

Large Scale Graph Processing with Apache Giraph

Sebastian Schelter

Invited talk at GameDuell Berlin
29th May 2012

the mandatory ,about me' slide

- PhD student at the **Database Systems and Information Management Group (DIMA)** of TU Berlin
 - **Stratosphere**, database inspired approach to a next generation large scale processing system, joint research project with HU Berlin and HPI Potsdam
 - European Research Project 'ROBUST' dealing with the **analysis of huge-scale online business communities**
- involved in open source as committer and PMC member of **Apache Mahout** and **Apache Giraph**



Overview

1) Graphs

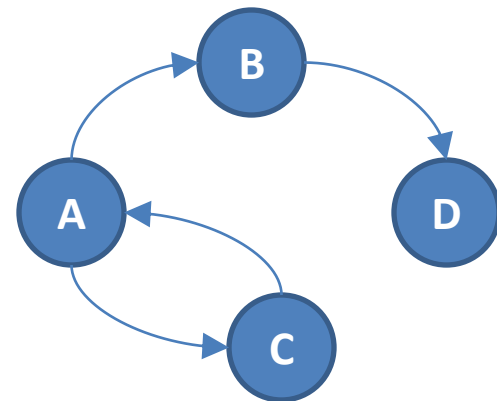
- 2) Graph processing with Hadoop/MapReduce
- 3) Google Pregel
- 4) Apache Giraph
- 5) Outlook: everything is a network

Graph recap

graph: abstract representation of a set of objects (**vertices**), where some pairs of these objects are connected by links (**edges**), which can be directed or undirected

Graphs can be used to model arbitrary things like road networks, social networks, flows of goods, etc.

Majority of graph algorithms are iterative and traverse the graph in some way

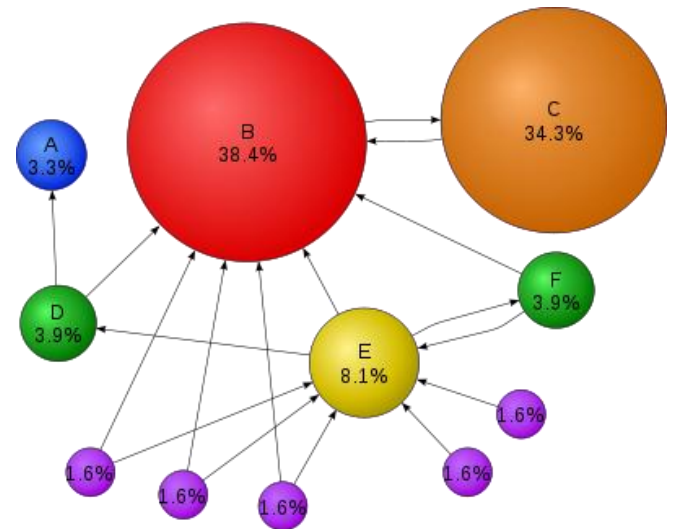


The Web

- the World Wide Web itself can be seen as a huge graph, the so called **web graph**
 - **pages are vertices** connected by **edges that represent hyperlinks**
 - the web graph has **several billion vertices** and **several billion edges**
- the success of major internet companies such as Google is based on the ability to conduct computations on this huge graph

Google's PageRank

- success factor of Google's search engine:
 - much **better ranking of search results**
- ranking is based on **PageRank**, a graph algorithm computing the 'importance' of webpages
 - simple idea: **look at the structure of the underlying network**
 - important pages have a lot of links from other important pages
- major technical success factor of Google: ability to conduct **web scale graph processing**



