CS 347
Parallel and Distributed Data Processing
Spring 2016

Notes 14: Distributed Graph Processing
Motivation

Many applications require graph processing
  E.g., PageRank

Some graph data sets are very large
  E.g., the Web

→ Need distributed graph processing

MapReduce is not a good fit for graph processing
Pregel

*Pregel: A System for Large-Scale Graph Processing.* G. Malewicz et al., SIGMOD 2010

Open-source version: Apache Giraph
Pregel, Königsberg (Pregolya, Kaliningrad)
Pregel

Distributed computational model/infrastructure for processing large graphs

E.g., PageRank on the web graph

\[
pr(x)_{i+1} = f \left( \frac{pr(a)_i}{out(a)}, \frac{pr(b)_i}{out(b)} \right)
\]
Pregel

\[ pr(x)_{i+1} = f \left( pr(a)_i / out(a), pr(b)_i / out(b) \right) \]

Synchronous computation based in iterations

In one iteration, each node
1. Gets messages from neighbors
2. Performs computation (updates its state)
3. Sends update messages to neighbors
Execution Flow

In MapReduce and Spark the execution flow is not a function of the data itself.

In Pregel the data (graph structure) drives the execution flow.
Execution Flow

Example

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

After each iteration, each vertex votes to halt or not. If *all* vertexes vote to halt, computation terminates.
Computation at a Node

Data available in iteration $i$:

- `outEdges`: `[[vertex, edgeWeight]]`
- `value`: ...
- `inputMessages`: `[[fromVertex, inputValue]]`
- `outputMessages`: `[[toVertex, outputValue]]`

`outEdges` and `value` are remembered for next iteration.
Computation at a Node

Example
Maximum value

change := False
for [fromVertex, inputValue] in inputMessages do
  if inputValue > value then
    value := inputValue
    change := True
if (iteration == 1) or change then
  for edge in outEdges do
    add [edge.toVertex, value] to outputMessages
else VoteToHalt()
Example: 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();
      SendMessageToAllNeighborsGetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
Example: 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();
        }
    }
};
Example: Single-Source Shortest Path

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();
    }
Example: Single-Source Shortest Path

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();
    }
};

Edge weight represents distance between neighbors
Architecture

Graph has nodes a, b, c, d...

Sample record: [a, value, edges]
Architecture

Partition graph and assign to workers
Architecture

Master

Read input data

Worker 1

Vertices
a, b, c

Worker 2

Vertices
d, e

Worker 3

Vertices
f, g, h

Input shard 1

Input shard 2

Worker 1 forwards input values to appropriate workers
Architecture

Master

Read input data

Worker 1

Vertices
a, b, c

Worker 2

Vertices
d, e

Worker 3

Vertices
f, g, h

Input shard 1

Input shard 2

CS 347

Notes 14

19
Run superstep 1
Architecture

Worker 1
- Vertices: a, b, c

Worker 2
- Vertices: d, e

Worker 3
- Vertices: f, g, h

Master

Worker 1 sends messages at the end of superstep 1

Halt?
Architecture

Run superstep 2

Master

Worker 1
- Vertices: a, b, c

Worker 2
- Vertices: d, e

Worker 3
- Vertices: f, g, h
Architecture

Master

Worker 1
Vertices
a, b, c

Worker 2
Vertices
d, e

Worker 3
Vertices
f, g, h

Checkpoint
Architecture

Each worker writes to stable storage its state, edges (in or out), and messages (incoming or outgoing)
If worker dies, find replacement and restart from latest checkpoint
Distribution Challenge

How to best to partition graph for efficient processing?
Limitations

Graph processing often part of larger flow
  Often need to join with unstructured/tabular data
  Spanning multiple systems
    Less opportunity for optimization
    Extra data movement

No granular fault tolerance
  Snapshot recovery only
GraphX

Graph processing on top of Spark

*GraphX: Graph Processing in a Distributed Dataflow Framework.* J.E. Gonzalez et al., OSDI 2014
GraphX

Graph representation
Vertex collection \{ [vertex, value] \}
Edge collection \{ [fromVertex, toVertex, weight] \}

Graph computation
Data flow of join and group by (punctuated by map operations)
1. Join vertex and edge collections to form triplets
   \[ \text{fromVertex, toVertex} \] \rightarrow \[ \text{fromValue, toValue, weight} \]
2. Group triplets by source or destination vertex
GraphX

Optimizations
Horizontal partitioning of collections by applying vertex cut
  Assign edges to machines that already have the vertices
    If multiple machines have the same vertex, pick less loaded
  Performs better than random edge partitioning of natural graphs
Efficient joins
Efficient materialized view management
  E.g., delta update propagation
Lineage for fault tolerance
Summary

Motivation
Pregel
  Computation model
  Examples
  Architecture
GraphX