Large Scale Graph Processing using MapReduce Model from the Twister Perspective

-- Litao Deng (Cloud Computing Group)
Why Large Scale Graph Processing?

• Widespread Application.
  o Social network analysis.
  o Transportation networks.
  o Hyperlink structure of the World Wide Web.

• Explosive Graph Size.
  o Millions of vertices and billions of edges.

Traditional approaches (single machine) may fail. ✗
Find insights in the Distributed/Cloud world (clusters). ✓
Why MapReduce Model?

- MapReduce is a programming model which simplified the data processing on large clusters. Proposed by Google 2004.

- There are many effective implementations of MapReduce model: Hadoop, DryadLINQ, Daytona and Twister. Open Source Project became the de facto standard.

Migrate the large scale graph processing to the MapReduce model.
Does it work?

- **Bad News:**
  - There is no general methods to effectively map graph algorithms to MapReduce model. Some research has demonstrated the overhead of MR iteration and communication is unacceptable.

- **Good News:**
  - Some attempts have obtained preliminary results.

---

**Graph Twiddling in a MapReduce world**

**Max-cover algorithm in map-reduce**

**Minimum Spanning Tree and Connectivity of Large Scale Graphs in MapReduce**
Common features in the graph processing

Traverse the Graph.
  o BFS.
  o DFS.
  o Shortest-Path.
  o Max-Flow.

In order to traverse a graph, we may iterate the graph one layer by another.
Twister

An iterative MapReduce runtime, which compared to the single step MapReduce computation.

- Static vs. Variable Data.
- Long Running Map/Reduce tasks.
- Granularity of Tasks.
- Side-effect-free Programming.
- Combine Operation.
- Programming Extensions.
Twister

- A distributed in-memory MapReduce runtime optimized for iterative MapReduce computations.

- Publish/Subscribe messaging infrastructure.

- Twister Driver, Twister Daemon, Broker Network.
Twister

- Handling Input and Output Data
- Handling Intermediate Data
- Pub/Sub Messaging
- Scheduling Tasks
- Fault Tolerance
Case Study

Map Shortest-Path Algorithm to iterative MapReduce model.

- **Intuition:**
  DistanceTo(startNode) = 0;
  For all nodes directly reachable from startNode, DistanceTo(n) = 1;
  For all nodes reachable from some other set of nodes S,
  DistanceTo(n) = 1 + min(DistanceTo(m), m is in S);

- **Solution:**
  Mapper: key -> node n; value -> D (distance from startNode), list of reachable nodes from n.
  Reducer: gather possible distances to a given P and selects the minimum one.
Shortcomings

• How to control the number of the iterations?

• If we would like to know all of the shortest-path between every pairwise in the graph (such as in the case of Betweenness Centrality). How could we solve this?

• When the memory cannot load all of the data?
Thanks

-- Stay hungry, stay foolish.