Pregel: A System for Large-Scale Graph Processing

G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, G. Czajkowski

Google, Inc.

Presenter: Mahsa Yarmohammadi

Motivation

- Many computing problems use large graphs
  - Web graphs
  - Social networks
  - Transportation routes
  - Citation relationships between publications
- Efficient processing is a challenge
- Pregel: a vertex-centric approach to distribute large graphs processing

Current Approaches

- No scalable general-purpose system for distributed processing of large graphs

- Four options:

1) A custom distributed system for each new algorithm.
2) MapReduce. sub-optimal performance and usage.
3) A single-machine library such as BGL, LEDA, JSDL, etc.
4) An existing parallel graph system library i.e. Parallel BGL or CGMgraph. No fault tolerance.

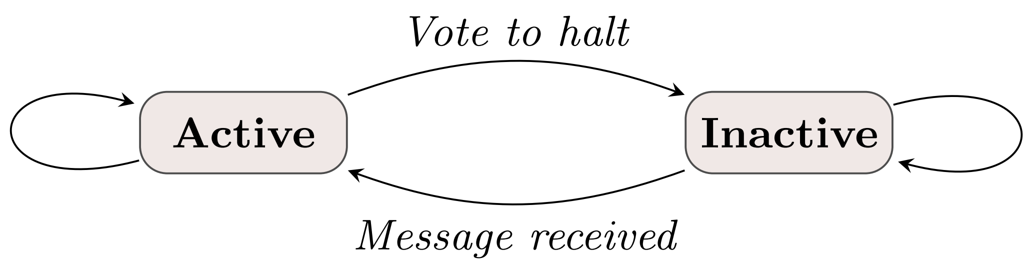
Pregel

- Pregel: scalable fault tolerance platform with API for arbitrary graph algorithms
- Pure message passing model
  - Network transfers are only messages
  - In MapReduce, network transfers are the entire state of the graph
- Similar to MapReduce in sense that
  - focusing on a local action
  - processing each item independently
  - composing the results of actions

Pregel

- A sequence of iterations, called *supersteps*
- Input: a directed graph
- At each superstep:
  - a user-defined function is ran for each vertex
  - read messages sent to V in superstep *S-1*
  - send messages to other vertices to receive at *S+1*
  - modify the state of V and its outgoing edges
- Output: a directed graph isomorphic to the input, or with added/removed vertices/edges

Model of Computation



- Superstep 0 : every vertex is active
- A vertex deactivates itself by voting to halt
- A vertex is reactivated if it recieves an external message
- Algorithm terminates if all vertices are inactive

Example

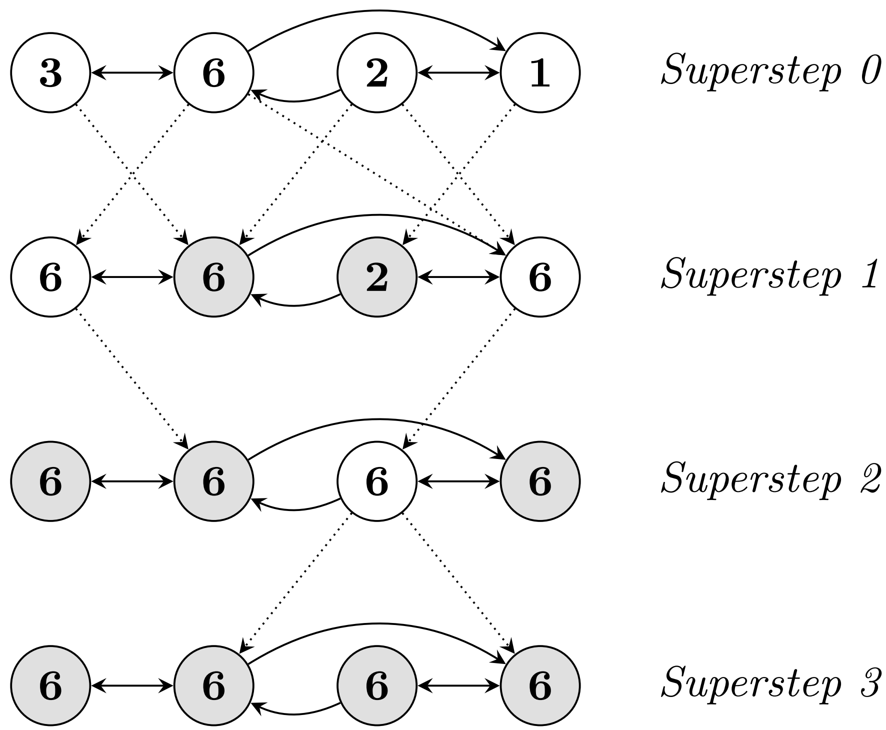


Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

The C++ API

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

Message Passing

- Vertices send messages to each other
  - destination vertex
  - message value
- A vertex can iterate over all received messages at *S* when **compute()** is called at *S+1*
- A vertex can iterate over its outgoing edges, send a message to the destinations vertex of each edge