Social Networks

- on facebook, twitter, LinkedIn, etc, the users and their interactions form a **social graph**
 - **users are vertices** connected by **edges that represent some kind of interaction** such as friendship, following, business contact
- fascinating research questions:
 - what is the structure of these graphs?
 - how do they evolve over time?
- analysis requires knowledge in both computer science and social sciences



six degrees of separation

- **small world problem**
 - through how many social contacts do people know each other on average?
- **small world experiment** by Stanley Milgram
 - task: deliver a letter to a recipient whom you don't know personally
 - you may forward the letter only to persons that you know on a first-name basis
 - how many contacts does it take on average until the letter reaches the target?
- **results**
 - it took 5.5 to 6 contacts on average
 - confirmation of the popular assumption of **'six degrees of separation'** between humans
 - experiment criticised due to small number of participants, possibly biased selection



four degrees of separation

- the small world problem as a graph problem in social network analysis
 - **what is the average distance between two users in a social graph?**
 - in early 2011, scientists conducted a **world scale experiment** using the Facebook social graph
 - 721 million users, 69 billion friendships links
 - result: average distance in Facebook is 4.74
 - **‘four degrees of separation’**
- large scale graph processing gives unprecedented opportunities for the social sciences

Overview

- 1) Graphs
- 2) Graph processing with Hadoop/MapReduce**
- 3) Google Pregel
- 4) Apache Giraph
- 5) Outlook: everything is a network

Why not use MapReduce/Hadoop?

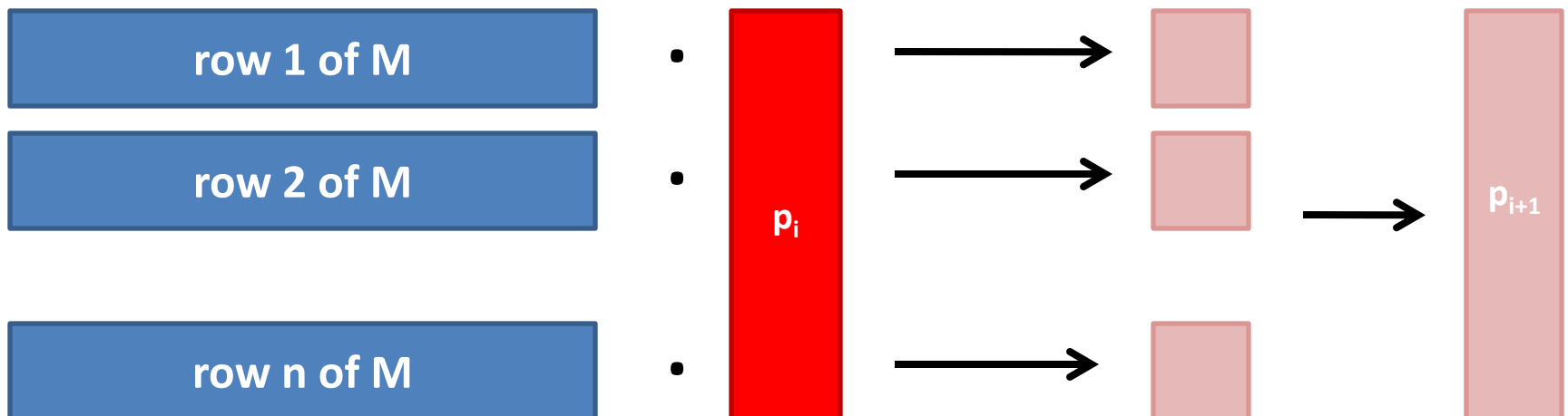
- MapReduce/Hadoop is the current standard for data intensive computing, why not use it for graph processing?
- Example: **PageRank**
 - defined **recursively**
 - each vertex distributes its authority to its neighbors in equal proportions

$$p_i = \sum_{j \in \{(j,i)\}} \frac{p_j}{d_j}$$

Textbook approach to PageRank in MapReduce

- PageRank p is the **principal eigenvector** of the Markov matrix M defined by the **transition probabilities between web pages**
- it can be obtained by iteratively multiplying an initial PageRank vector by M (**power method**)

$$p_{i+1} = Mp_i$$



Drawbacks

- **Not intuitive:** only crazy scientists think in matrices and eigenvectors
- **Unnecessarily slow:** Each iteration is a single MapReduce job with lots of overhead
 - separately scheduled
 - the graph structure is read from disk
 - the intermediary result is written to HDFS
- **Hard to implement:** a join has to be implemented by hand, lots of work, best strategy is data dependent



