Today’s topics
- Pregel: Think like a vertex!
- GraphX
  - Vertex cut, RDD based graph parallel architecture

Graph partitioning: Edge-cut
- Vertices are equally distributed among partitions
- Then, edges are distributed across partitions
  - Edges along with the corresponding vertices are replicated and passed according to the requirement between partitions
  - Communication cost associated with edge-cut algorithms is directly proportional to the number of edges cut
- Both edge data and vertex data are passed to between partitions

Graph partitioning: Vertex-cut
- Edges are equally distributed
- Then, vertices are cut and replicated across the partitions
  - Vertex data is passed between the partitions
  - Communication cost is directly proportional to the number of the vertex replicas
  - Load balancing factor is determined by the number of edges assigned to each of the partitions
- Passes just vertex data
Graph partitioning: Vertex-cut: Example

- Passes just vertex data

Example
Distribute edges equally over 2 partitions with a hash function, \((V_{s} + V_{d}) \mod n\), where \(V_{s}\) is the source vertex id, and \(V_{d}\) is the dest vertex id.

\((1,2), (5,2), (2,5), (5,4), (4,5), (4,3)\) are placed into the partition 1
Rest of them are placed into the partition 2

Assume, vertices are partitioned using hash partitioner, \(v \mod n\)
Finally, it will create incomplete subgraphs

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 (messaging)
  - 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)

1. 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

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 2
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

Step-by-step execution (4/4)

4. After the computation halts, the master may instruct each worker to save its portion of the graph

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

GEAR Workshop II | Large Scale Graph Analysis
Pregel: “Think Like Vertex!”
PageRank with Pregel

http://www.cs.colostate.edu/~cs535
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,
  \[ PR(p; t+1) = \frac{1 - d}{N} + d \sum_{j \in \text{Out}(p)} \frac{PR(p; t)}{L(p_j)} \]

- 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

In Pregel

```java
public 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();
  }
}
```

http://www.cs.colostate.edu/~cs535
GraphX unifies computation on Tables and Graphs

Providing a single system to support the entire pipeline

Two separate systems

Apache Spark RDD dataflow systems

Graph analysis frameworks

Dependency Graph

Data pipelining systems are not optimized for the Graph processing

- Hadoop is 60x slower than GraphLab
- Spark is 16x slower than GraphLab

Key Research Question:

“How can we naturally express and efficiently execute graph computation in a general purpose dataflow framework?”

Approach 1: Representation of data

Approach 2: Optimization of computing

http://www.cs.colostate.edu/~cs535
The Property Graph Data Model

- Property graph
  - Associates user-defined properties with each vertex and edge
  - Meta-data (e.g., user profiles and timestamps) and program state (e.g., the PageRank of vertices or inferred affinities)
- Dataflow model vs. property graph model
  - Dataflow systems whose operators (e.g., join) can span multiple collections
  - Operations in graph processing systems (e.g., vertex programs) are typically defined with a single property graph with a pre-declared, sparse structure

The GAS Decomposition

- Gonzalez et al.¹ observed that most vertex programs interact with neighboring vertices by collecting messages in the form of a generalized commutative associative sum and then broadcasting new messages in an inherently parallel loop


Types of graph computation

- **Gather**: Your computation gathers information from neighboring vertices
  - e.g., authority value of the HITS algorithm
  - e.g., current PageRank value

- **Apply**: The vertex applies an update the vertex property
  - e.g., update the authority value with the sum of new authority values after normalizing the value
  - e.g., Add passed PageRank values and normalize it and update the current PageRank value

- **Scatter**: A vertex should send out information to neighboring vertices.

PageRank example with the GAS decomposition

```python
def Gather(a: Double, b: Double) = a + b

def Apply(v, msgSum):
  PR(v) = 0.15 + 0.85 * msgSum
  if converged(PR(v))
    voteToHalt(v)

def Scatter(v, j) = PR(v) / NumLinks(v)
```

http://www.cs.colostate.edu/~cs535
The GAS Decomposition

- pull-based model of message computation
  - The system asks the vertex program for value of the message between adjacent vertices
  - Rather than the user sending messages directly from the vertex program
  - Therefore, vertex-cut is suitable for this style of computation
- Limited communication pattern
  - Supports only between adjacent vertices

Graph Partitioning: EdgePartition2D

- Inspired by the multilevel k-way partitioning
- 2D graph partitioning
- Upper bound of $2\sqrt{n} - 1$ on the vertex replication factor
  - where $n$ is the number of partitions

Step 1: Creating a partition table

- If $n$ is a perfect square
  - rows = the floor value of $(n + \text{cols} - 1)$
  - cols = the ceiling of the decimal value of $\sqrt{n}$
  - For example, if $n = 27$, cols = 6 and rows = 5
  - The last column would have 3 rows

Step 3: Storing edge properties

- Storing Edge Properties
  - (col x rows + rows) otherwise

Vertex assignment

- Using elementary modular hash $v \% n$
- Vertices are equally distributed among the partitions

