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

FAQs
• Presentation Schedule is ready
  • Team Once (Dec.1)
  • Team Ocho (Dec.1)
  • Team Siete (Dec.6)
  • Team Seis (Dec.6)
  • Team Cinco (Dec.6)
  • Team Cuatro (Dec.6)
  • Team Tres (Dec.8)
  • Team Dos (Dec.8)
  • Team Uno (Dec.8)

Today’s topics
• Graph storage and processing models
  • Pregel

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

Graph analysis at Google?
• MapReduce tasks
  • Google’s 80% of data analysis
    • Large-scale web search indexing
    • Clustering problems for Google News
    • Produce 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/restore

• 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

Pregel
Finding SSSP using Dijkstra’s Algorithm with a Single Node

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

![Diagram of Dijkstra's Algorithm](image)

<table>
<thead>
<tr>
<th>(Source)</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Using MapReduce

**Adjacency Matrix**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**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)

**Mapper**

- Calculates <dest node ID, dist>
- <A, <0, <(B,10), (C, 5)>>>
- <B, <inf, <(C,2), (D,1)>>>
- <C, <inf, <(B,3), (D,9)>>>
- <D, <inf, <(E,4)>>>
- <E, <inf, <(A,7), (D,6)>>>

Flushed to local FS

**Pregel**

Finding SSSP using MapReduce
Using MapReduce

Reduce input: <nodeID, dist>
- A, 0, <B,10>, <C,5>>
- A, inf

B, 10, B, inf
- C, inf, <B,3>, <D,9>>
- C, 5, C, inf

D, inf, <E,4>>
- D, inf, <D, inf>

E, inf, <A,7>, <D,6>>
- E, inf, E, inf

Reducer: Find the minimum distance <nodeID, dist>
- A, 0, <B,10>, <C,5>>
- B, 10, B, inf
- C, 5, C, inf
- D, inf, <E,4>>
- E, inf, <A,7>, <D,6>>

Map output: <dest node ID, dist>
- B, 10
- C, 5
- D, 11
- E, inf

Keep going...

Flushed to local FS
Computational Model

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

Processors

Communication

Barrier Synchronization

Local computations

Superstep: a Sequence of Iterations

Input

Output

Supersteps

Computational 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

Computational Model (2/2)

• Input to the Pregel computation
  • A directed graph
    • Vertex
      • 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

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
    • Creates small set of disconnected vertices selected from a large graph
  • E.g. graph mining algorithm
    • Generates aggregated statistics mined from the graph

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 messages passed from the previous superstep

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

• When to terminate?
  • Voting to halt
  • There are no messages in transit
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

It you did not update your value, inactive the state.
SSSP using parallel BFS in Pregel

Vote to halt
Message received
Inactive
Active

SSSP using parallel BFS in Pregel

Vote to halt
Message received
Inactive
Active
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

Big Data research?

- Feel free to join our research group meeting
  - Friday 1:00PM 325 (or 210)

- Visit my web page
  -/~sangmi

- Come and talk to me!