Big Graph Processing

Fenggang Wu
Nov. 6, 2016
## Agenda

<table>
<thead>
<tr>
<th>Project</th>
<th>Publication</th>
<th>Organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pregel</td>
<td>SIGMOD’10</td>
<td>Google</td>
</tr>
<tr>
<td>PowerGraph</td>
<td>OSDI’12</td>
<td>CMU</td>
</tr>
<tr>
<td>GraphX</td>
<td>OSDI’14</td>
<td>UC Berkeley AMPLab</td>
</tr>
<tr>
<td>PowerLyra</td>
<td>EuroSys’15</td>
<td>Shanghai Jiao Tong University</td>
</tr>
<tr>
<td>One Trillion Edges</td>
<td>VLDB’15</td>
<td>Facebook</td>
</tr>
<tr>
<td>Chaos</td>
<td>SOSP’15</td>
<td>EPFL</td>
</tr>
</tbody>
</table>
Pregel: A System for Large-Scale Graph Processing

- C++, close source project by google
- Distributed Graph Processing Framework
Big Graph Processing

• E.g.
  • Problem: Single source shortest path (SSSP)
  • H/W: Large number of commodity computing nodes
  • Input: Google bigtable.

• How can a programmer design (from scratch)?
  • Distributed version of the algorithm (depends on the algorithm).
  • Communication and Storage mechanism.
  • Cluster management, job assignment, checkpoint, fault tolerant.
  • etc.

• What if other problems?
  • Page rank, clustering?
Distributed Graph Processing Framework

SSSP Solution
- Distributed SSSP
- Communication
- Storage
- Clustering mgmt
- Job Scheduling/tracking
- checkpointing
- fault tolerant

PageRank Solution
- Distributed PageRank
- Communication
- Storage
- Clustering mgmt
- Job Scheduling/tracking
- checkpointing
- fault tolerant

Clustering Solution
- Distributed Clustering
- Communication
- Storage
- Clustering mgmt
- Job Scheduling/tracking
- checkpointing
- fault tolerant

...
Distributed Graph Processing Framework (Cont’d)

- SSSP Solution
  - Distributed SSSP

- PageRank Solution
  - Distributed PageRank

- Clustering Solution
  - Distributed Clustering

- Communication
- Storage
- Clustering mgmt
- Job Scheduling/tracking
- Checkpointing
- Fault tolerant
Pregel: Code Examples

SSSP Solution (17 LOC)

```java
Class ShortestPathVertex
    : public Vertex<int, int, int> {
        public:
            virtual void Compute(MessageIterator* msgs) {
                int minDist = IsSource((vertex_id()) ? 0 : INF;
                for (; !msgs->Done(); msgs->Next())
                    minDist = min(minDist, msgs->Value());
                if (minDist < GetValue()) {
                    *MutableValue() = minDist;
                    OutEdgeIterator iter = GetOutEdgeIterator();
                    for ( ; !iter.Done(); iter.Next())
                        SendMessageTo(iter.target(),
                            minDist + iter.GetValue());
                }
                VoteToHalt();
            }
    }
```

PageRank Solution (16 LOC)

```java
Class PageRankVertex
    : public Vertex<double, void, double, double> {
        public:
            virtual void Compute(MessageIterator* msgs) {
                if (superstep() >= 1) {
                    double sum = 0;
                    for (; !msgs->Done(); msgs->Next())
                        sum += msgs->Value();
                    *MutableValue() = 0.15 + 0.85 * sum;
                }
                if (supersteps() < 30) {
                    const int64 n = GetOutEdgeIterator().size();
                    Send_MessageToAllNeighbors(GetValue() / n);
                } else {
                    VoteToHalt();
                }
            }
```

That’s it!

