Giraph

Neil Butcher

The College of Engineering at the University of Notre Dame

Background

- Giraph scalable platform for implementing graph algorithms
- Developed by Apache
- Based off ‘Pregel’
- Utilizes Hadoop MapReduce framework to target graph problems
- Open Source

Advantages of Solving Problems with Giraph

- Message-based communication: no locks
- Global synchronization: no semaphores
- Simple to program
- Massively parallel: task based programming
- Fault tolerant: Saves intermediate results

Giraph Algorithms: Basic Idea

- Algorithms are written from the perspective of a vertex
- Vertices send messages to each other to share pertinent information

How it Works

- *'compute'* function has ability to:
  - modify state of vertex and its outgoing edges
  - Can send messages to other vertices
  - Receive messages sent in previous *superstep*
- Things that happen during a *superstep:*
  - A '*compute*' function is invoked on each vertex that received a message in the previous superstep
  - Next superstep begins *only* after all vertices have completed their work
  - If no messages are in flight, halt program

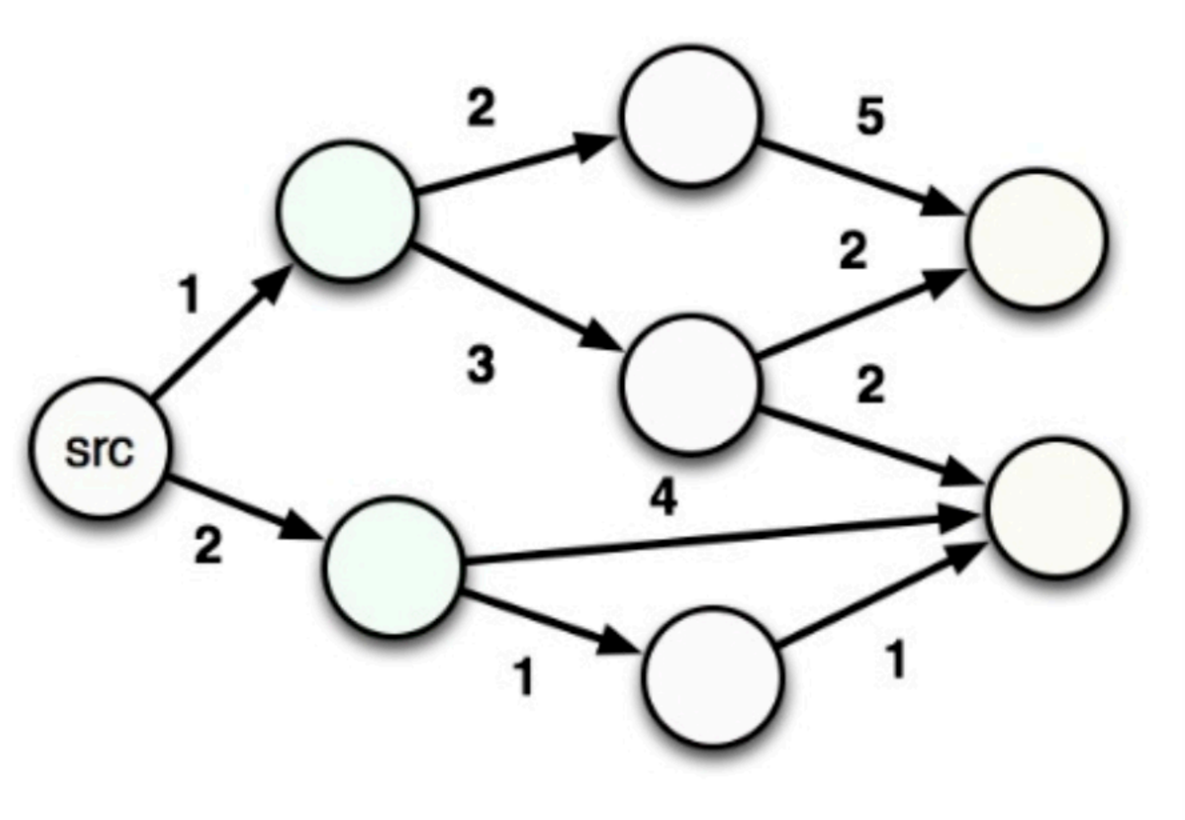
Single Source Shortest Path Algorithm

```
public void compute(Iterable<DoubleWritable> messages) {
  double minDist = Double.MAX_VALUE;
  for (DoubleWritable message : messages) {
    minDist = Math.min(minDist, message.get());
  }
  if (minDist < getValue().get()) {
    setValue(new DoubleWritable(minDist));
    for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
      double distance = minDist + edge.getValue().get();
      sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
    }
  }
  voteToHalt();
}
```

