Pregel: A System for Large-Scale Graph Processing

Ashvin Goel
Electrical and Computer Engineering
University of Toronto
ECE1724

Authors: Grzegorz Malewicz, Matthew Austern, Aart Bik, James Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski (Google, Inc.)

Some slides adapted from Aishwarya G, Subhasish Saha
What is Pregel?

- Scalable and fault-tolerant graph processing framework
- Provides flexible API for expressing arbitrary graph algorithms
  - Vertex-centric computation model (think like a vertex)
  - Bulk Synchronous Parallel (BSP) message-passing model for communication and synchronization
BSP Model

- In BSP, computation is a sequence of supersteps
- In each superstep:
  - Each process reads input messages, executes independently, and sends messages for other processes
  - On completion, it waits for all other processes to complete
  - All messages are delivered at the start of the next superstep

![Diagram of BSP model with processes and supersteps](image)
Pregel Computation Model

• Programmer writes a user-defined function that operates on a vertex (think like a vertex)
  • Similar to map-reduce, which operate on a single key

• Vertex state:
  Vertex ID
  Current value
  List of outgoing edges and their values
  A queue containing incoming message
  A flag to determine if vertex is active
Pregel Computation Model

• Each vertex:
  • Receives messages sent in the previous superstep
  • Executes the user-defined function
  • May modify its state or state of outgoing edges
  • May send messages to outgoing edge vertices
    • These messages are received at the start of the next superstep
  • May mutate the topology of the graph (e.g., add edge)
  • Votes to halt if it has no further work to do

• Program termination:
  • When all vertices are inactive, and no messages in transit
Pregel API

- Programmer subclasses Vertex class

```cpp
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();
};
```
Example: Parallel SSSP in Pregel

Example taken from talk by Taewhi Lee, 2010
Example: Parallel SSSP in Pregel

- **Inactive Vertex**
- **Active Vertex**
- **Edge weight**
- **Message**
Example: Parallel SSSP in Pregel
Example: Parallel SSSP in Pregel
Example: Parallel SSSP in Pregel
Example: Parallel SSSP in Pregel
Example: Parallel SSSP in Pregel
Example: Parallel SSSP in Pregel
Example: Parallel SSSP 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();
    }
Pregel Architecture

- Pregel uses a master/worker model
  - Master coordinates workers, handles worker failures
  - Workers process their tasks, communicate with other workers asynchronously (computation and communication overlap)

Graph data stored persistently in GFS or BigTable
Pregel Execution

- Master decides the number of graph partitions and assigns one or more partitions to each worker
  - A vertex is deterministically mapped to a partition based on ID
  - So, all workers know the partition to which any vertex belongs

- Workers load input graph data in parallel

- Each worker initializes its vertices, marks them active

- Each worker executes compute() on all active vertices in a loop, using a separate thread per partition
Combiners

- A worker can combine messages sent to a given vertex
  - Requires combiner() to be commutative and associative
  - Reduces message traffic and disk space on the receiver side
- E.g., for SSSP, say v0-v5 send a message to v6

\[ a = \min(v0, v1, v2) \]
\[ b = \min(v3, v4, v5) \]

\[ \min(a, b) \]
Aggregator

- Used for global communication, and synchronization
  - E.g., compute aggregate statistics from vertex-reported values
- During a superstep:
  - Each worker aggregates values from its vertices to form a **partially aggregated value**
  - At the end of superstep, partially aggregated values from each worker are aggregated into a **global aggregate**
  - Global aggregate is sent to the master
- Master sends global aggregate values to all workers at the beginning of next superstep
Topology Mutations

• Needed for clustering applications
  • Output is a smaller graph

• Problem is that mutations may race and conflict
  • Two requests to add vertex V with different values

• Solution: apply the mutations at start of next superstep, in order:
  • Remove edges, then vertices
  • Add vertices, then edges

• Resolve rest of the conflicts with user-defined handlers
Pregel Fault Tolerance

• Uses checkpointing for failure recovery
  • The master periodically instructs workers to save the state of their partitions to persistent storage
    • Partition state includes vertex values, edge values, incoming messages

• Failure detection
  • Master uses regular ping messages

• Failure recovery
  • The master reassigns graph partitions to the currently available workers
  • All workers reload their partition state from most recent available checkpoint
Evaluation

SSSP on a 1 billion vertex binary tree
Evaluation

SSSP on log-normal graph (mean out-degree is 127.1) with 800 workers
Conclusions

• “Think like a vertex” computation model

• Combiners, aggregator, topology mutations enable many graph algorithms to be run on Pregel

• Highly influential
  • Apache Giraph builds on Pregel design
  • Facebook made improvements, used it on its trillion-edge social graph (look for: scaling apache giraph to a trillion edges)
Discussion
Q1

- We have discussed it briefly but let’s reconsider why Map-Reduce is not a good fit for graph processing?
Q2

• Why must the combiner() function be commutative and associative?
Q3

- Worker processing in each superstep is shown below:
  1. Receive incoming messages
  2. Persist incoming messages, graph state (vertex, edge values)
  3. Compute, modify vertex and outgoing edge state
  4. Buffer outgoing messages
  5. Barrier

- What guarantees are provided by Pregel's processing model (and how)? Why are these guarantees useful?