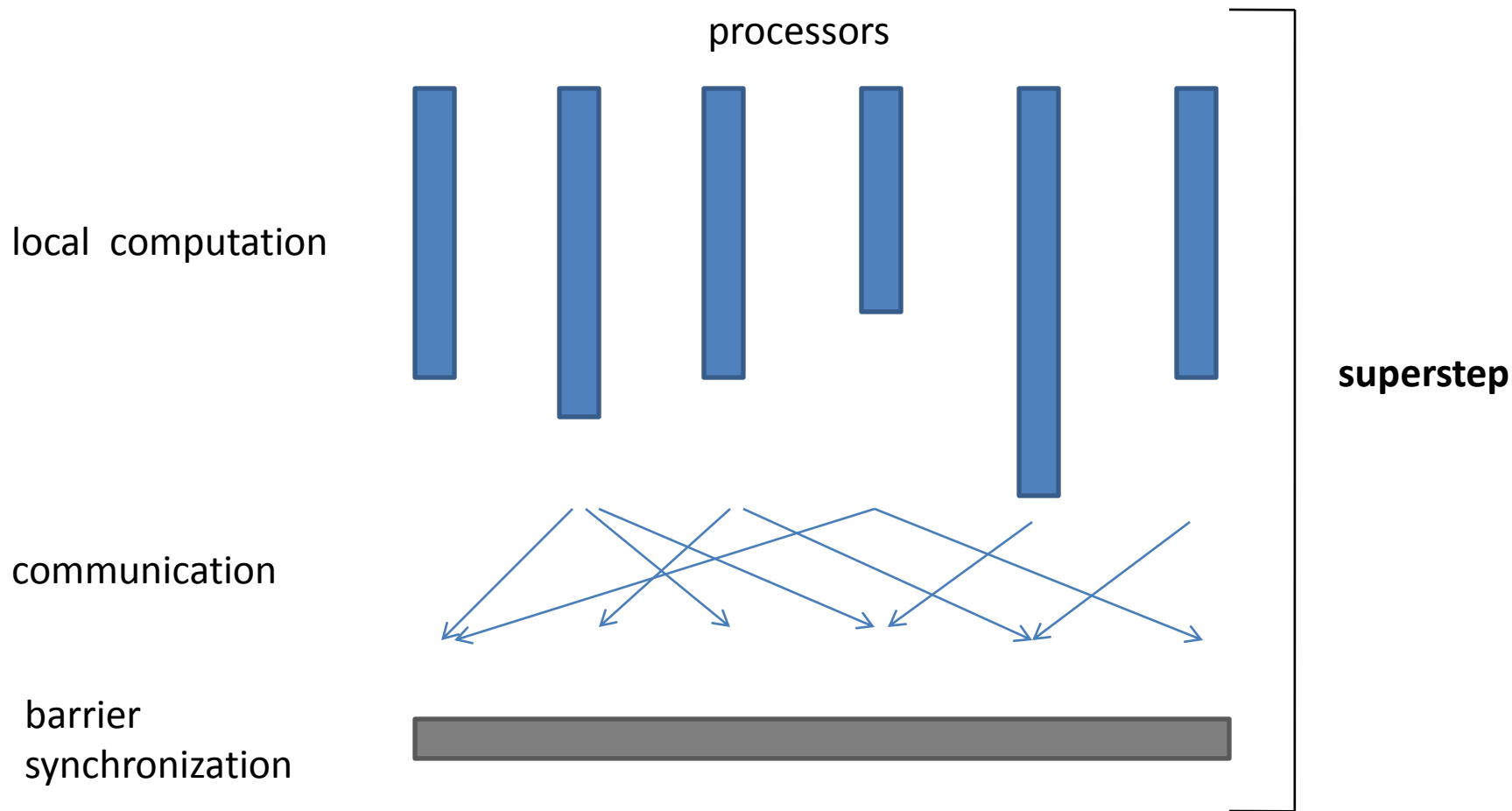
Overview

- 1) Graphs
- 2) Graph processing with Hadoop/MapReduce
- 3) Google Pregel**
- 4) Apache Giraph
- 5) Outlook: everything is a network