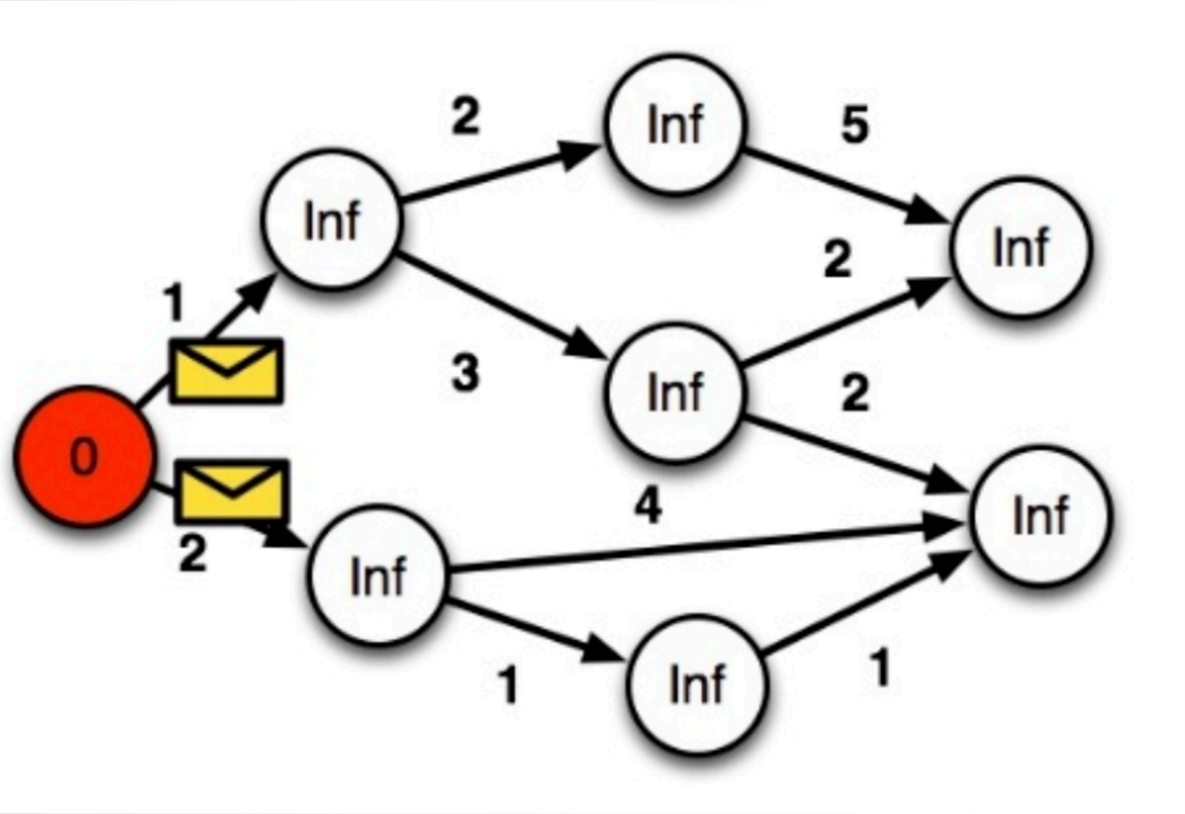
Read updates from other vertices, find minimum