Combiners

- Reduce **compute()** overhead
- Combine several messages intended for a vertex V into a single message
- For example, if V only needs *sum* of integer messages it receives
- Only work for commutative and associative operations

Aggregators

- A mechanism for global communication, monitoring, and data.
- Each V can provide a value to an aggregator at *S*, the sysmtem combines values and make it public to all Vs at *S+1*
- Examples: min, max, sum, other statistics
- Only work for commutative and associative operations

Topology Mutations

- When **compute()** sends a request to add/remove vertex/edge
- To resolve conflicting requests:
  1) partial ordering
  2) handler
- edge removal > vertex removal > vertex addition > vertex addition > **compute()**
- User-defined handlers for remaining conflicts

Input and Output

- Many possible file formats
  - text files
  - set of vertices in a database
  - rows in Bigtable

- Support for other formats by subclassing **Reader** and **Writer** classes.

Basic Architecture

- A master machine assigns graph partitions to worker machines
- Master instructs each worker to perform a superstep
- When the worker is finished, it tells the master how many vertices will be active for next superstep
- Repeat untill no active vertex is remained

Fault Tolerance

- Checkpointing procedure
- At the beginning of a superstep, a checkpoint is saved to persistent storage by each worker
  - log vetex/edges values, incoming messages
- Master sends “ping” messages to workers
- If no response after an interval, then failed
- Master reassigns partitions to other workers
- Repeat missing supersets after checkpoints

Fault Tolerance

- Checkpointing procedure
- At the beginning of a superstep, a checkpoint is saved to persistent storage by each worker
  - log vetex/edges values, incoming messages
- Master sends “ping” messages to workers
- If no response after an interval, then failed
- Master reassigns partitions to other workers
- Repeat missing supersets after checkpoints
- Confined recovery: log also outgoing messages, only recomputes lost partitions

Applications - PageRank

```
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();
    }
  }
};
```

Applications – Shortest Paths

```
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();
  }
};
```

```
class MinIntCombiner : public Combiner<int> {
  virtual void Combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};
```

