Pregel: A System for Large-Scale Graph Processing

Written by G. Malewicz et al. at SIGMOD 2010
Presented by Chris Bunch
Tuesday, October 12, 2010

Graphs are hard

- Poor locality of memory access
- Very little work per vertex
- Changing degree of parallelism
- **Running over many machines makes the problem worse**

State of the Art Today

- Write your own infrastructure
  - Substantial engineering effort
- Use MapReduce
  - Inefficient - must store graph state in each stage, too much communication between stages

State of the Art Today

- Use a single-computer graph library
  - Not scalable ☹
- Use existing parallel graph systems
  - No fault tolerance ☹

Bulk Synchronous Parallel

- Series of iterations (supersteps)
- Each vertex invokes a function in parallel
- Can read messages sent in previous superstep
- Can send messages, to be read at the next superstep
- Can modify state of outgoing edges

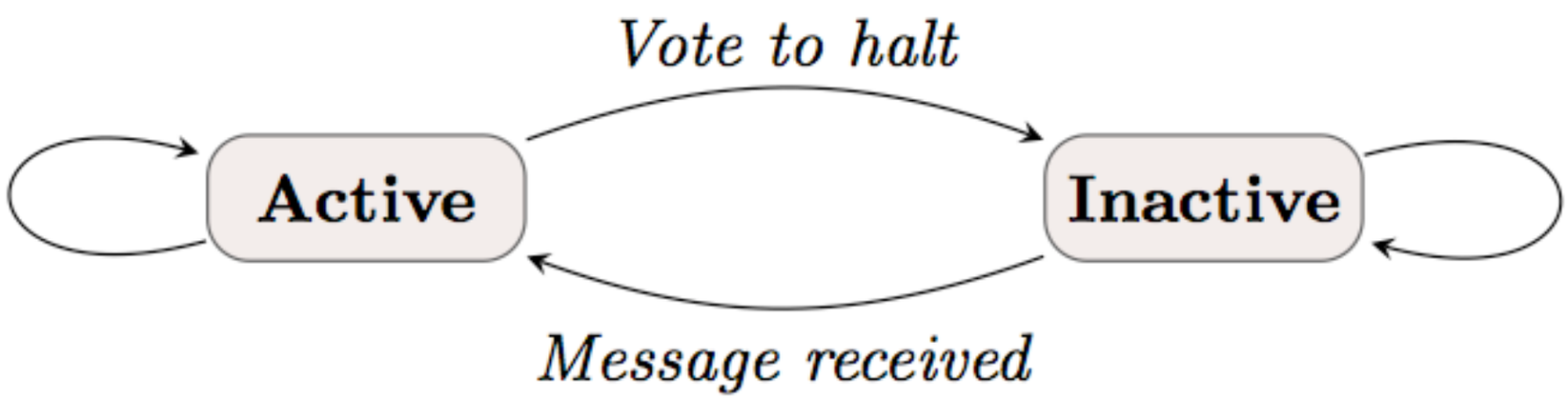
Compute Model

- You give Pregel a directed graph
- It runs your computation at each vertex
- Do this until every vertex votes to halt
- Pregel gives you a directed graph back

Primitives

- Vertices - first class
- Edges - not
- Both can be dynamically created and destroyed

Vertex State Machine



C++ API

- Your code subclasses Vertex, writes a Compute method
- Can get/set vertex value
- Can get/set outgoing edges values
- Can send/receive messages

C++ API

- Message passing:
- No guaranteed message delivery order
- Messages are delivered exactly once
- Can send messages to any node
- If dest doesn't exist, user's function is called

C++ API

- Combiners (off by default):
- User specifies a way to reduce many messages into one value (ala Reduce in MR)
- Must be commutative and associative
- Exceedingly useful in certain contexts (e.g., 4x speedup on shortest-path compuation)

C++ API

- Aggregators:
- User specifies a function
  - Each vertex sends it a value
  - Each vertex receives aggregate(vals)
  - Can be used for statistics or coordination

