Pregel

Large-Scale Graph Processing

William Jones

Analysing large graphs is hard.

- We are keenly interested in analysing certain very large graphs. (e.g. the Web graph)
- These graphs are now too large to store and process on one machine.
- Parallelising graph algorithms is hard:
  - Poor memory locality (lots of remote reading of data). This ruins performance.
- There is no scalable system for implementing arbitrary graph algorithms over arbitrary graphs in a large-scale distributed environment.

How would you solve this?

- Use a custom distributed environment.
  - Not general purpose.
- Use existing distributed platform like MapReduce
  - Sub-optimal and inefficient. Often graph algorithms are iterative.
- Use existing parallel graph system like Parallel BGL or CGMgraph.
  - None currently address fault tolerance and other important issues like internal cluster communication.

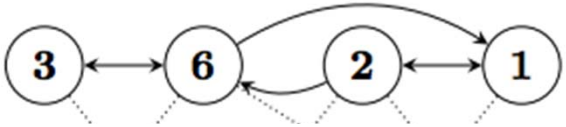
Pregel aims to solve this problem

- Allows efficient processing of large, distributively-stored, graphs.
- Abstracts away distributed computed related issues like fault tolerance and machine communication.
- Outlines a ‘vertex-centric system’
  - **All programmer needs to do is outline a single function.**

Pregel’s computation model

- Consists of a sequence of iterations (supersteps), where the **same user-defined function** is executed for each vertex.
- This function specifies behaviour at a single vertex V and superset S. It can read messages sent to the vertex in superstep S-1, send messages to other vertices that will be read in superset S+1, and modify that state of V and its outgoing edges.
- Initially each vertex is in an active state. Each vertex can **‘vote to halt’**, where it runs no further computation in any further superstep unless it receives a message from another vertex. It is then is reactivated again and needs to explicitly vote to halt to deactivate itself again. **Algorithm terminates when all vertices have halted.**

Pregel simple example - find maximum value



Superstep 0

Pregel details - the master, combiners, aggregators and fault tolerance.

- Master node
  - Coordinates and maintains a list of all workers.
  - Maintains aggregator.
- Aggregators
  - Nodes send master a value at each iteration for aggregation.
  - Provides a global statistic to each node at each superstep, important for some algorithms like Dijkstra's algorithm.
- Combiners
  - Combines messages to reduce message traffic.
- Fault tolerance.
  - Achieved through checkpointing
  - Master instructs workers to save their state to persistent storage at the beginning of each superstep.
  - If Master detects these workers as down, it reassigns their partitions to available workers and recomputes the superstep.

Pregel more complicated 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();
  }
};
```

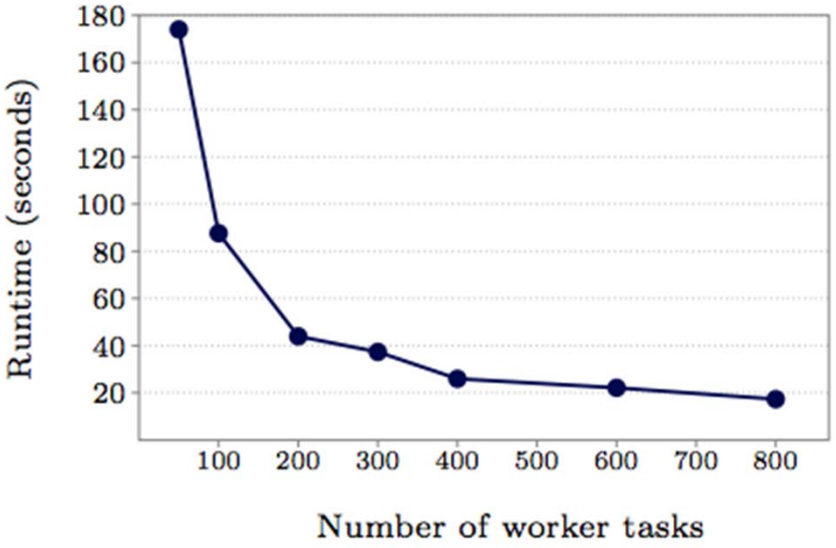
Set value to 0 if the vertex is the source and INF otherwise.

Compute all the potential min distances from incoming arcs.

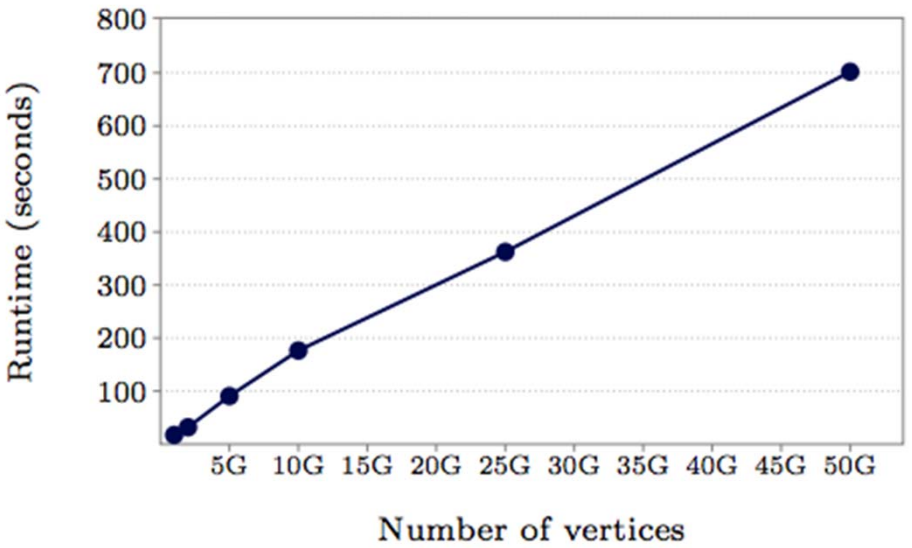
If this is less than the current min distance, alert neighbours through outgoing arcs

Halt until another message is sent to me.

Experiments - SSSP with varying graph size and worker numbers



The 16x increase in worker tasks from 50 to 800 leads to a 10-fold speedup.



Run time varies linearly with increasing number of vertices. (As it should)

Pregel critical analysis

- It uses network transfers only for messages. Omitting the need to read from remote memory.
    - This is conventionally why algorithms over distributively stored graphs are slow.
- Authors claim all graph algorithms can be transformed into this vertex-centric approach.
    - However no proof is presented.
- If fault tolerance occurs, it’s not clear whether only the work for the reassigned graph partition, or the entire work for that superstep is recomputed.
- Doesn’t address when infinite loops might occur and how to account for them.

Questions/Discussion