Experiments

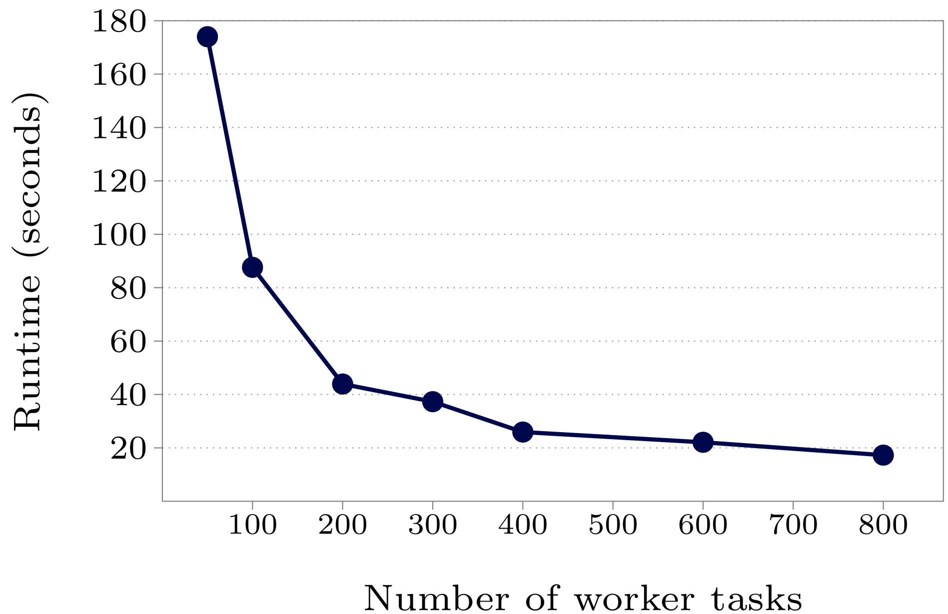


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multi-core machines

Experiments

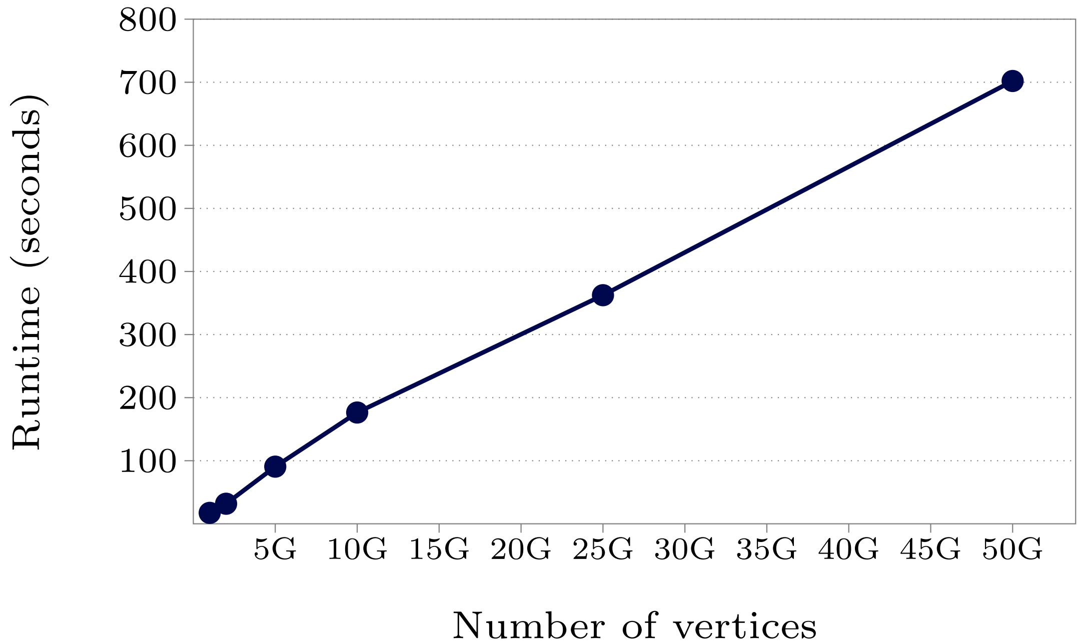


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Experiments

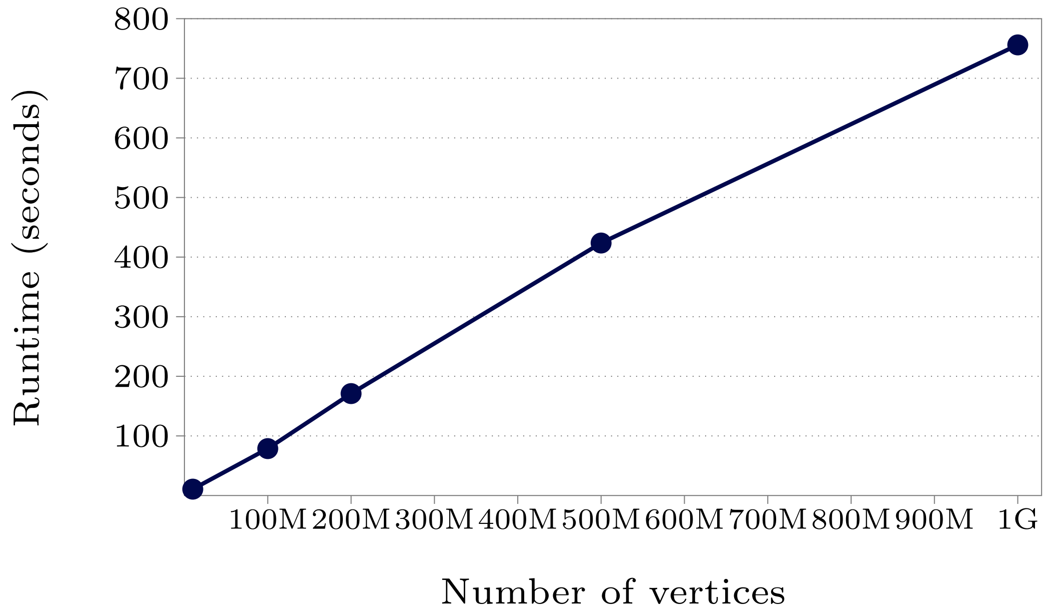


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 **billion edges in the largest case): varying graph sizes on** 800 **worker tasks scheduled on** 300 **multicore machines**

Conclusion and Future Work

- Pregel: a model suitable for large-scale graph computing
- High-quality, scalable and fault tolerant
- Dozens of Pregel applications have been deployed
- Spill some computation states to local disk instead of RAM
- Dynamic repartitioning