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

Presented By
Riyad Parvez

Real World Graph Processing

- Web graph:
  - PageRank (influential vertices)
- Social graph:
  - Popularity rank, personalized rank, shortest paths, shared connections, clustering (communities), propagation
- Advertisement:
  - Target ads
- Communication network:
  - Maximum flow, transportation routes
- Biology network
  - protein interactions
- Pathology network
  - find anomalies

Graph Processing is Different

- Poor locality of memory access.
- Very little work done per vertex.
- Changes degree of parallelism over the course of execution.

Why not MapReduce (MR)

- Graph algorithms can be expressed as series of MR jobs.
- Data must be reloaded and reprocessed at each iteration, wasting I/O, network. bandwidth, and processor resources.
- Needs an extra MR job for each iteration just to detect termination condition.
- MR isn't very good at dynamic dependency graph.

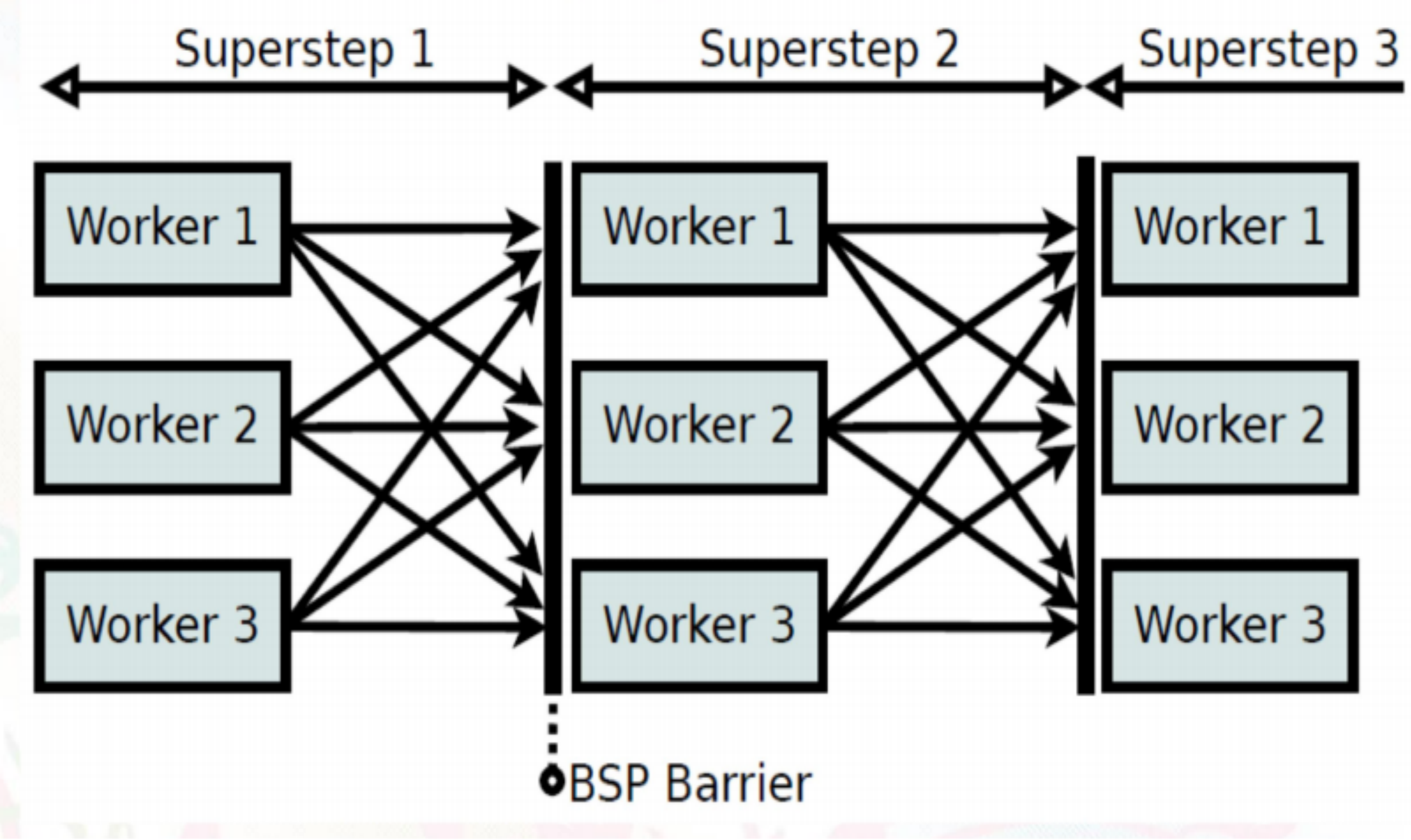
Pregel

- Developed at Google
- Modeled after Bulk Synchronous Parallel (BSP) computing model
- Distributed message passing system
- Computes in vertex-centric fashion
  - "*Think like a vertex*"
- Scalable and fault tolerant
- Influenced systems like Apache Giraph and other BSP distributed systems

Bulk Synchronous Parallel

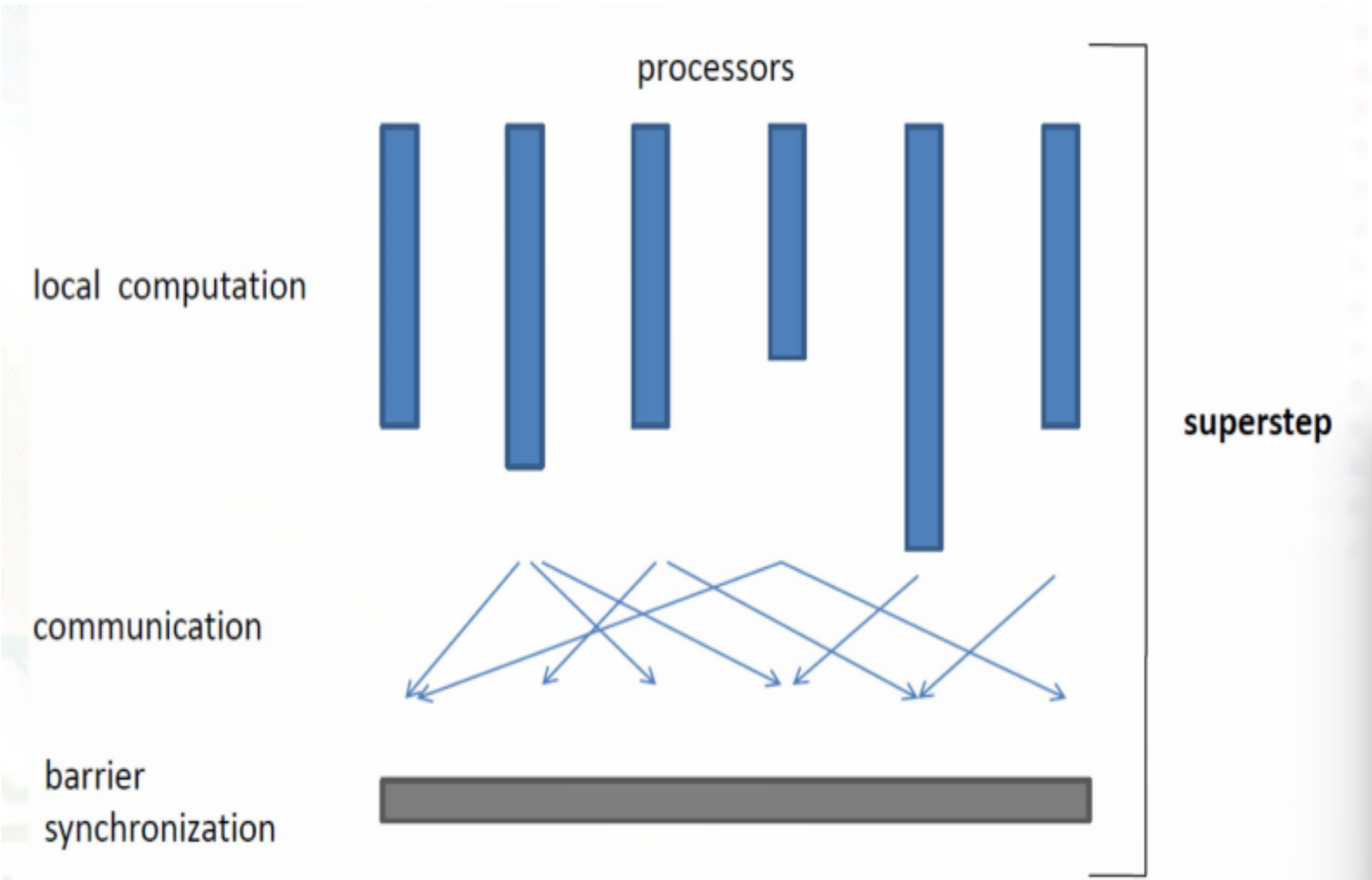
- Leslie Valiant introduced in 1990.
- Computations are consist of a sequence of iterations, called *superstep*.
- During superstep, framework calls user-defined *computation* function on every vertex.
- Computation function specifies behaviour at a single vertex *V* and a single superstep *S*.
- Supersteps end with barrier synchronization.
- All communications are from superstep *S* to superstep *S+1*.
- Performance of the model is predictable.

