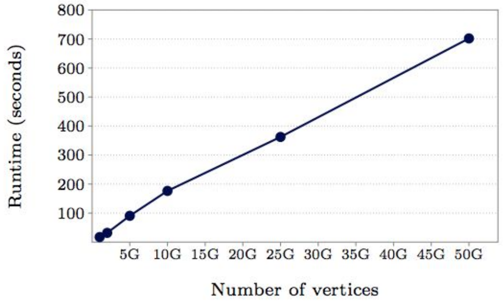


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Experiments

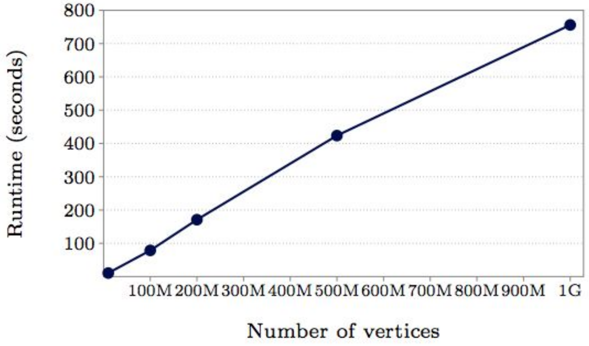


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Final Remarks

- Fault tolerant, scalable implementation of model
- “Think like a vertex” vs “Think like a graph”
  - Improved locality
  - Improved linear scalability
- Model for many other graph processing algorithms (ie Apache Giraph)