C++ API

- Topology mutations:
- Vertices can create / destroy vertices at will
- Resolving conflicting requests:
  - Partial ordering: E Remove, V Remove, V Add, E Add
  - User-defined handlers: You fix the conflicts on your own

C++ API

- Input and output:
  - Text file
  - Vertices in a relational DB
  - Rows in BigTable
  - Custom - subclass Reader/Writer classes

Implementation

- Executable is copied to many machines
- One machine becomes the Master
  - Coordinates activities
- Other machines become Workers
  - Performs computation

Implementation

- Master partitions the graph
- Master partitions the input
  - If a Worker receives input that is not for their vertices, they pass it along
- Supersteps begin
- Master can tell Workers to save graphs

Fault Tolerance

- At each superstep S:
  - Workers checkpoint V, E, and Messages
  - Master checkpoints Aggregators
- If a node fails, everyone starts over at S
- Confined recovery is under development
- **what happens if the Master fails?**

The Worker

- Keeps graph in memory
- Message queues for supersteps S and S+1
- Remote messages are buffered
- Combiner is used when messages are sent or received (save network and disk)

The Master

- Master keeps track of which Workers own each partition
  - Not who owns each Vertex
- Coordinates all operations (via barriers)
- Maintains statistics and runs a HTTP server for users to view info on

Aggregators

- Worker passes values to its aggregator
- Aggregator uses tree structure to reduce vals w/ other aggregators
  - Better parallelism than chain pipelining
- Final value is sent to Master

PageRank in Pregel

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

Shortest Path in Pregel

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

Evaluation

- 300 multicore commodity PCs used
- Only running time is counted
  - Checkpointing disabled
- Measures scalability of Worker tasks
- Measures scalability w.r.t. # of Vertices
  - in binary trees and log-normal trees