Bulk Synchronous Parallel



Source: https://cs.uwaterloo.ca/~kdaudjee/courses/cs848/slides/jenny.pdf

Bulk Synchronous Parallel



Source: https://cs.uwaterloo.ca/~kdaudjee/courses/cs848/slides/jenny.pdf

Pregel Computation Model

- Computation on locally stored data.
- Computations are in-memory.
- Terminates when all vertices are inactive or no messages to be delivered.
- Vertices are distributed among workers using *hash(ID) mod N*, where N is the number of partitions (default partitioning)
- Barrier synchronization
  - Wait and synchronize before the end of superstep
  - Fast processors can be delayed by slow ones
- Persistent data is stored on a distributed storage system (GFS/BigTable)
- Temporary data is stored in disk.

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();
};
```

Figure 3: The Vertex API foundations.

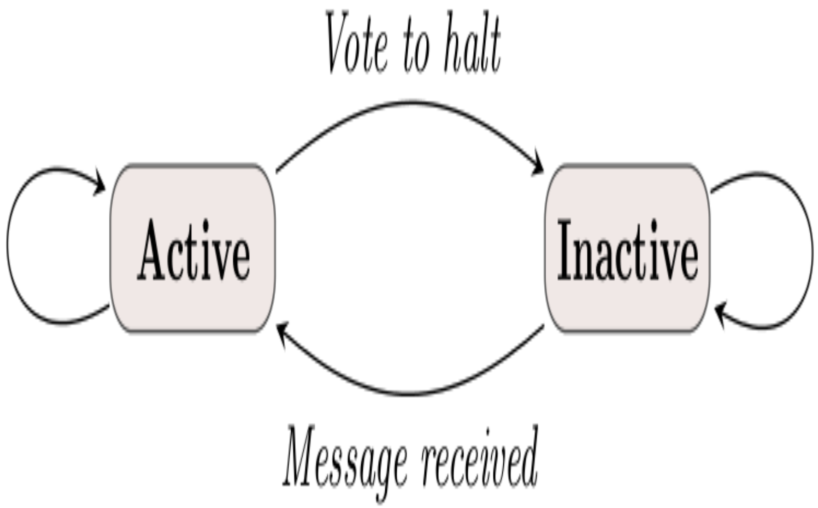
Source: http://kowshik.github.io/JPregel/pregel_paper.pdf

Vertex

- Can mutate local value and value on outgoing edges.
- Can send arbitrary number of messages to any other vertices.
- Receive messages from previous superstep.
- Can mutate local graph topology.