Google Pregel

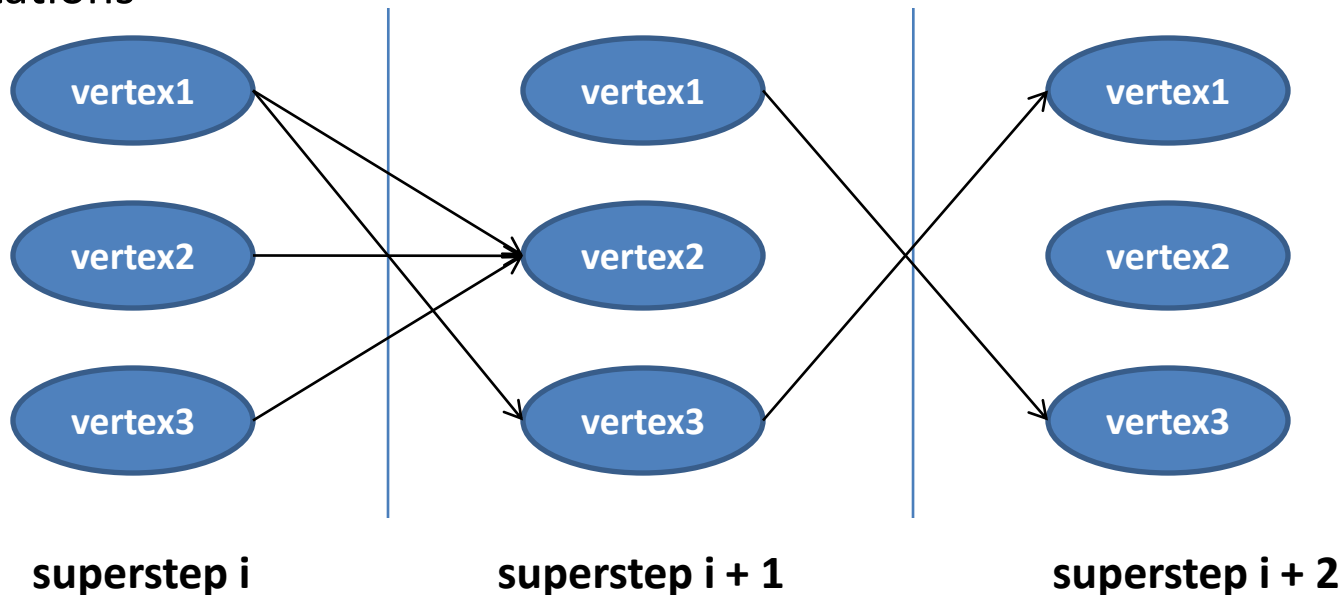
- distributed system especially developed for **large scale graph processing**
- intuitive API that let's you **,think like a vertex'**
- **Bulk Synchronous Parallel (BSP)** as execution model
- fault tolerance by **checkpointing**

Bulk Synchronous Parallel (BSP)



