Pregel: A System for Large-Scale Graph Processing

Presenter: Jinliang Wei

CMU CSD

November 13, 2013
Why Pregel?

Need to process large-scale graphs.

- Option 1: Implement distributed infrastructure per algorithm? Too much repeated effort
- Option 2: Existing distributed computing platform: MapReduce? Parallel databases? Not suitable for graph processing
- Option 3: Single-computer graph algorithm library? Limited scale
- Option 4: Existing parallel graph systems? Fault tolerance and other issues
Main Features

- Vertex program
- Message passing
- Synchronous parallel
Example: PageRank
Example: PageRank as a vertex program

Iteratively run the following steps:

- Read messages from adjacent vertices (their ranks)
- Update my rank
- Send my rank to adjacent vertices
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();
};
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();
        }
    }
};
Message passing

- Message passing vs. Distributed Shared Memory (DSM)
- Communication between vertex programs is done via explicit message sending and receiving.
- Simple to implement
- No shared resource - no need for consistency model or concurrency control
- Disadvantages?
Bulk Synchronous Parallel

- A superstep: local computation + communication + synchronization barrier
- All vertex programs must reach the barrier before starting the next superstep.
- Messages sent won’t be seen by other until the next iteration.
Combiners

- Messages may be combined to reduce communication overhead.
- User-defined function to combine messages.
Aggregators

- Enables restricted global communication.
- Each vertex supplies a value. All values are combined by a reduction operator. The aggregated value is available for all vertices to read at the next iteration.
- Inherited by Distributed GraphLab.
Topology Mutation

- Add or remove vertices and edges.
- How to handle conflicts, e.g. two requests to add one vertex with different values?
- Partial ordering
  - Additions follow removals.
  - Edge removals before vertex removals.
- User-defined handler
Implementation

- The graph is partitioned and distributed among worker machines.
- Default is hash partitioning. Allows custom assignment function.
- The master instructs workers to perform a superstep.
- Workers run Compute() on each vertex.
Fault Tolerance

- Check pointing.
- Confined check pointing - under development.
  Only recover lost partition.
Evaluation
Look back from 2013

- The first widely-known distributed graph processing system.
- Influntial to many graph processing systems: Giraph, GraphLab, GraphChi...
Problems with message passing

- A vertex program must keep running to send out messages. Otherwise, its adjacent vertices won’t know that vertex’s value.
- In real applications, some vertices may converge earlier than others.
- Wasted CPU resource.
- What about DSM?
Other problems with Pregel

- BSP: well-known straggler problem
- Load balancing - power-law graph