- All active vertices participate in the computation in a superstep.

Vertex State Machine

- Initially, every vertices are active.
- A vertice can deactivate itself by *vote to halt*.
- Deactivated vertices don't participate in computation.
- Vertices are reactivated upon receiving message.



Example

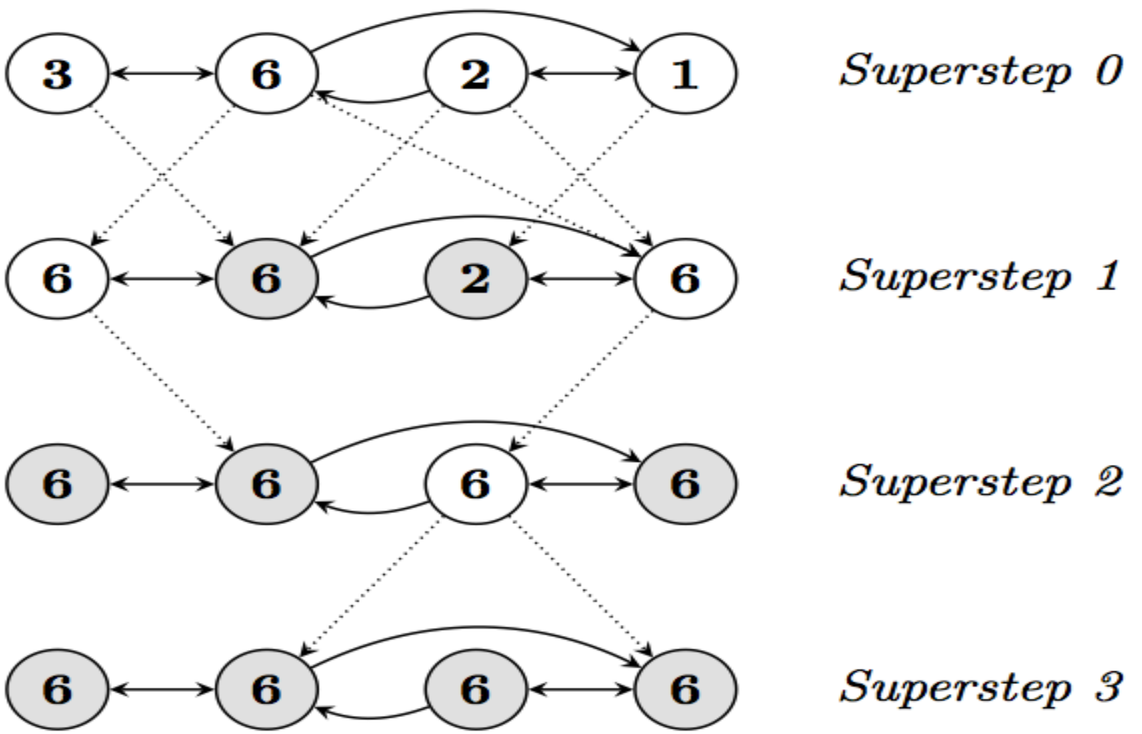


Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

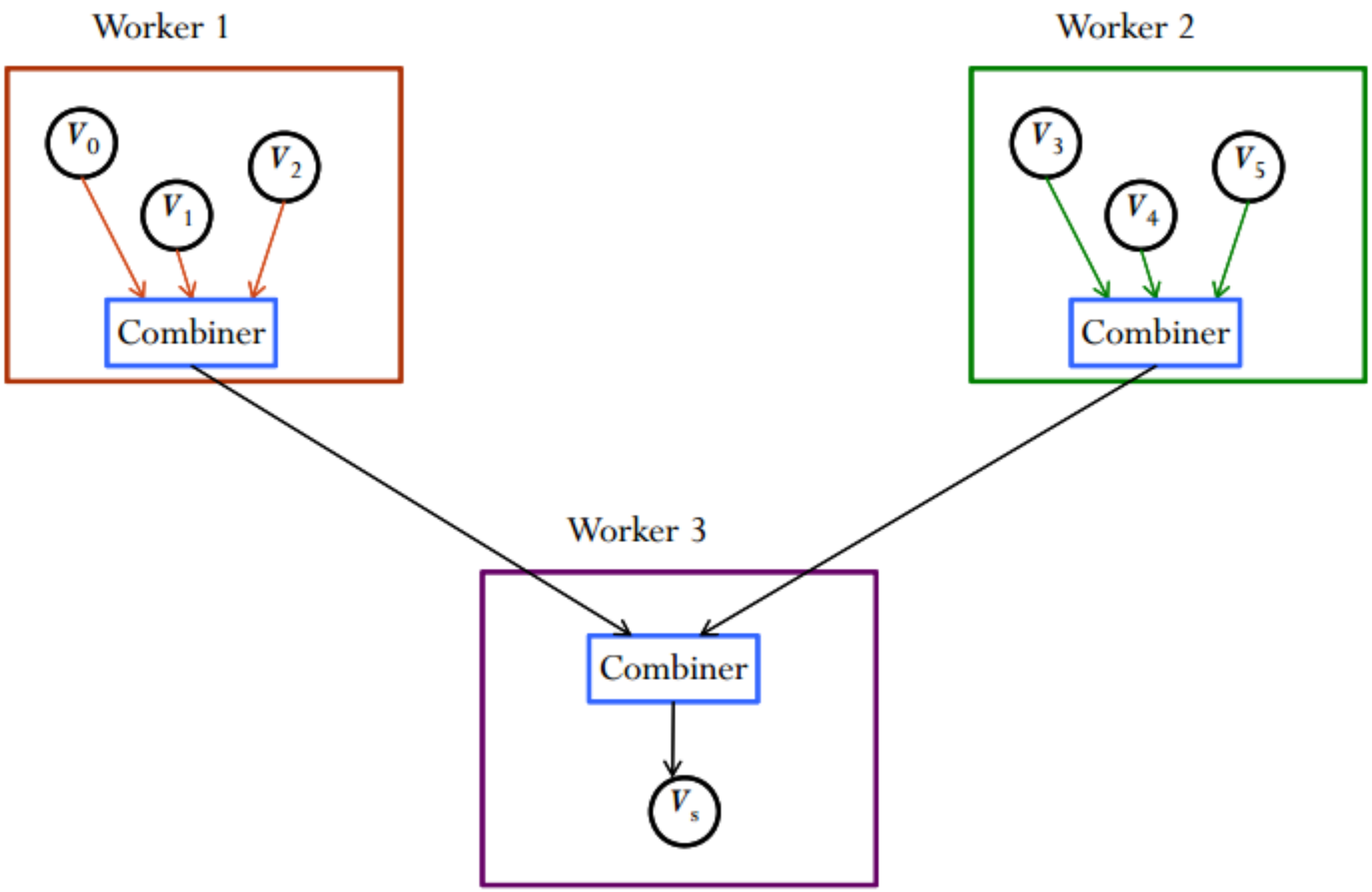
Messages

- Consists of a message value and destination vertex.
- Typically sent along outgoing edges.
- Can be sent to any vertex whose identifier is known.
- Are only available to receiver at the beginning of superstep.
- Guaranteed to be delivered.
- Guaranteed not to be duplicated.
- Can be out of order.

Combiner

- Sending messages incurs overhead.
- System calls *Combine()* for several messages intended for a vertex *V* into a single message containing the combined message.
- No guarantees which messages will be combined or the order of combination.
- Should be enabled for commutative and associative messages.
- Not enabled by default.

Combiner

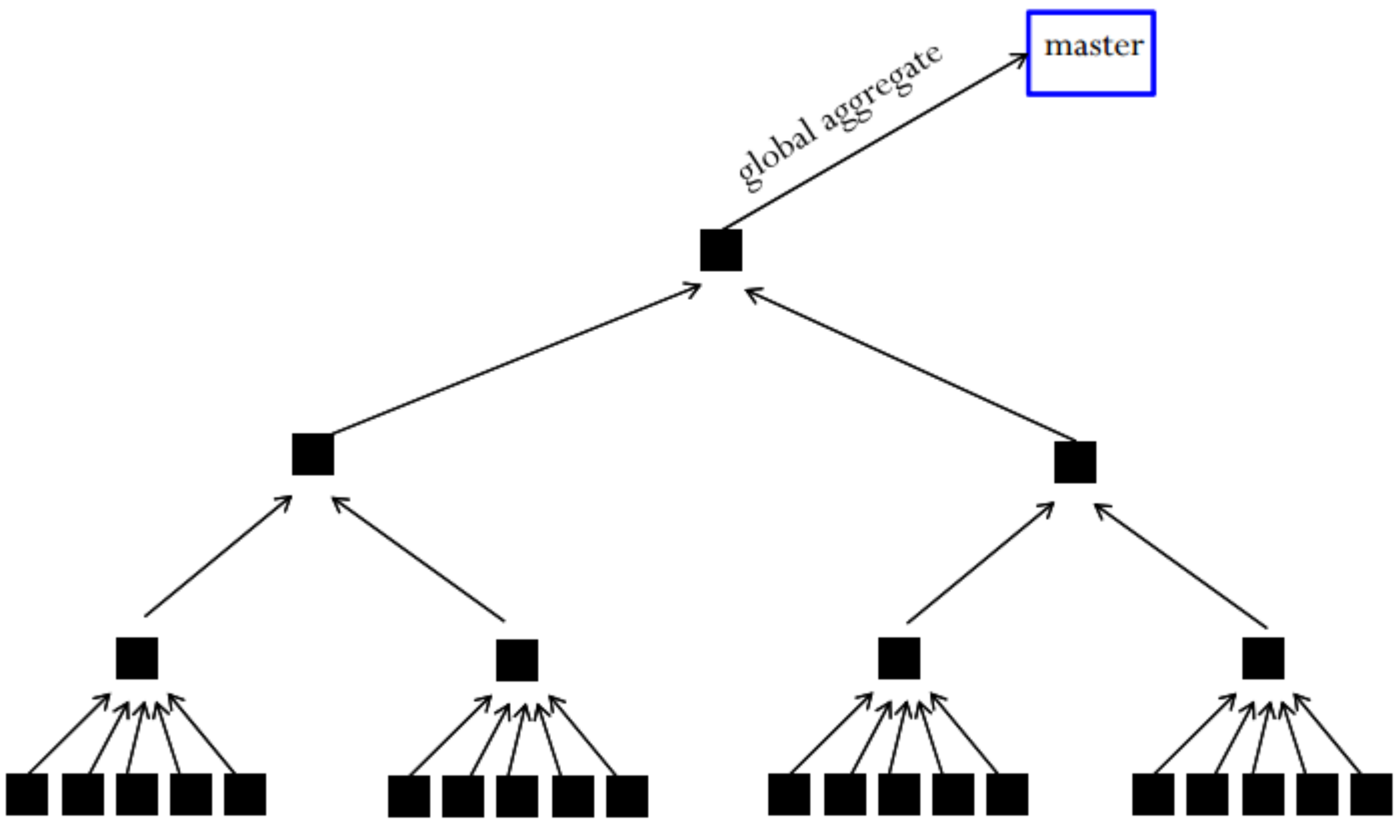


Source: https://wiki.engr.illinois.edu/download/attachments/188588798/pregel.pdf?version=1