Send distance to other vertices

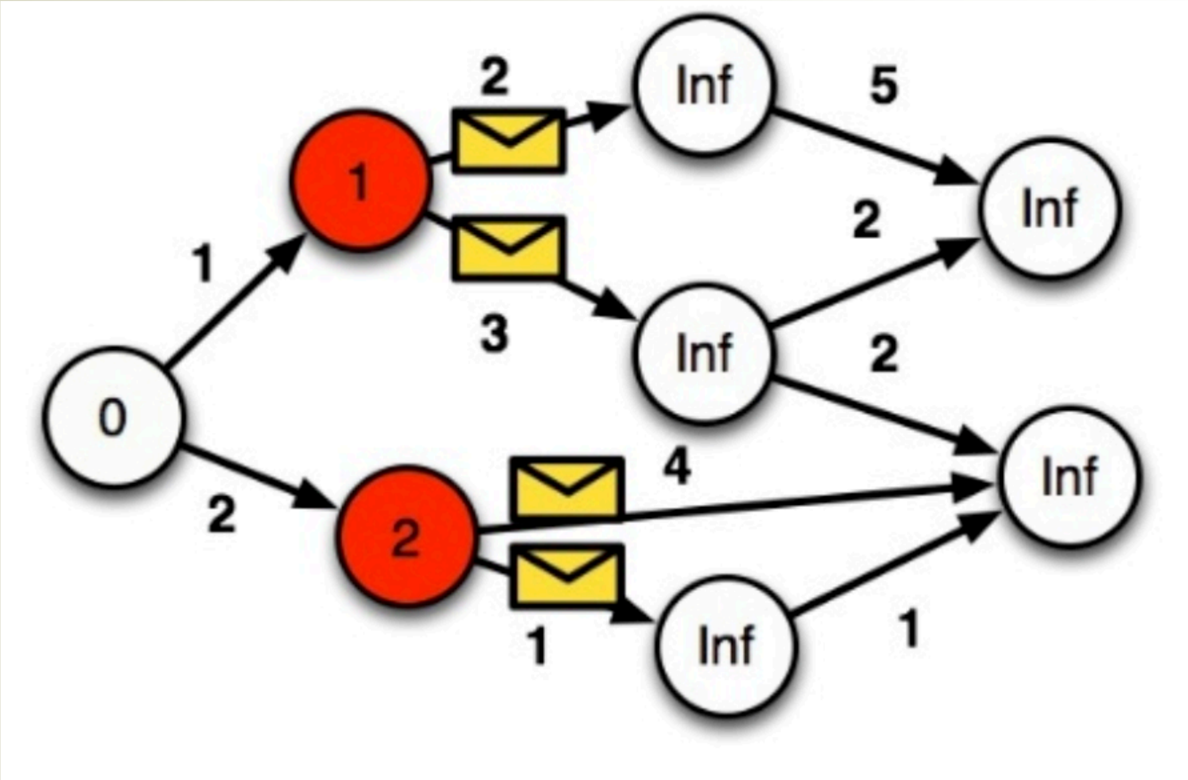
Single Source Shortest Path Example



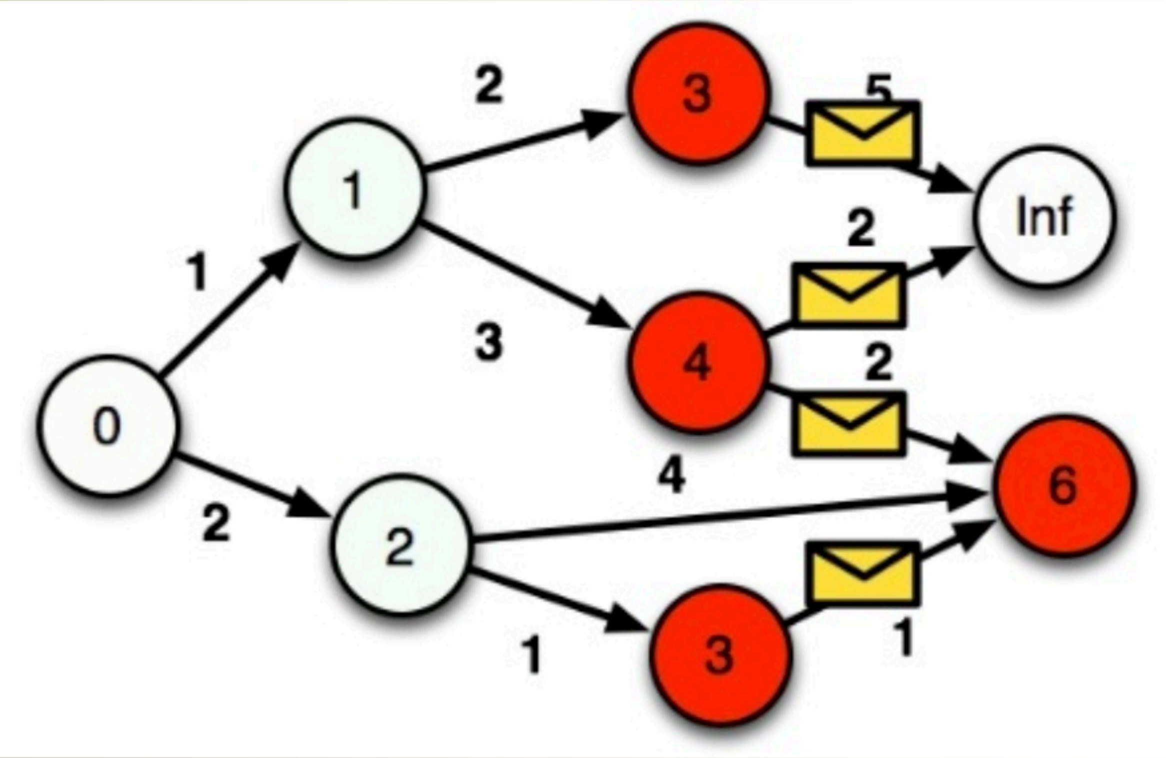
Single Source Shortest Path Example



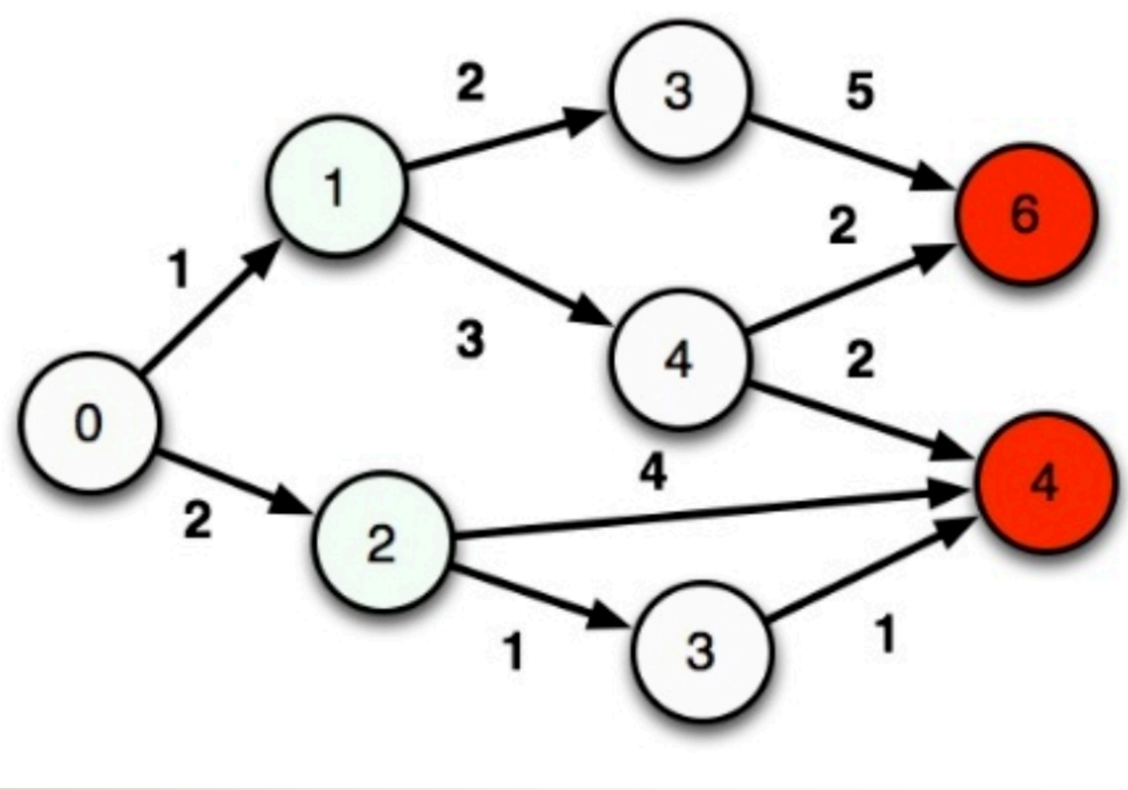
Single Source Shortest Path Example



Single Source Shortest Path Example



Single Source Shortest Path Example



More Complex Example: PageRank

```java
// Essence of PageRank in Giraph
public compute(Iterator<DoubleWritable> msgIterator) {

  if (getSuperstep() >= 1) {
   double sum = 0;
   while (msgIterator.hasNext()) {
    sum += msgIterator.next().get();
   }

   DoubleWritable vertexValue =
     new DoubleWritable((0.15f / vertex.getNumVertices()) + 0.85f * sum);
   vertex.setVertexValue(vertexValue);
  }

  if (getSuperstep() < getConf().getInt(SUPERSTEP_COUNT, -1)) {


   long edges = getNumOutEdges();
   sendMsgToAllEdges(
     new DoubleWritable(vertex.getVertexValue().get() / edges));


  } else {
   voteToHalt();
  }
 }
```

