State of the art and its limitations

BGL, LEDA, NetworkX, JDSL, Stanford GraphBase, FGL
- Single-computer graphs => not distributed

Parallel BGL, CGMgraph
- Distributed but do not address fault tolerance

MapReduce
- Can be used to mine large Graphs but leads to suboptimal performance

SQL
- Not ideal for graph algorithms
An ideal implementation of algorithm to process large graph would be:

- Distributed
- Fault-Tolerant
- Algorithm-Flexible
- Scalable

**Pregel:**

- Inspired by Valiant's Bulk Synchronous Parallel Model [1]
- Sequence of iterations called super steps
- Voting system for nodes to halt
- C++ implementation
How does it work?
- Assigns unique IDs
- Partitions based on the modulo N value of the ID
- Maintains a list of alive workers and their value

Diagram:

- Partition 1
  - Node 1
  - Node 3
  - Node 6

- Partition 2
  - Node 1
  - Node 2
  - Node 3

- Edge-cutting

- Master

Figure showing a network of nodes and partitions with connections and labels for edge-cutting, master, and partitions.
Superstep 2

1  6
3  6
4  6

Superstep 3

1  6
3  6
4  6

Biggest number is 6!

Send Algorithm

0 Active, [6, 6]

0 Active, [6, 6]
How strong is the problem solving?

- PageRank
- Shortest Paths
- Bipartite Matching (Marriage Problem)
- Semi-Clustering
Partitioning graphs is a **hard** problem.

Step runtime: 6 sec.
Limitations & Discussion & Criticism

- Partitioning graphs is a **hard problem**
  - And maybe modulo N of ID is not the best solution
  - Maybe we should take into consideration the **structure of the graph**

Step runtime: 1 ms.
Limitations & Discussion & Criticism

• Is bulk-synchronous computation a good solution?
  • "Natural Graphs" have generally weird structure with some nodes being more popular than others.
  • Shouldn't "celebrity" nodes have more priority? Is there a heuristic for that?
References


Questions?
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.
class PageRankVertex
    : public Vertex<double, void, double> {
public:
    virtual void Compute(MessageIterator* msgs) {
        if (superstep() >= 1) {
            double sum = 0;
            for (; !msgs->Done(); msgs->Next())
                sum += msgs->Value();
            *MutableValue() =
                0.15 / NumVertices() + 0.85 * sum;
        }
        if (superstep() < 30) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        } else {
            VoteToHalt();
        }
    }
};

Figure 4: PageRank implemented in Pregel.