CSE6331: Cloud Computing

Leonidas Fegaras

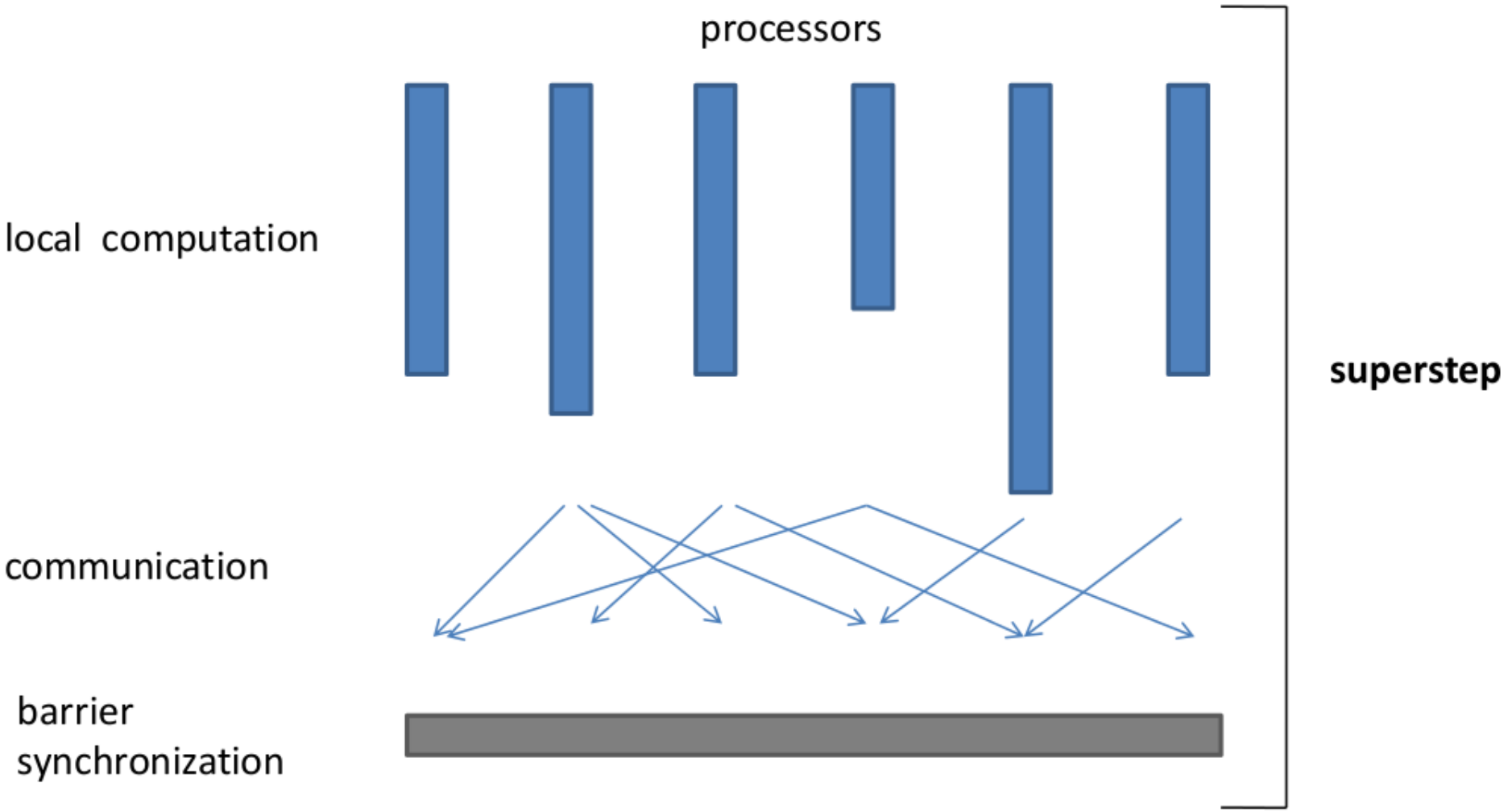
University of Texas at Arlington

Graph Processing with Pregel and Giraph

Vertex-Centric Distributed Graph Processing

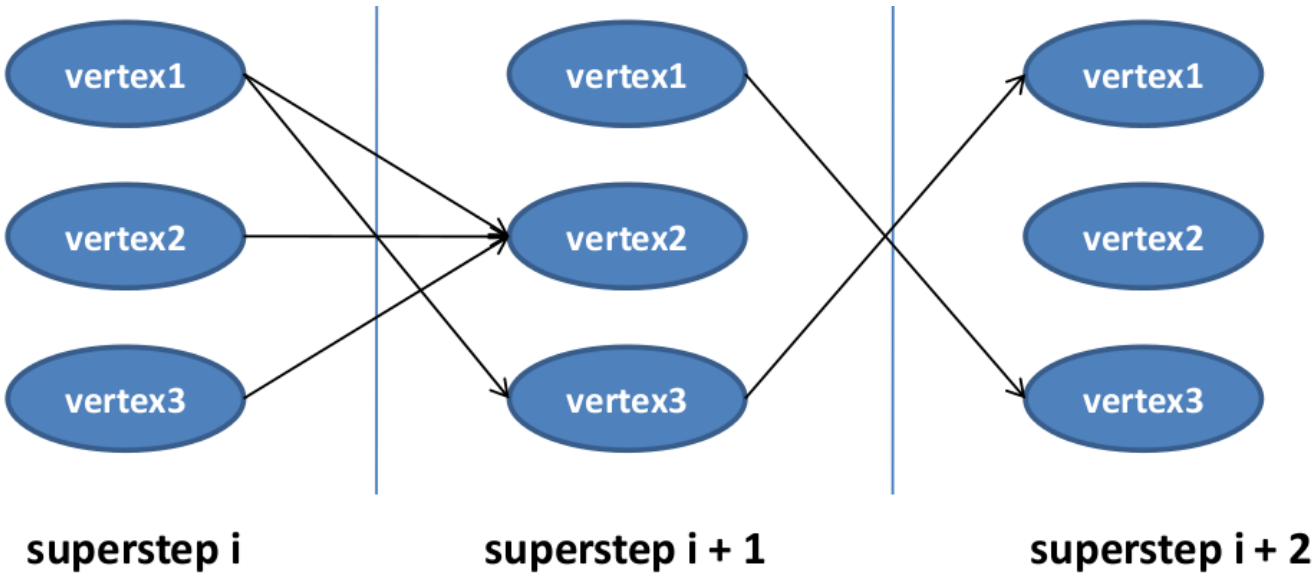
- Google's Pregel
  - Distributed system for large scale graph processing
  - Vertex-centric graph processing
  - Focuses on independent local actions on graph vertices
  - Vertex-centric = "think like a vertex"
  - Fault-tolerant
  - Bulk Synchronous Parallel (BSP) as execution model
- Apache Giraph is an open source implementation of Pregel
  - Part of Hadoop ecosystem
  - In-memory distributed execution based on BSP
  - Fault-tolerance by checkpointing
- GraphX
  - Spark's API for graphs and graph-parallel computation

The Bulk Synchronous Parallel (BSP) Execution Model



Vertex-Centric Computations

- Each vertex has an id, a vertex value, a list of edges to adjacent neighbors
- Each edge has an id (the destination id) and an edge value
- Each vertex used at each superstep can recompute its vertex value and send messages to other vertices, which are delivered before the next superstep
- Additional features: termination votes, combiners, aggregators, topology mutations



Model of Computation

- Computation proceeds in supersteps
  - In a superstep, vertices execute the same user-defined function *compute* in parallel
- In a superstep, a vertex can:
  - Modify its vertex value
  - Modify the value of its outgoing edges
  - Receive messages sent to it in the previous superstep
  - Send messages to other vertices to be received in the next superstep
  - Add/remove outgoing edges
- Edges may have value, but have no associated computation
- This is a pure message-passing model: no remote reads or shared memory

PageRank in Pregel

- P_i: PageRank of node i
- C_i: number of outgoing links from node i

$$P_i = \sum_{j \to i} \frac{P_j}{C_j}$$

```
class PageRankVertex {
    void compute ( Iterator messages ) {
        if (getSuperstep() > 0)