Aggregator

- Mechanism for global communication, monitoring and global state.
  - Vertices provide value to aggregator in superstep *S*.
  - Values are combined using a reduction operator.
  - Resulting value is available to all vertices at superstep *S+1*.
- New aggregator is defined by subclassing "*Aggregator*" class.
- Reduction operator should be associative and commutative.

Reduction (Aggregator)



Source: https://wiki.engr.illinois.edu/download/attachments/188588798/pregel.pdf?version=1

Topology Mutation

- Vertices can dynamically create/destroy vertices, edges.
- Mutations and conflict resolution take place at barrier.
- Except local mutation (self-edge) immediately takes place.
- Order of mutations
  - Edge deletion
  - Vertex deletion
  - Vertex addition
  - Edge addition

Master

- Partitions the input and assigns one or more partitions to each worker.
- Keeps list of
    - All alive workers
    - Worker's unique identifiers
    - Addressing informations
    - Partition of the graph is assigned to the worker.
- Coordinates barrier synchronization i.e., superstep.
- Fault tolerance by checkpoint, failure detection and reassignment.
- Maintains statistics of the progress of computation and the state of the graph.
- Doesn’t participate in computation.
- Not responsible for load-balancing.

Worker

- Responsible for computation of assigned vertices.
- Keeps two copies of active vertices and incoming messages
  - Current superstep
  - Next superstep
- Place *local messages* immediately in message queue.
- Buffer *remote messages*.
  - Flush asynchronously in single message if threshold reached.

Fault Tolerance

- Checkpoint at the beginning of superstep.
  - Master saves aggregators.
  - Workers save vertices, edges and incoming messages.
- Worker failure detected by ping messages.
- Recovery
  - Master reassigns failed worker partition to other available workers.
  - All workers restart from superstep *S* by loading state from the most recently available checkpoint.
- *Confined recovery*: recovery is only confined to lost partitions
  - Workers also save outgoing messages.
  - Recomputes using logged messages from healthy partitions and recalculated ones from recovering partitions.

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();
    }
  }
};
```