• Thank you!....?
Outline

• Computation Model (distributed version of any algorithm)
• Implementation Details
Computation Model

• “Think Like a Vertex”
  • Coding graph algorithms as vertex-centric programs to process vertices in parallel and communicate along edges

• Message Passing among Vertices

• Many iterations of “Super Steps”
Example: Finding Largest Value in a Graph

- “Think Like a Vertex”!
- Nodes/Edges
- Super Steps
- Active/Inactive nodes
Pregel Code for finding the max value

Class MaxFindVertex
:
  public Vertex<double, void, double> {
    public:
      virtual void Compute(MessageIterator* msgs) {
        int currMax = GetValue();
        SendMessageToAllNeighbors(currMax);
        for ( ; !msgs->Done(); msgs->Next()) {
          if (msgs->Value() > currMax)
            currMax = msgs->Value();
        }
        if (currMax > GetValue())
          *MutableValue() = currMax;
        else VoteToHalt();
      }
    }
Super Step
Model Of Computation: Entities

• Vertex:
  • Identified by a unique identifier.
  • Has a modifiable, user defined value.

• Edge:
  • Source vertex and Target vertex identifiers.
  • Has a modifiable, user defined value.
  • Partitioned with its source vertex
Model Of Computation: Progress

• In superstep 0, all vertices are active.
• Only active vertices participate in a superstep.
  • They can go inactive by voting for halt.
  • They can be reactivated by an external message from another vertex.
• The algorithm terminates when all vertices have voted for halt and there are no messages in transit.
Comparison with MapReduce

• Graph algorithms can be implemented as a series of MapReduce invocations but it requires passing of entire state of graph from one stage to the next, which is not the case with Pregel.

• Also Pregel framework simplifies the programming complexity by using supersteps.
Some Optimizations

• Combiners
  • Combine the message first before send to another machine

• Aggregators
  • In super step S, Every vertex can produce a value for aggregation. Aggregated value is available to all vertices for super step S+1
  • Usage: global communication & statistics
C++ APIs

• virtual function: Compute()
• Combiners and Aggregators
• Topology Mutations
Implementation Details

• Basic Architecture
• Pregel Execution
• Fault Tolerance
Basic Architecture

• The Pregel library divides a graph into partitions, based on the vertex ID, each consisting of a set of vertices and all of those vertices’ outgoing edges.

• The default function is hash(ID) mod N, where N is the number of partitions.
Pregel Execution

• 1. Many copies of the user program begin executing on a cluster of machines. One of these copies acts as the master.

• The master is not assigned any portion of the graph, but is responsible for coordinating worker activity.
Pregel Execution

2. The master determines how many partitions the graph will have and assigns one or more partitions to each worker machine.

Each worker is responsible for maintaining the state of its section of the graph, executing the user’s Compute() method on its vertices, and managing messages to and from other workers.
Pregel Execution

• 3. The master assigns a portion of the user’s input to each worker.
• The input is treated as a set of records, each of which contains an arbitrary number of vertices and edges.
• After the input has finished loading, all vertices are marked as active.
Pregel Execution

• 4. The master instructs each worker to perform a superstep. The worker loops through its active vertices, and call Compute() for each active vertex. It also delivers messages that were sent in the previous superstep.

• When the worker finishes it responds to the master with the number of vertices that will be active in the next superstep.
Processing Model:
All "active" node will be executed
Whole processing completed when
a. No more active node
b. No more in-transit messages

Superstep execution:
1. Receive message from inbox
2. Modify node and arc properties
3. Halt self (until new message received)
4. Send messages to other nodes (causing them active)
5. Remove existing or create new arcs
Fault Tolerance

• Checkpointing is used to implement fault tolerance.
  • At the start of every superstep the master may instruct the workers to save the state of their partitions in stable storage.
  • This includes vertex values, edge values and incoming messages.

• Master uses “ping“ messages to detect worker failures.
• When one or more workers fail, their associated partitions’ current state is lost.

• Master reassigns these partitions to available set of workers.
  • They reload their partition state from the most recent available checkpoint. This can be many steps old.
  • The entire system is restarted from this superstep.