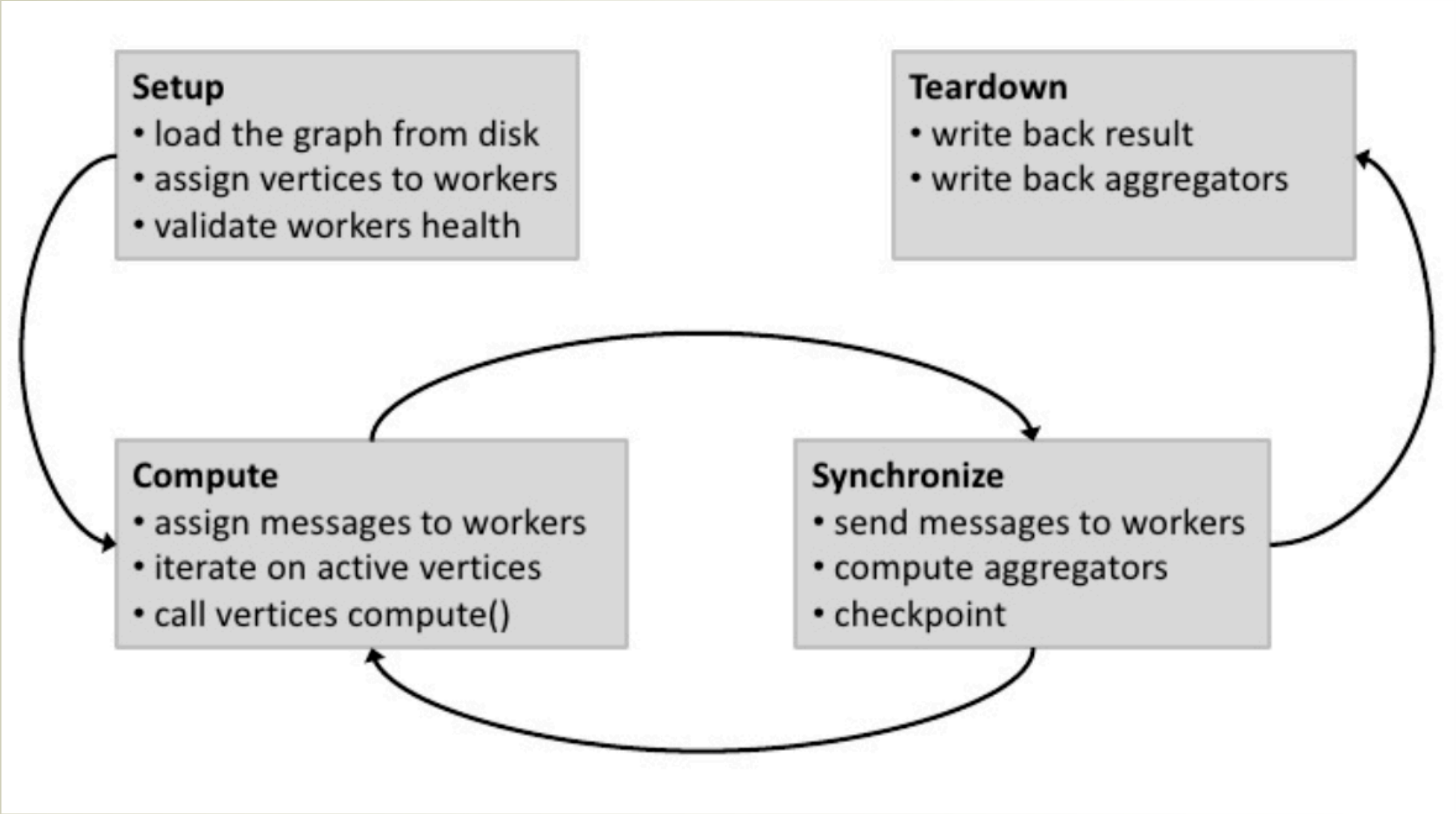
Our neighbors send us their values, we add them up

And compute our new value

Do this a pre-set number of times

And send our new value to everybody else...
or it's time to quit

Giraph Job Lifetime



Implementing Algorithm in Giraph

- Define a *Vertex* class
  - Subclass of existing implementations
- Define a *VertexInputFormat* to read the graph
- Define *VertexOutputFormat* that defines how to extract result based on Vertex final state
- Many other features can be utilized to improve performance

Aggregators

- Each vertex can store values that can be read by all vertices in proceeding superstep
- Can maintain values (sum, min, max, accumulate, user defined, ect)
- Aggregators must be registered on master

Combiners

- User defined function to combine messages before being sent or delivered
- Saves on network and memory

Checkpointing

- Can be expensive but necessary
- Ensures no single point of failure
- Store work at user defined intervals
- Restart on failure

Zookeeper Responsibilities: Computation State

- Handles partition/worker mapping
- Global state
- Checkpoint paths, aggregator values, statistics

Master Responsibilties: Coordination

- Assigns partitions to workers
  - Hashmapping is default
  - Can be user defined
- Monitors workers
- Coordinates supersteps (ending, starting ect)

Worker Responsibilities: Vertices

- Workers are assigned vertices
- Perform compute
- Pass messages between vertices
- Computes local aggregation values