            setVertexValue(sum(messages));
        if (getSuperstep() < k)
            sendMessageToAllNeighbors(pageRank / getNumOutEdges());
        else voteToHalt();
    }
}
```

The Giraph API

- Need to implement the Vertex<I,V,E,M> interface
  - I - Vertex id (WritableComparable)
  - V - Vertex data (Writable)
  - E - Edge data (Writable)
  - M - Message data (Writable)

void addEdge(Edge<I,E> edge)	Add an edge to this vertex
Iterable <Edge<I,E>> getEdges()	Get the out-edges of this vertex
E getEdgeValue(I vid)	Get the edge value with the given target vertex id
I getId()	Get the vertex id
V getValue()	Get the vertex value
void removeEdges(I vid)	Remove all edges pointing to the given vertex id
void setEdges (Iterable <Edge<I,E>> edges)	Set the outgoing edges for this vertex
void setEdgeValue(I vid, E value)	Set the edge value of the edge to the target vid
void setValue(V value)	Set the vertex data
void voteToHalt()	Halt untl a message is sent to this vertex
void sendMessage(I vid,M m)	Send a message m to the node vid
void sendMsgToAllEdges(M m)	Send a message m to all outgoing edges

Connected Components in Giraph

```
public void compute ( Iterator<IntWritable> messages ) {
    int m = getValue().get();
    for ( IntWritable message : messages )
        m = Math.min(m,messages.next().get());
    if (getSuperstep() == 0 || m != getValue().get())
        setValue(new IntWritable(m));
    sendMsgToAllEdges(m);
    voteToHalt();
}
```

Every vertex votes to halt on every superstep, but

1. if there is at least one vertex whose value changed, it will send messages to neighbors
2. if a vertex receives a message, it will wake up and continue, although it has voted to halt

Single Source Shortest Paths in Giraph

1. find the minimum value arriving on any message
2. if that value is less than the current value of the vertex
3. the minimum will be adopted as the vertex value
4. the value plus the edge value will be sent along every outgoing edge

```
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();
  }
```