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 -2010

Presented by K.M.D.Muthumali Karunarathna

27th October 2015

Outline

- Problem
- Current Solutions
- Limitations of current solutions
- Pregel
- Future work

Problem

Billions of vertices,Trillions of edges poses challenges to their efficient processing in large graphs.

e.g.

Web graphs

Social Networks (Facebook, Twitter)

Frequently added algorithms

- Shortest path computation
- Different flavors of clustering (e.g. K-means, K-median)
- PageRank theme (PageRank is a “vote”, by all the other pages on the Web, about how important a page is)

Problems with graph algorithms

- Poor locality of memory access
- Very little work per vertex
- A changing degree of parallelism over the course of execution

Implementing an algorithm for a large graph

- Crafting custom distributed infrastructure
- Relying on an existing distributed computing platform like MapReduce
- Using single computer graph algorithm library such as BGL, LEDA
- Using an existing parallel graph system like BGL, CGMgraph

None of this alternatives fit the author's purpose

Solution simply

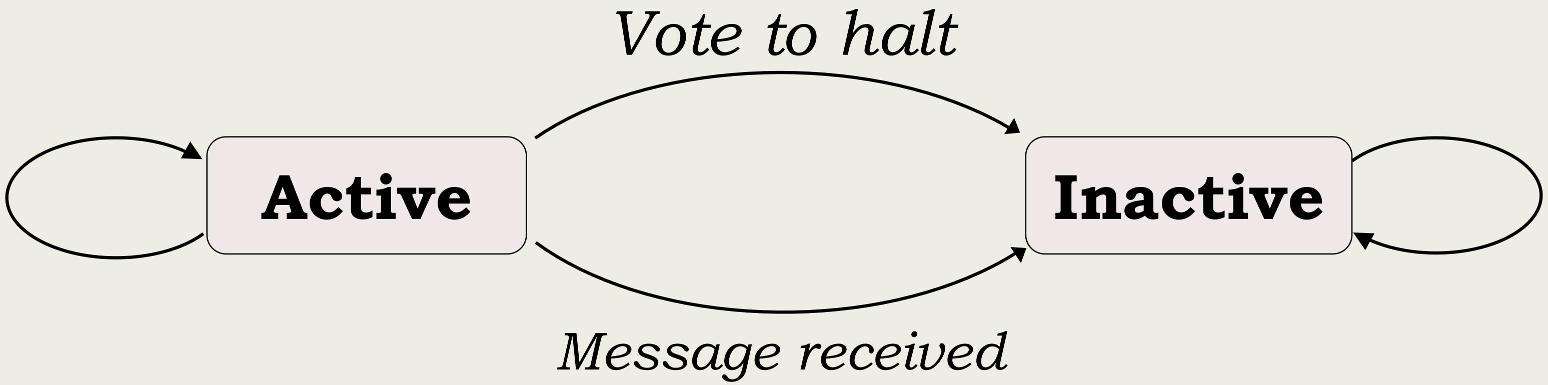
- A computational model,
- Which expressed as a sequence of iterations
- A vertex can receive messages sent in the previous iteration
- Send messages to other vertices
- Modify its own state and that of its outgoing edges

- Efficient, Scalable, Fault tolerance implementation on clusters
- Its implied synchronicity makes reasoning about programs easier