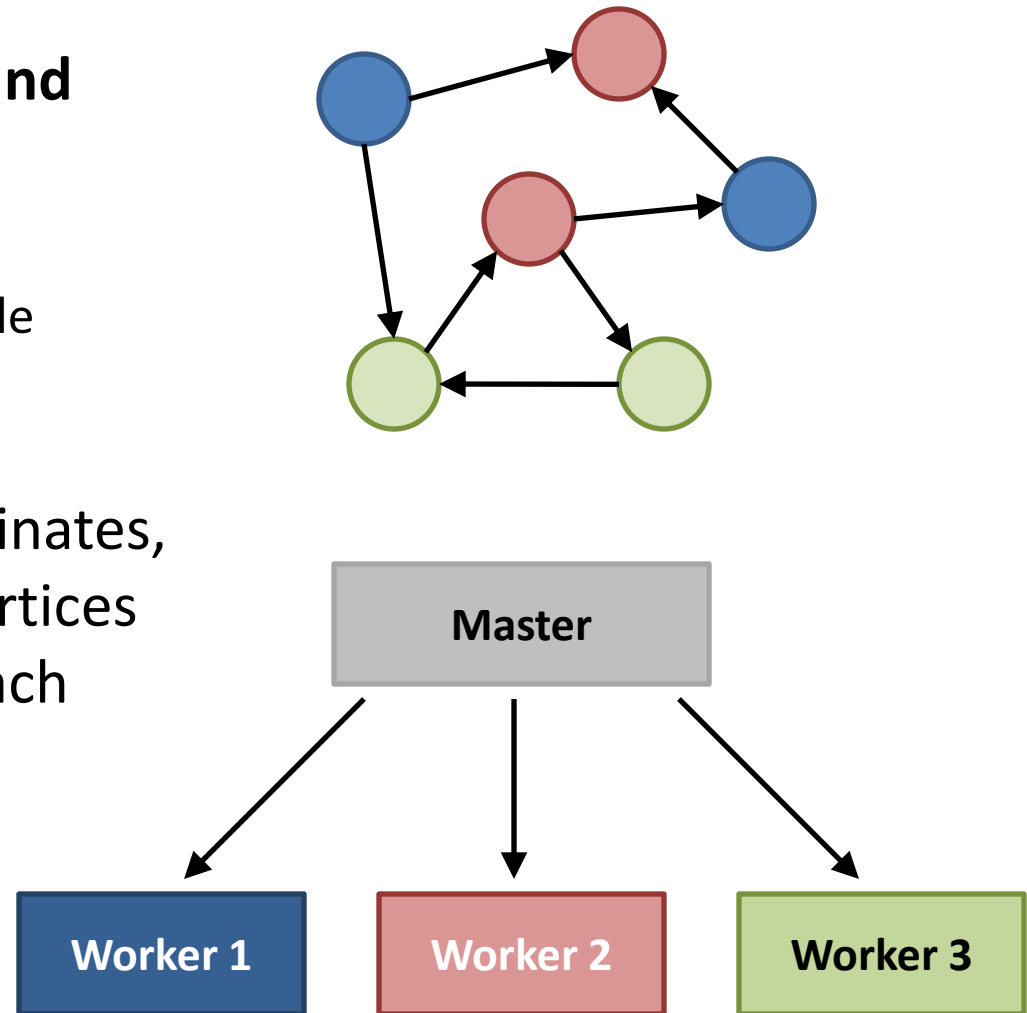
Vertex-centric BSP

- each vertex has an **id**, a **value**, a **list of its adjacent neighbor ids** and the corresponding **edge values**
- each vertex is invoked **in each superstep**, can **recompute its value** and **send messages to other vertices**, which are **delivered over superstep barriers**
- advanced features : **termination votes**, **combiners**, **aggregators**, **topology mutations**



Master-slave architecture

- **vertices are partitioned and assigned to workers**
 - default: hash-partitioning
 - custom partitioning possible
- **master** assigns and coordinates, while **workers** execute vertices and communicate with each other

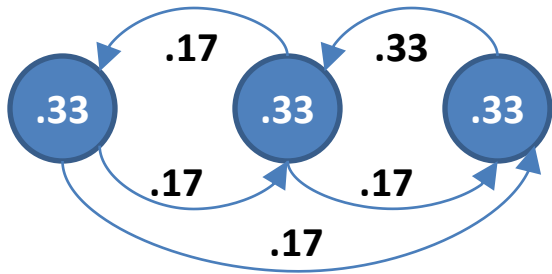


PageRank in Pregel

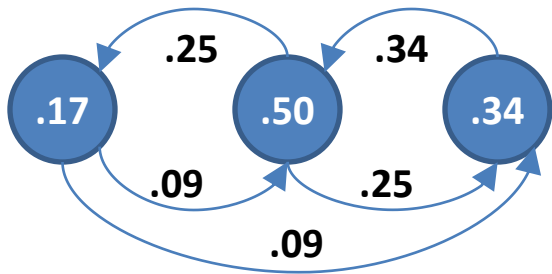
```
class PageRankVertex {  
  void compute(Iterator messages) {  
    if (getSuperstep() > 0) {  
      // recompute own PageRank from the neighbors messages  
      pageRank = sum(messages);  
      setVertexValue(pageRank);  
    }  
  
    if (getSuperstep() < k) {  
      // send updated PageRank to each neighbor  
      sendMessageToAllNeighbors(pageRank / getNumOutEdges());  
    } else {  
      voteToHalt(); // terminate  
    }  
  }  
}
```