Edge assignment

- The source vertex ($sr$) is mapped on the columns
  - If $n$ is a perfect square
    - col = $\text{floor}(sr \times \text{mixingPrime} / \sqrt{n})$, if $n$ is a perfect square
    - col = $\text{floor}(sr \times \text{mixingPrime} / \sqrt{n})$, otherwise
  - where mixingPrime is a large prime number to improve the balance of edge distributions
- The destination vertices ($dr$) is mapped on the rows
  - If $n$ is a perfect square
    - row = $\text{floor}(dr \times \text{mixingPrime} / \sqrt{n})$, if $n$ is a perfect square
    - row = $\text{floor}(dr \times \text{mixingPrime} / \sqrt{n})$, otherwise
  - and col < cols - 1
  - row = $\text{floor}(dr \times \text{mixingPrime} / \text{lastColIndex})$, otherwise

Edge assignment

- The source vertex ($sr$) is mapped on the columns
  - If $n$ is a perfect square
    - col = $\text{floor}(sr \times \text{mixingPrime} / \sqrt{n})$, if $n$ is a perfect square
    - col = $\text{floor}(sr \times \text{mixingPrime} / \sqrt{n})$, otherwise
  - where mixingPrime is a large prime number to improve the balance of edge distributions
- The destination vertices ($dr$) is mapped on the rows
  - If $n$ is a perfect square
    - row = $\text{floor}(dr \times \text{mixingPrime} / \sqrt{n})$, if $n$ is a perfect square
    - row = $\text{floor}(dr \times \text{mixingPrime} / \sqrt{n})$, otherwise
  - and col < cols - 1
  - row = $\text{floor}(dr \times \text{mixingPrime} / \text{lastColIndex})$, otherwise

http://www.cs.colostate.edu/~cs535

Spring 2019 Colorado State University, page 7
Understanding the effect of EdgePartition2D

- Let's locate an edge \((v_{\text{src}}, v_{\text{des}})\)
- All the edges where \(v_{\text{src}}\) is the source vertex
- Would be placed in the same column, col
- Example:
  - If \(v_{\text{src}} = 9\) and \(\text{mixingPrime} = 3\) for the 2D (row, col) partitions
  - \((9, 3)\% 3 = 2\)
- The actual cell will be determined by the destination vertex
  - If \(v_{\text{src}} = 2\) and \(\text{mixingPrime} = 3\)
  - \((2, 3)\% 3 = 1\)
- Therefore, the edge \((v_{\text{src}}, v_{\text{des}})\) is stored in the cell \((v_{\text{src}}, v_{\text{des}})\% 3\)

\[0 \quad 1 \quad 2 \quad 3 \quad 4\]

Understanding the effect of EdgePartition2D

- Therefore, any edge containing \(v\) has to be placed in any of \(\lfloor v/\text{mixingPrime}\rfloor \% \text{mixingPrime}\) partitions
- The upper bound on the vertex replication factor is \(\lfloor v/\text{mixingPrime}\rfloor \% \text{mixingPrime}\)
- This is directly related to the communication cost to synchronize the status of the vertex properties

```latex
\begin{array}{cc}
\text{Source} & \text{Destination} \\
0 & 1 \\
1 & 2 \\
2 & 3 \\
3 & 4 \\
\end{array}
```

Mirror Vertices

- High-degree vertices often have multiple neighbors on the same remote machine
- Avoiding repetitive communications to the same node

Graph Parallel Computation

- A sequence of join stages and group-by stage
  - Combined with map operations
  - Join stage
    - Vertices and edge properties are joined to form the triplet view
    - Triplet view: consisting each edge and its corresponding source and destination
  - Group-by stage
    - The triplets are grouped by source or destination vertex
    - Construct the neighborhood of each vertex and compute aggregation

---

**Encoding Property Graphs as Tables**

<table>
<thead>
<tr>
<th>Vertex Table (RDD)</th>
<th>Edge Table (RDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property Graph</td>
<td></td>
</tr>
</tbody>
</table>

---

http://www.cs.colostate.edu/~cs535
Join and group-by stages vs. the GAS decomposition

- Join and group-by stages captures the GAS decomposition
- Gathers: group-by with the same destination vertex
- Applies: intervening map operation
- Scatters: join stage with new vertex property to all adjacent vertices

Graph Operators

- mrTriplets (MapReduce Triplets) operator
  - map and group-by dataflow operator on the triplets view

Distributed Graph Representation

- GraphX represents graphs internally as a pair of vertex and edge collections built on the Spark RDD abstraction
  - Vertex collection
    - Hash-partitioned by the vertex ids
  - Edge collection
    - 2D Edge partitioning
      - Strong upper bounds on the communication complexity of operators
      - E.g. mrTriplets
      - The edges within a partition are clustered by source vertex id
      - CSR (compressed sparse raw)
      - Hash-indexed by their target id

Incremental View Maintenance

- Iterative graph algorithms often modify only a subset of the vertex properties in each iteration
- Incremental view maintenance
  - Avoiding unnecessary movement of unchanged data
  - GraphX tracks vertices with modified properties
  - Only the changed vertices are routed to their edge-partition join sites

Highly skewed power-law degree distributions

System Performance Comparison

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Edges</th>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>twitter-2010 (5,4)</td>
<td>1,468,305,182</td>
<td>41,653,230</td>
</tr>
<tr>
<td>uk-2007-05 (5,4)</td>
<td>3,738,733,648</td>
<td>105,896,555</td>
</tr>
</tbody>
</table>

http://www.cs.colostate.edu/~cs535
System Performance Comparison [2/3]

(c) Spark did not finish within 8000 seconds, Giraph and Spark + Part. ran out of memory.

System Performance Comparison [3/3]

- Left: Strong scaling for PageRank on twitter data
- Right: Effect of partitioning on communication