PART 2. SCALABLE FRAMEWORKS FOR REAL-TIME BIG DATA ANALYTICS

3. GRAPH ANALYSIS

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs535

This material is built based on,
- Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, Names C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski, “Pregel: a system for large-scale graph processing”, Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135-146
- Apache Giraph
  - http://giraph.apache.org
- Apache Hama
  - Open source project inspired by Pregel
  - http://hama.apache.org
- Apache GraphX

FAQs
- November 30: Exam review
  - Presentation Schedule is ready
- [11/25, Tuesday 8:45AM – 10:45AM]
  - Team Rocky Mountain Maple
  - Team Rocky Mountain Juniper
  - Team Building Aspen
  - Team Ponderosa Pine
  - Team Peachleaf Willow
  - Team Narrowleaf Cottonwood
- [11/27, Thursday 8:45AM – 10:45AM]
  - Team Limber Pine
  - Team Engelmann Spruce
  - Team Chokecherry
  - Team Boxelder
- [12/5, Tuesday 8:45AM – 10:45AM]
  - Team Rocky Mountain Maple
  - Team Rocky Mountain Juniper
  - Team Building Aspen
  - Team Ponderosa Pine
  - Team Peachleaf Willow
  - Team Narrowleaf Cottonwood
- [12/7, Thursday 8:45AM – 10:45AM]
  - Team Limber Pine
  - Team Engelmann Spruce
  - Team Chokecherry
  - Team Boxelder
  - Final report+software submission deadline: 5PM December 4
  - Software demo: December 7, 8

Today’s topics
- Graph storage and processing models

Pregel:
A system for large-scale graph processing

Graph analysis at Google?
- MapReduce tasks
  - Google’s 80% of data analysis
  - Large-scale web search indexing
  - Clustering problems for Google News
  - Producing reports for popular queries (e.g. Google Trend)
  - Processing of satellite imagery data
  - Language model processing for statistical machine translations
  - Large-scale machine learning problems
  - Back-up/recover
- The other 20%?
Graph analysis at Google?

- Large graph analysis
  - Graph algorithms
    - PageRank
    - Shortest path
    - Connected components
    - Clustering techniques
  - Graph data
    - Web graph
    - Transportation routes
    - Citation relationships
    - Social networks

Processing of large graphs

- Poor locality of memory access for graph algorithms
- Very little work per vertex
- A changing degree of parallelism over the course of execution

MapReduce is NOT great for graph processing

- Many iterations are needed for parallel graph processing
- Materializations of intermediate results at every MapReduce iteration causes performance bottleneck

Single Source Shortest Path (SSSP)

- Find shortest path from a source node to all target nodes
- If you have a single processor machine?
  - Dijkstra’s algorithm

Dijkstra’s Algorithm (Single node)
Dijkstra's Algorithm (Single node)

Adjacency list:
- A: (B, 10), (C, 5)
- B: (C, 2), (D, 1)
- C: (B, 3), (D, 9)
- D: (E, 4)
- E: (A, 7), (D, 6)

Using MapReduce

Adjacency list:
- A: (B, 10), (C, 5)
- B: (C, 2), (D, 1)
- C: (B, 3), (D, 9)
- D: (E, 4)
- E: (A, 7), (D, 6)

Pregel
Finding SSSP using MapReduce

Mapper: calculates <dest node ID, dist>
Map input: <nodeID, <init, adj list>>
- B: <B, 10>, <C, 5>
- C: <B, 2>, <D, 1>
- D: <E, 4>
- E: <A, 7>, <D, 6>

Flushed to local FS
Using MapReduce

Reduce input: <nodeID, dist>
- A, <0, <(B,10), (C,5)>>
- B, <10, <(C,2), (D,1)>>
- C, <5, <(B,3), (D,9)>>
- D, <inf, <(E,4)>>
- E, <inf, <(A,7), (D,6)>>

Reduce output: <dest node ID, dist>
- A, 10
- B, 10
- C, 5
- D, <inf, <(E,4)>>
- E, <inf, <(A,7), (D,6)>>

Map output: <dest node ID, dist>
- A, 10
- B, 10
- C, 5
- D, <inf, <(E,4)>>
- E, <inf, <(A,7), (D,6)>>

Flushed to local FS

Keep going...

Pregel Model of Computation
Computational Model

Supersteps: A Sequence of Iterations

Inspired by Valiant's Bulk Synchronous Parallel model (1990)

Local computations
Communication
Barrier Synchronization

Computation Model (1/2)

- **Superstep**: the vertices compute in parallel
  - Each vertex
    - Receives messages from the previous superstep
    - Executes the same user-defined function
    - Sends messages to other vertices
    - Mutates the topology of the graph if need be
    - Votes to halt if it has no further work to do
  - **When to terminate?**
    - All vertices are simultaneously inactive
    - Voting to halt
    - There are no messages in transit

Computation Model (2/2)

- **Input to the Pregel computation**
  - A directed graph
  - String vertex ID
  - Associated user-defined value
  - Edge
    - Associated with their source vertices
    - User-defined value and a target vertex ID

- **Computation in the vertex**
  - Executes the same user-defined function
  - Modifies the state
  - Sometimes changes the outgoing edges
  - Receive/send message
  - Mutate topology
  - There is no computation associated with the edges

Vertex State Machine

- In superstep 0,
  - Every vertex is in the active state
  - All active vertices participate in the computation of any given superstep