• Confined recovery can be used to reduce this load
PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

• OSDI’12, CMU

• Real world natural graph
  • highly skewed, Power law graph $P(d) \propto d^{-\alpha}$
  • 1% nodes associated with 50% edges
  • Pregel assumes vertex having small neighborhood to maximized parallelism
  • Problem: work balance; partitioning; communication; storage; computation

• PowerGraph Abstraction
  • Gather Apply Scatter model
  • Vertex cut (balanced p-way, random, etc.)
  • Async GAS, improves parallelism
GraphX: Graph Processing in a Distributed Dataflow Framework

- Community:
  - specialized graph processing framework > general purpose distributed dataflow framework.
  - tailored APIs, accelerated execution for iterative algorithms.

- The paper argues that general purpose dataflow framework can still have a comparable performance, and provide many other advantages over specialized ones.
  - Unified analytic pipeline

- View graphs as RDDs, graph computation as data flow Ops.

- Optimizations
  - Filtered Index Scanning
  - Automatic Join Elimination
  - Memory-based Shuffle
  - Batching and Columnar Structure
  - Variable Integer Encoding

---

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>GraphX</th>
<th>Pagerank</th>
<th>Connected Comp.</th>
<th>K-core</th>
<th>Triangle Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>(20)</td>
<td>(20)</td>
<td>(50)</td>
<td>(50)</td>
<td></td>
</tr>
<tr>
<td>Connected Comp.</td>
<td></td>
<td>(20)</td>
<td>(50)</td>
<td>(50)</td>
<td></td>
</tr>
<tr>
<td>K-core</td>
<td></td>
<td>(50)</td>
<td>(50)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- GAS Pregel API (34)
- Spark (30,000)
PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs

• Existing systems use a “ONE SIZE FITS ALL” design for skewed graphs, resulting in suboptimal performance
• Edge-cut (Pregel, GraphLab): Locality
• Vertex-cut (PowerGraph, GraphX): Parallelism
• Propose Hybrid-Cut, apply different cut according to the degree of nodes
One Trillion Edges: Graph Processing at Facebook-Scale

• Existing graph processing models experience scaling issues.

• Optimized from APACHE GIRAPH (by Yahoo!, inspired by Pregel, in Java)
  • Why Giraph: Java, interaction with HDFS; Performance; BSP model.

• Optimizations:
  • Flexible vertex/edge based input
  • Parallelization support
  • Memory Optimization
  • Sharded Aggregators.

• Compute module extentions
Chaos: Scale-out Graph Processing from Secondary Storage

• Early graph processing: single machine
• Recently: graph fit entirely in memory

• Contributions:
  • First, Chaos partitions for sequential storage access, rather than for locality and load balance, resulting in much lower pre-processing times.
  • Second, Chaos distributes graph data uniformly randomly across the cluster and does not attempt to achieve locality, based on the observation that in a small cluster network bandwidth far outstrips storage bandwidth.
  • Third, Chaos uses work stealing to allow multiple machines to work on a single partition, thereby achieving load balance at runtime.
Reference

• Pregel: a system for large-scale graph processing. Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski. SIGMOD 2010.

• PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. Joseph E. Gonzalez, Yucheng Low, Haijie Gu, and Danny Bickson, Carlos Guestrin. OSDI'12.

• GraphX: Graph Processing in a Distributed Dataflow Framework. Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, Ion Stoica. OSDI'14.

• Chaos: Scale-out Graph Processing from Secondary Storage. Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, Willy Zwaenepoel. SOSP'15.

• PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. Rong Chen, Jiaxin Shi, Yanzhe Chen and Haibo Chen. Eurosys'15.

• Large-Scale Distributed Graph Computing Systems: An Experimental Evaluation. Yi Lu, James Cheng, Da Yan, Huanhuan Wu. VLDB'15.

• One Trillion Edges: Graph Processing at Facebook-Scale. Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, Sambavi Muthukrishnan. VLDB’15.
Reference (Cont’d)

- http://cs.brown.edu/~debrabant/cis570-website/slides/pregel.pptx
- https://s3.amazonaws.com/dxcweb-contents/ECE750T5-PowerGraph-Extended.pdf