$$p_i = \sum_{j \in \{(j,i)\}} \frac{p_j}{d_j}$$

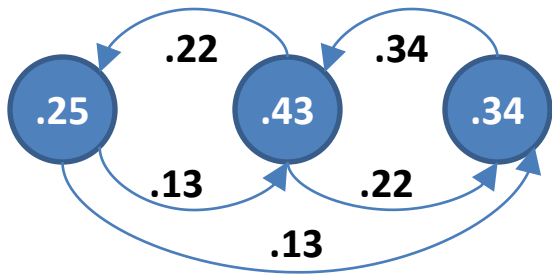
PageRank toy example



Superstep 0

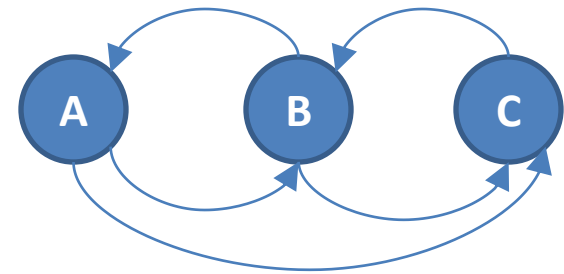


Superstep 1



Superstep 2

Input graph



Cool, where can I download it?

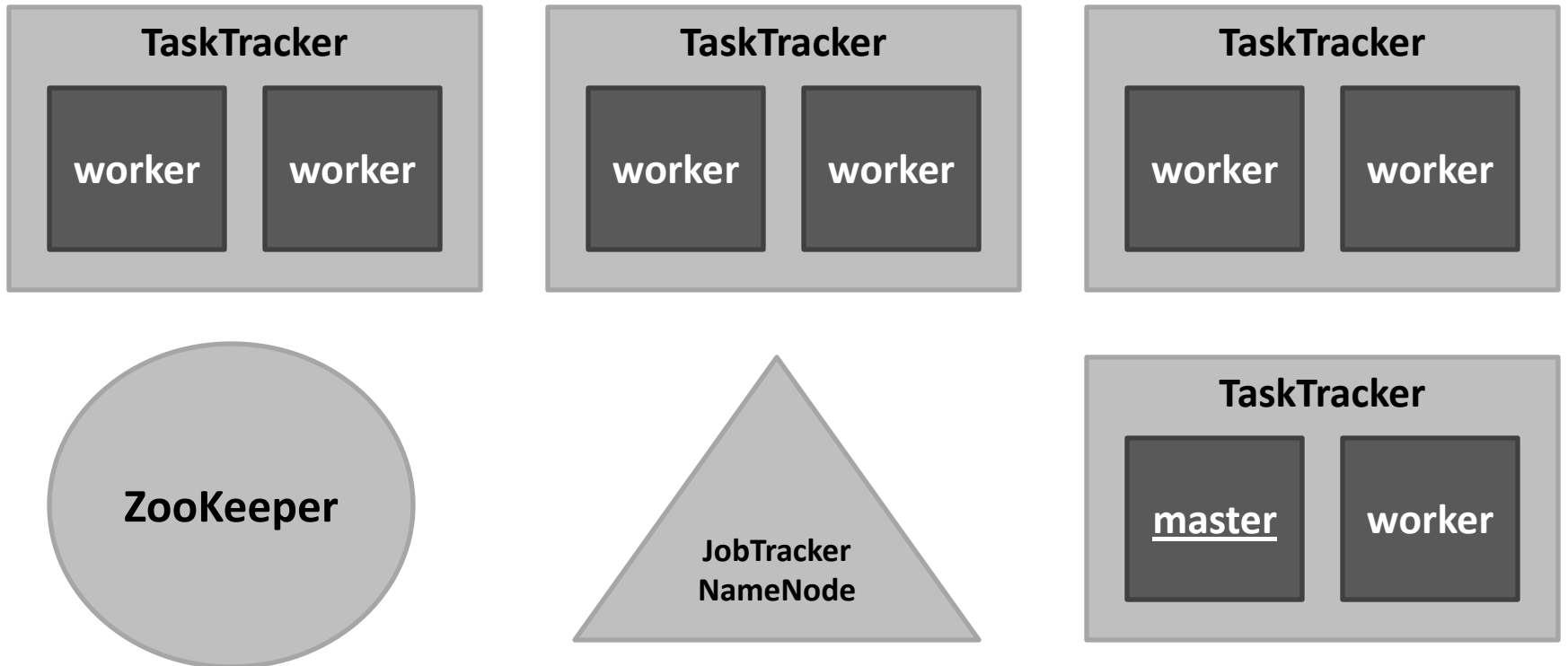
- Pregel is **proprietary**, but:
 - **Apache Giraph** is an open source implementation of Pregel
 - runs on **standard Hadoop infrastructure**
 - computation is executed **in memory**
 - can be a job in a **pipeline** (MapReduce, Hive)
 - uses **Apache ZooKeeper** for synchronization