Figure 4: PageRank implemented in Pregel.

Source: http://kowshik.github.io/JPregel/pregel_paper.pdf

Performance

Experimental Setup:

- Hardware: A cluster of 300 multi-core commodity PCs.
- Algorithm: SSSP with unit weight edges.
    - All-pairs shortest paths impractical b/c O(|V|2) storage.
- Measures scalability w.r.t. both the number of workers and the number of vertices.
- Data collected for:
    - Binary trees (to test scalability).
    - log-normal random graphs (to study performance in a realistic setting).
- No checkpointing.

Performance

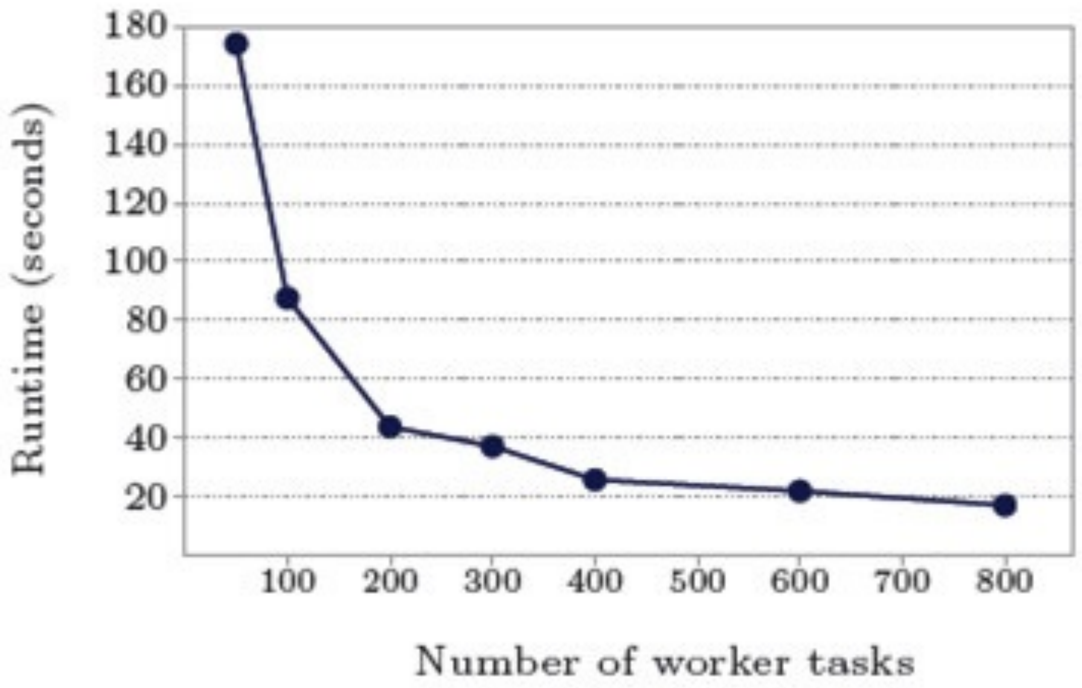


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multicore machines

Source: http://kowshik.github.io/JPregel/pregel_paper.pdf

Performance

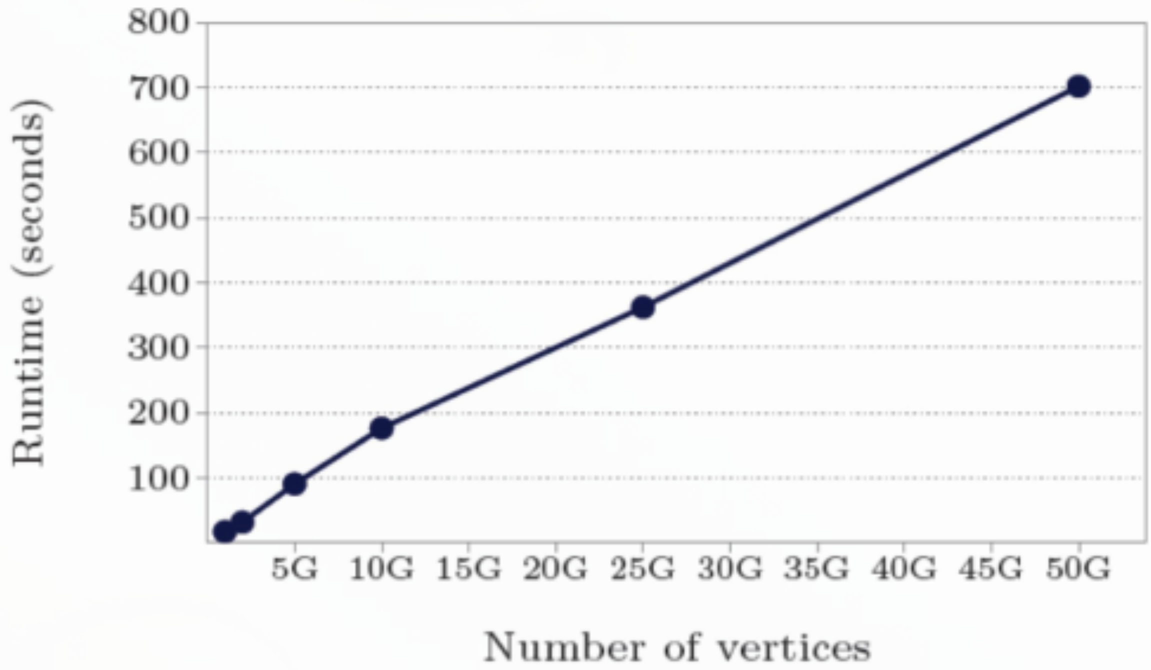


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Source: http://kowshik.github.io/JPregel/pregel_paper.pdf

Performance

