One Trillion edges : Graph Processing at Facebook-Scale

Tong Niu
tong.niu.cn@outlook.com
Outline

• Introduction
  • Improvements
• Experiment Results
• Conclusion & Future Work
• Discussion
Introduction

• Graph Structures(entities, connections)
  • social networks
    • Facebook manages a social graph that is composed of people, their friendships, subscriptions, likes, posts, and many other connections.

1.39B active users in 2014 with more than 400B edges
Introduction

• What is Apache Giraph?
  • “Think like a vertex”
    • Each vertex has an id, a value, a list of adjacent neighbors and corresponding edge values

• Bulk synchronous processing (BSP)
  • Break up to several supersteps (iteration)
  • Messages are sent during a superstep from one vertex to another and then delivered in the following supersteps
Introduction

• What is Apache Giraph?
Introduction

• What is Apache Giraph?

  • **Master** – Application coordinator
    • Assigns partitions to workers
    • Synchronizes supersteps
  
  • **Worker** – Computation, messaging
    • Load the graph from input splits
    • Does the computation/messaging of its assigned partitions
1. Flexible vertex/edge based input

• Original input:
  • All data(vertex/edge) need to be read from the same record and assumed to the same data source

• Modified input:
  • Allow loading vertex data and edges from separate sources
  • Add an arbitrary number of data sources
2. Parallelization support

Original:

- Scheduled as a single MapReduce job

Modified:

- Add more workers per machine
- Use local multithreading to maximize resource utilization
3. Memory optimization

• Original:
  • Large memory overhead because of flexibility

• Modified:
  • Serialize the edges of every vertex into a bit array rather than using native direct serialization methods
  • Create an OutEdges interface that allow developers to achieve edge stores
4. Sharded aggregators

- global computation (min/max value)
- provide efficient shared state across workers
- make the values available in the next superstep
4. Sharded aggregators

• Original:
  • Use znodes in zookeeper to store partial aggregated data from workers, master aggregate all of them and write result back to znode for workers to access it
  • every worker has plenty of data that need to be aggregated

• Modified:

Randomly assigned to one of the workers
Distribute final values to master/workers
K-Means clustering

In a graph application, input vectors are vertices, and centroids are aggregators.
1. Worker phases

- Add preApplication() to initialize positions of centroids
- Add preSuperstep() to calculate the new position for each of the centroids before next superstep

2. Master computation

- Centralized computation prior to every superstep that can communicate with the workers via aggregators
3. Composable computation

- Allows us to use different message types, combiners and computation to build a powerful k-means application.

4. Superstep splitting

- For a message heavy superstep
- Send a fragment of messages to the destinations and do a partial computation during each iteration.
Experiment results

Scalability of workers (200B edges)

Scalability of edges (50 workers)
Experiment results

- Giraph (200 machines) vs Hive (at least 200 machines)
  - compare CPU time and elapsed time
    - label propagation algorithm

<table>
<thead>
<tr>
<th>Graph size</th>
<th>Hive</th>
<th>Giraph</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>701M+ vertices</td>
<td>Total CPU 9.631M secs</td>
<td>Total CPU 1.014M secs</td>
<td>9x</td>
</tr>
<tr>
<td>48B+ edges</td>
<td>Elapsed Time 1,666 mins</td>
<td>Elapsed Time 19 mins</td>
<td>87x</td>
</tr>
</tbody>
</table>

- Weighted PageRank

<table>
<thead>
<tr>
<th>Graph size</th>
<th>Hive</th>
<th>Giraph</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B+ vertices</td>
<td>Total CPU 16.5M secs</td>
<td>Total CPU 0.6M secs</td>
<td>26x</td>
</tr>
<tr>
<td>400B+ edges</td>
<td>Elapsed Time 600 mins</td>
<td>Elapsed Time 19 mins</td>
<td>120x</td>
</tr>
</tbody>
</table>
Conclusion & Future work

How a processing framework supports Facebook-scale production workloads. We have described the improvements to Giraph.

1. Determine a good quality graph partitioning prior to our computation.
2. Make our computation more asynchronous to improve convergence speed.
3. Leverage Giraph as a parallel machine-learning platform
Discussion