Model of computation

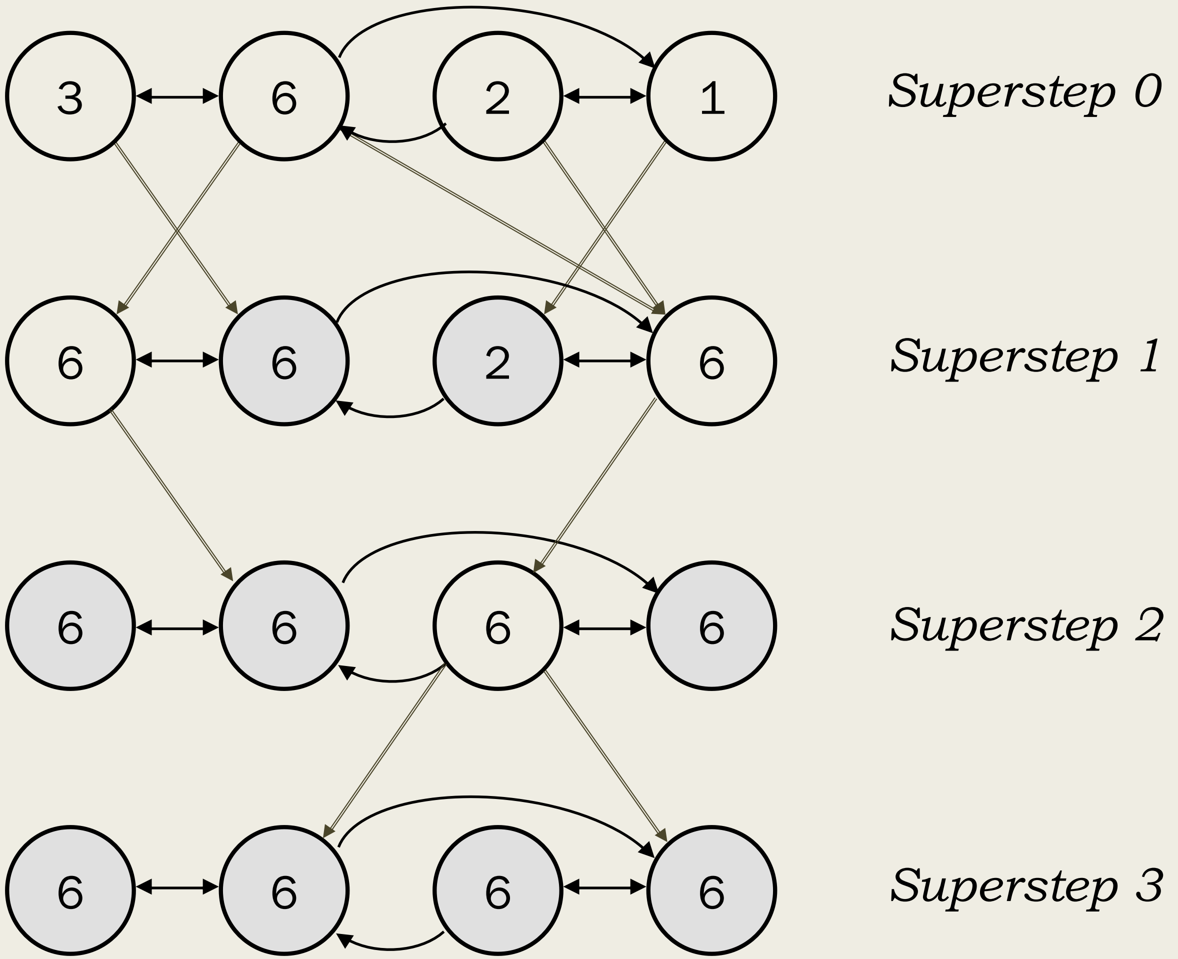
- Consists of a sequence of iterations (supersteps), where the same user-defined function is executed for each vertex.
- This function specifies behavior at a single vertex V and superset S. It can read messages sent to the vertex in super Steps-1, send messages to other vertices that will be read in superset S+1, and modify that state of V and its outgoing edges.

Vertex State Machine



- Initially each vertex is in an active state. Each vertex can ‘vote to halt’, where it runs no further computation in any further super step unless it receives a message from another vertex. ,It is then reactivated again and needs to explicitly vote to halt to deactivate itself again. Algorithm terminates when all vertices have halted.

Pregal - Find maximum value



Pregel Solution

- Allows efficient processing of large, distributively-stored, graphs.
- Abstracts away distributed computer related issues like fault tolerance .
- A ‘vertex-centric system’

  - All programmer needs to do is outline a single function.

Pregel in detailed

- Master node

1. Coordinates and maintains a list of all workers.
2. Maintains aggregator

- Aggregators

1. Nodes send master a value at each iteration for aggregation.
2. Provides a global statistic to each node at each super step, important for some algorithms like Dijkstra's algorithm

- Combiners
  - Combines messages to reduce message traffic.

- Input and outputs
  - Can be generated from any arbitrary format and stored in a form most suitable for a given application.