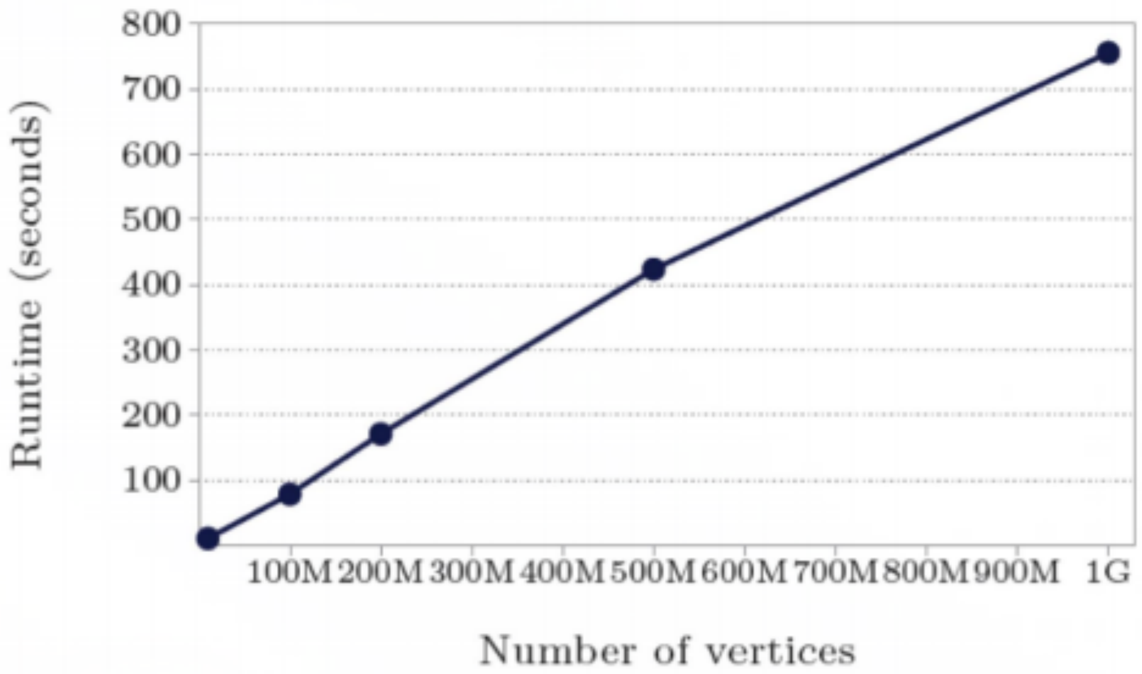


Figure 9: SSSP—log-normal random graphs, mean out-degree 127.1 (thus over 127 billion edges in the largest case): varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Source: http://kowshik.github.io/JPregel/pregel_paper.pdf

Summary

- Distributed system for large scale graph processing.
- Vertex-centric BSP model
  - Message passing API
  - A sequence of supersteps
  - Barrier synchronization
- Coarse grained parallelism
- Fault tolerance by checkpointing
- Runtime performance scales near linearly to the size of the graph (CPU bound)

Discussion

- No fault tolerance for master is mentioned in the paper (Probably Paxos or replication).
- Static partitioning! What happens if a worker is too slow?
- Dynamic partitioning, network overhead for reassigning vertices and state.
- Good for sparse graph. But communication overhead for dense graph can bring the system down to knees.