Pregel

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING

MALEWICZ, GRZEGORZ, ET AL.

PRESENTED BY BRIAN WHEATMAN

What is it

Efficient, scalable, fault tolerant, graph processing

Small amount of programming effort for scalable graph analysis

Distribution details hidden from user

Think like a vertex

Challenges

Little work per vertex

Changing parallelism over the course of the algorithm

Poor locality

Other options

Make your own infrastructure

- Substantial implementation effort

MapReduce

- Suboptimal performance
- Entire state is transmitted between steps

Single node compute

- Limits scale

Existing graph systems

- Not fault tolerant

Compute model

Think like a vertex

Directed edges associated with source vertex

Supersteps

- Modify its own state
- Modify its edges
- Review and send messages

Edges do not have associated compute

API

Have to write a new compute function for the vertex class

Each vertex has a single value associated with it

- The value can be a large complex type if needed

No remote reads

Message Passing

- Any number can be sent
- Viewable in the next superstep
- Can send to any node, not just neighbors

Combiners

Messages can be combined

- Reduce number of messages
- Reduce size of buffers
- Examples
  - Sum
  - Min
  - Max

Aggregators

For global communication

Each vertex provides a value that are globally combined

Can be used for information about the graph and statistics

- Finding the number of edges
  - Each vertex outputs its out degree and sum them
- Can also make histograms

Global coordination

- When a condition is satisfied and can start the next phase

Topology Mutation

Vertices can add and remove edges

This can cause conflicts

- Two different vertices trying to add the same new vertex

Conflict resolution

- Removals before additions
- Edge removals before vertex removals
- Vertex additions before edge additions
- User-defined handlers deal with the rest

Examples

PageRank

Shortest Paths

PageRank

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual 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();
    }
  }
};
```

SSSP

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  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();
  }
};
```

SSSP Combiner

```
class MinIntCombiner : public Combiner<int> {
  virtual void Combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};
```

Disadvantages

All computation are synchronous

- Asynchronous operations can lead to faster convergence

Does not take into account known information on graphs

- Such as small world or power law.

Lost single node performance

- GraphChi found they could get ¼ the performance with 1/30th of the cores

References

Malewicz, Grzegorz, et al. "Pregel: a system for large-scale graph processing." *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010.

Low, Yucheng, et al. "Distributed GraphLab: a framework for machine learning and data mining in the cloud." *Proceedings of the VLDB Endowment* 5.8 (2012): 716-727.

Gonzalez, Joseph E., et al. "Powergraph: distributed graph-parallel computation on natural graphs." *OSDI*. Vol. 12. No. 1. 2012.

Kyrola, Aapo, Guy E. Blelloch, and Carlos Guestrin. "Graphchi: Large-scale graph computation on just a pc." USENIX, 2012.

Implementation

On top of Google cluster architecture

- 1000s of commodity machines
- Name service
  - Instances are described by name independent of hardware
- Distributed storage system
  - GFS
  - BigTable

Partitions

- Either just hash(node ID) or user defined function
  - It is known where every vertex is stored by every machine

Worker

Maintains state of its partitions in memory

Queues for incoming messages and outgoing messages

- Buffering messages limits internode traffic

Calls compute for each superstep

Combiners are called in all queues

Master

Determines how many partitions the graph has

- Assigns one or more to each node

Maintains a list of active workers

Ensures everything proceeds in lockstep

- When a node fails goes to failure recover mode

Fault Tolerance

Uses a persistent distributed storage system

Check pointing

Failure detection via pings

Outgoing messages are logged