- Fault tolerance.
  - Achieved through checkpointing
  - Master instructs workers to save their state to persistent storage at the beginning of each superstep (Vertex values, Edge values, Incoming messages)
  - If Master detects these workers as down, it reassigns their partitions to available workers and recomputes the superstep

Pregel example - SSSP

```
class ShortestPathVertex
        : public Vertex<int, int, int> {
   void Compute(MessageIterator* msgs) {
        int mindist = IsSource(vertex_id()) ? 0 : INF;
        for (; !msgs->Done(); msgs->Next())
              mindist = min(mindist, msgs->Value());
       if (mindist < GetValue()) {
              *MutableValue() = mindist;
              OutEdgeIterator iter = GetOutEdgeIterator();
              for (; !iter.Done(); iter.Next())
                    SendMessageTo(iter.Target(), mindist + iter.GetValue());
       }
       VoteToHalt();
   }
};
```

Figure 5: Single-source shortest paths

Experiments - SSSP with varying graph size and worker numbers

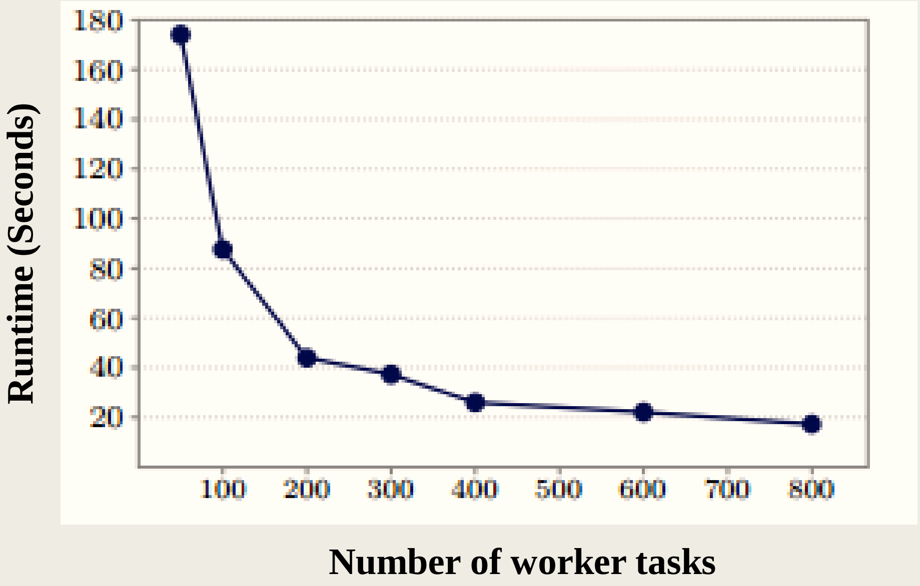


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multicore machines

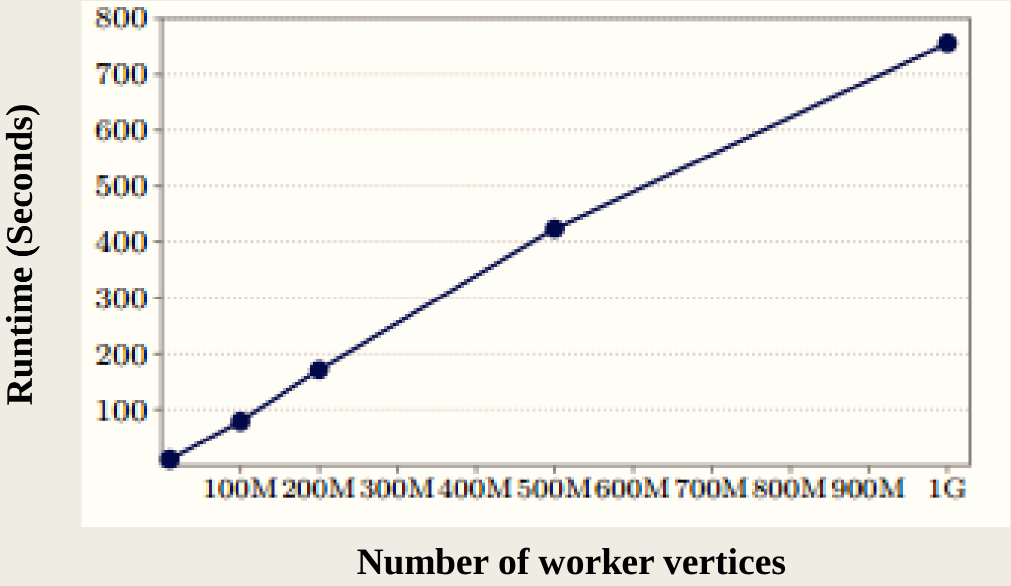


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Critical Analysis

- What will happen if fault tolerance occurs and it’s not clear whether only the work for the reassigned graph partition or the entire work for that super step is recomputed?

- They doesn't address when infinite loops might occur and how to account for them

Future Work

- Partitioning based on the graph
- Handle complex parallelizable functions over the whole graph
- Avoid waiting for slow workers
- Confined recovery to improve the cost and latency of recovery

THANK YOU