- A vertex deactivates itself by voting to halt
  - The vertex has no further work to do unless triggered externally
  - Pregel will not process that vertex in subsequent supersteps
  - Unless there is a message passed from the previous superstep

- Once a vertex is re-activated
  - It must explicitly deactivate itself again

Output of a Pregel program

- Set of values explicitly output by the vertices
  - Often a directed graph isomorphic to the input
  - E.g. clustering algorithm
  - E.g. graph mining algorithm
    - Generates aggregated statistics mined from the graph
Message Passing (1/2)
- Vertices communicate directly with one another by sending messages
  - Message value
  - Name of the destination vertex
- A vertex can send any number of messages in a superstep
  - There is no guaranteed order of messages in the iterator.
    - However,
    - Message is delivered reliably
    - There will be no duplicate

Message Passing (2/2)
- Common usage pattern
  - A vertex V to iterate over its outgoing edges and sending a message to the destination vertex of each edge
  - Destination vertex need not be a neighbor of V
    - E.g.: A vertex can learn the identifier of a non-neighbor from a message received earlier
      - E.g.: implicitly vertex info is distributed
  - If destination does not exist, user-defined handler will be executed.
    - Create the missing vertex or remove the dangling edge

SSSP using parallel BFS in Pregel

SSSP using parallel BFS in Pregel
SSSP using parallel BFS in Pregel

- What was the criteria to vote to halt?
SSSP using parallel BFS in Pregel

- What was the criteria to vote to halt?
  - If there is no change of value (distance to the current node), vote to halt

Pregel vs. MapReduce

- Many of graph algorithms can be written as a series of chained MapReduce invocations
  - Pregel
    - Once the vertices and edges are loaded into computing nodes, they will stay on that machine
    - Only messages will be transferred through the network
  - MapReduce
    - Passes the entire state of graph for every iteration
    - External coordinator is required to create a “chain” of MapReduce jobs

System Architecture

- Master/worker model
  - Worker
    - Processes user-defined tasks
    - Communicates with other workers (messageing)
  - Master
    - Maintains information about workers
    - No portion of graph assigned
    - Recovers from faults
    - Uses monitoring tools
  - Underlying persistent data storage: GFS or BigTable
  - Temporary data is stored on local disk

Step-by-step execution (1/4)

- A client launches a Pregel job
  - Many copies of the user program begin executing on a cluster of machines
  - One of these copies acts as the master
  - Workers use the cluster management system’s name service to discover the master’s location
  - Send registration messages to the master
Step-by-step execution (2/4)

2. The master assigns a partition of the input to each worker
   - Worker:
     - Loads the vertices and marks them as active
     - Maintains the state of its section of the graph
     - Executes user's `Compute()` method on its vertices
     - Manages messages to and from other workers

Step-by-step execution (3/4)

3. The master instructs each worker to perform a superstep
   - Performs user-defined function on the active vertices
   - Messages are sent asynchronously
   - Before the end of the superstep:
     - This step is repeated until: (a) all of the vertices are inactive simultaneously && (b) no messages are transferred

Fault tolerance (1/2)

- System maintains checkpoints
  - The master periodically requests the workers to save the state of their partitions to persistent storage
    - State is saved as checkpoints, and includes:
      - Vertex values, edge values, incoming messages

Fault tolerance (2/2)

- Failure detection
  - Regular “ping” message

- Recovery
  - The master reassigned graph partitions to the current available workers
  - The workers all reload their partition state from most recent available checkpoint

PageRank Algorithm in Pregel
PageRank Algorithm

- Link analysis algorithm
- Probability distribution
- Represents the likelihood that a person randomly clicking on links will arrive at any particular page

- Probability
  - Between 0 and 1
  - PageRank of 0.5
  - There is a 50% chance that a person clicking on a random link will be directed to the document with the 0.5 PageRank

Iterative approach

- A link to a page counts as a vote of support
- At \( t=0 \), \( PR(p,0)=1/N \)
- At each time step, the computation yields,

\[
PR(p_{j}; t+1) = \frac{1-d}{N} + d \sum_{j \in M(p_i)} \frac{PR(p_j; t)}{L(p_j)}
\]

- Damping factor, \( d \)
  - An imaginary surfer who is randomly clicking on links and he/she will eventually stop
  - The probability that the imaginary person will continue in that step
  - Generally assumed as around 0.85

- \( PR(p,t) \): PageRank for the page \( p \) at timestep \( t \)
- \( L(p) \): number of links from page \( p \)

Example:

- \( PR(A)=0.385875 \), \( PR(B)=0.47799375 \), \( d =0.85 \)
- \( PR(A) = (1-d) + (0.85 \times 0.47799375) = 0.3425 \)
- \( PR(B) = (1-d) + (0.85 \times 0.3425) = 0.291775 \)

- The numbers just keep going up.
- But will the numbers stop increasing when they get to 1.0?
- What if a calculation over-shoots and goes above 1.0?

The numbers are heading down.
- The numbers will get to 1.0 and stop
In Pregel

```cpp
class PageRankVertex : public Vertex<double, void, double> {
  // Constructor and destructor
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next()) {
        sum += msgs->Value();
      }
      mutable_value() = 0.15 / NumVertices() + 0.85 * sum;
    } else if (superstep() < 30) {
      // Compute PageRank
      const int64_t n = GetOutEdgeIterator().size();
      sendMessageToAllNeighbors(mutable_value() / n);
    } else {
      // Stop processing
      voteToHalt();
    }
  }
};
```