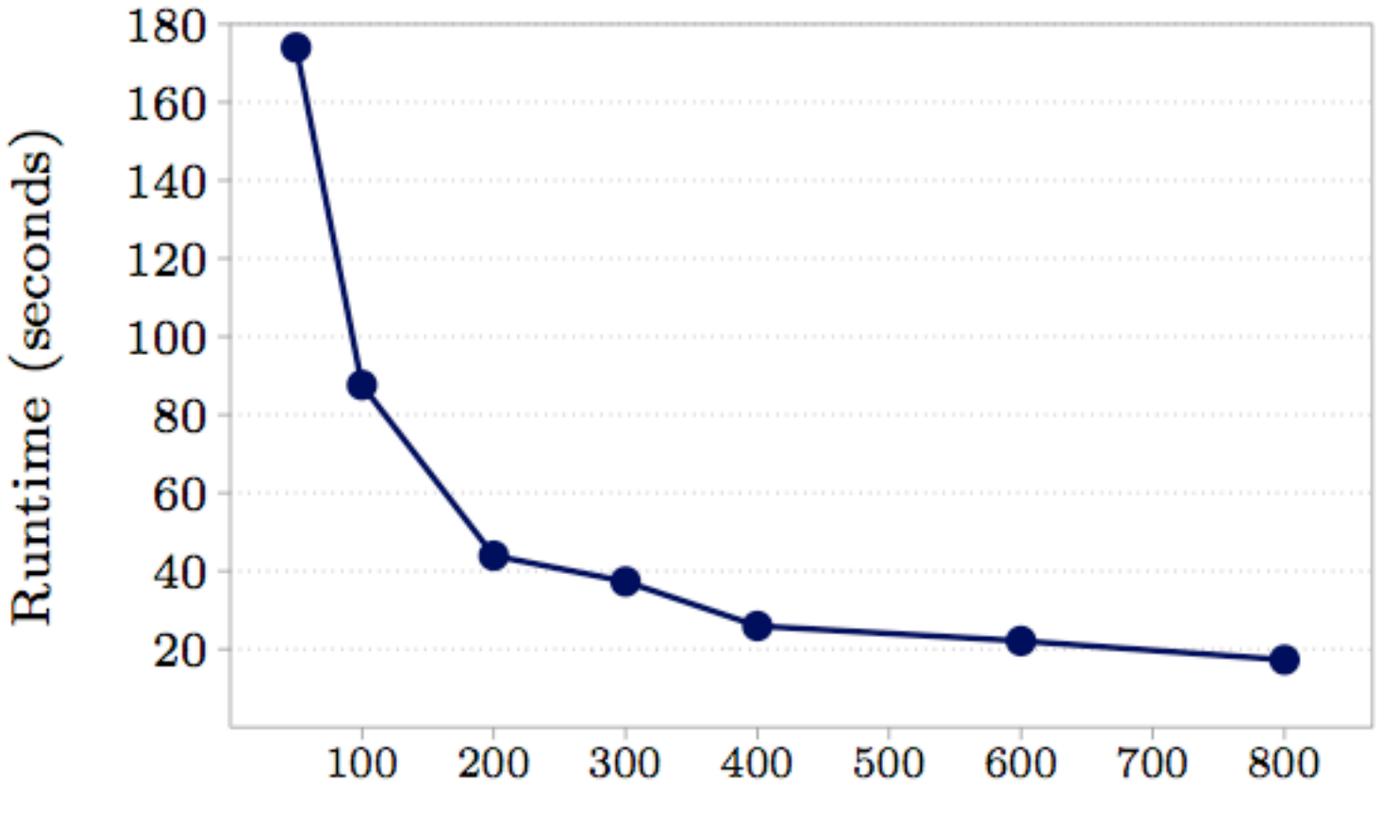


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multi-core machines

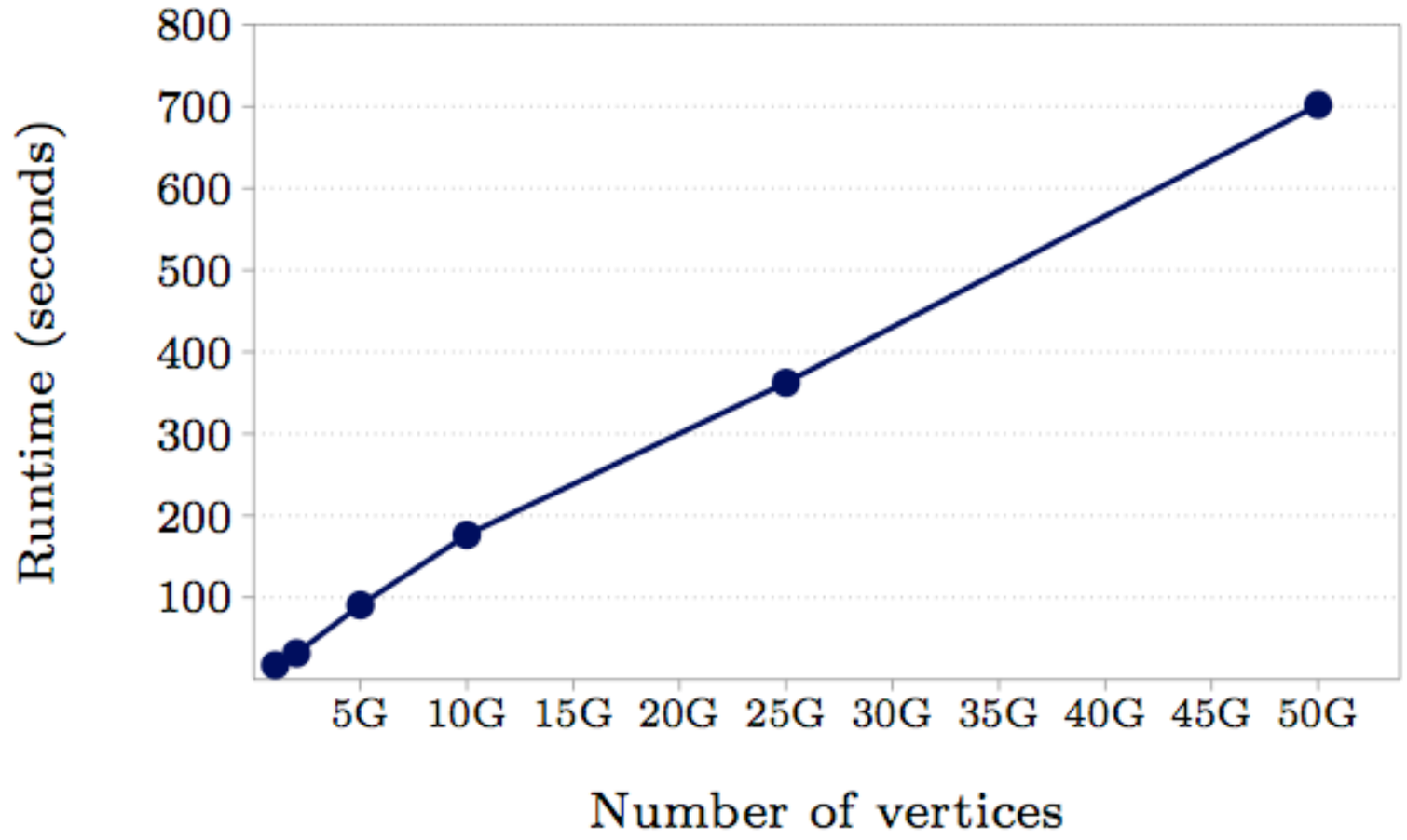


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

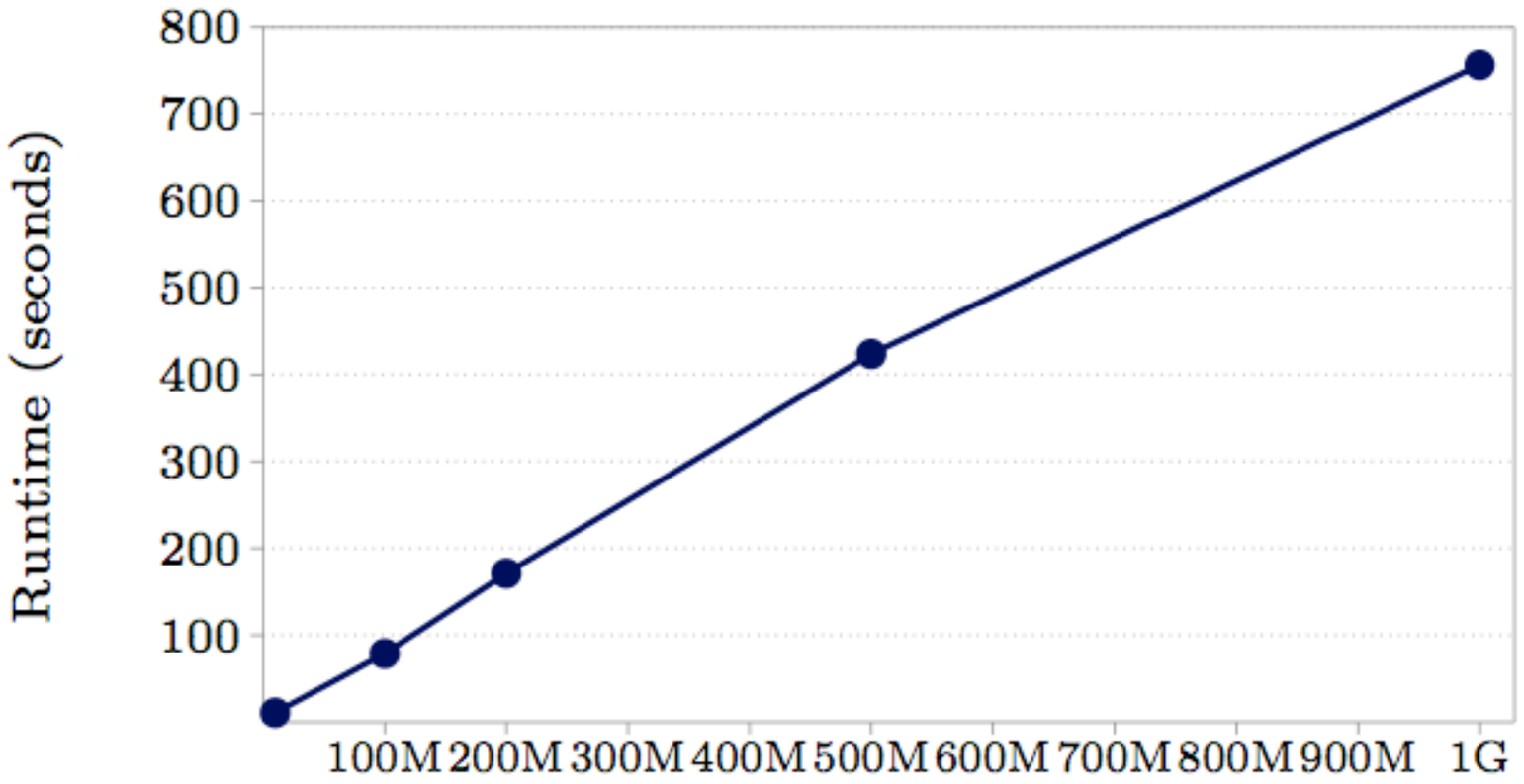


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Current / Future Work

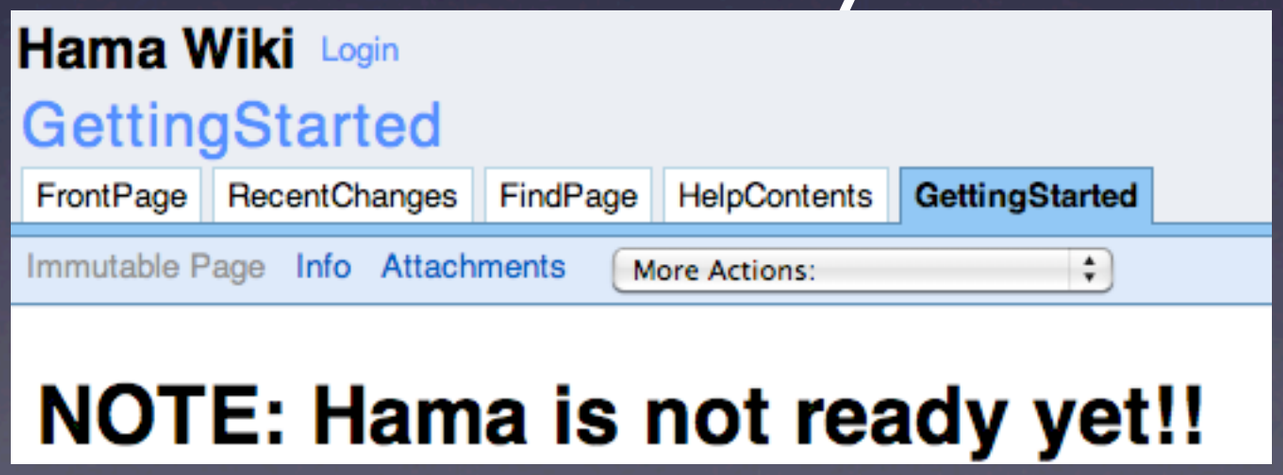
- Graph must fit in RAM - working on spilling over to / from disk
- Assigning vertices to machines to optimize traffic is an open problem
  - Want to investigate dynamic re-partitioning

Conclusions

- Pregel is production-ready and in use
- Usable after a short learning curve
- Vertex centric is not always easy to do
- Pregel works best on sparse graphs w / communication over edges
- Can't change the API - too many people using it!

Related Work

- Hama - from the Apache Hadoop team
- BSP model but not vertex centric ala Pregel
- Appears not to be ready for real use:
-



Related Work

- Phoebus, released last week on github
- Runs on Mac OS X
- Cons (as of this writing):
  - Doesn't work on Linux
  - Must write code in Erlang (since Phoebus is written in it)

Thanks!

- To my advisor, Chandra Krintz
- To Google for this paper
- To all of you for coming!