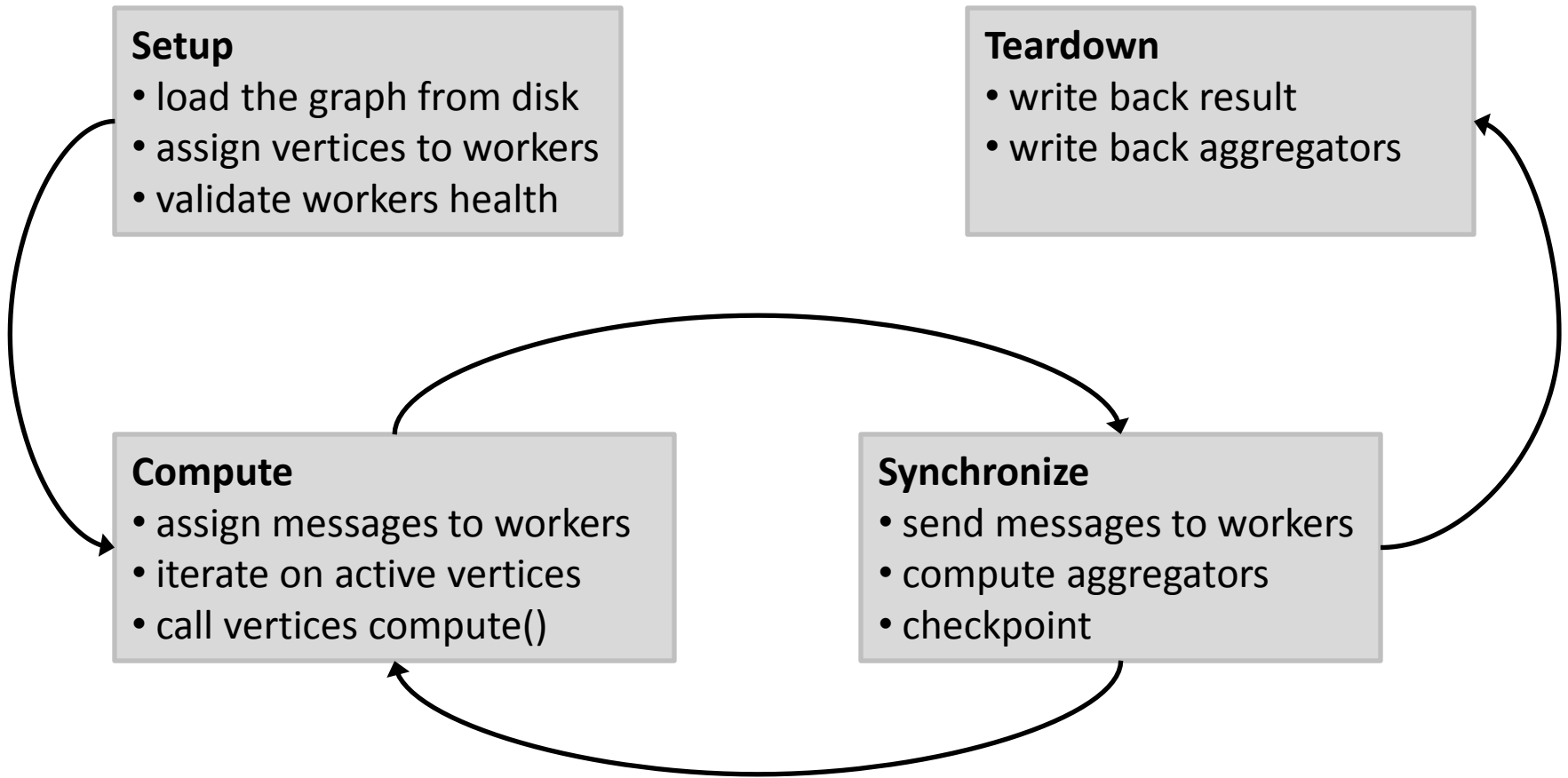
Overview

- 1) Graphs
- 2) Graph processing with Hadoop/MapReduce
- 3) Google Pregel
- 4) Apache Giraph**
- 5) Outlook: everything is a network

Giraph's Hadoop usage



Anatomy of an execution

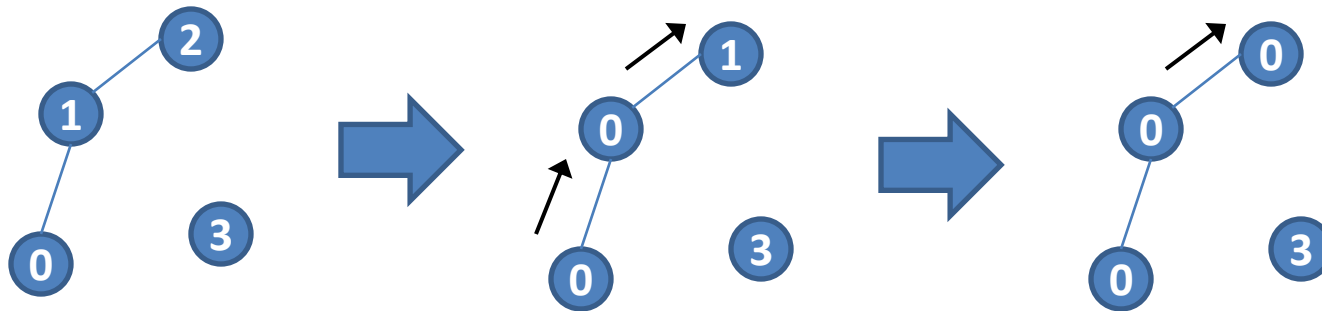


Who is doing what?

- **ZooKeeper:** responsible for **computation state**
 - partition/worker mapping
 - global state: #superstep
 - checkpoint paths, aggregator values, statistics
- **Master:** responsible for **coordination**
 - assigns partitions to workers
 - coordinates synchronization
 - requests checkpoints
 - aggregates aggregator values
 - collects health statuses
- **Worker:** responsible for **vertices**
 - invokes active vertices compute() function
 - sends, receives and assigns messages
 - computes local aggregation values

Example: finding the connected components of an undirected graph

- algorithm: propagate smallest vertex label to neighbors until convergence



- in the end, all vertices of a component will have the same label

Step 1: create a custom vertex

```
public class ConnectedComponentsVertex
    extends BasicVertex<IntWritable, IntWritable, NullWritable, IntWritable> {

    public void compute(Iterator messages) {
        int currentLabel = getVertexValue().get();
        while (messages.hasNext()) {
            int candidate = messages.next().get();
            currentLabel = Math.min(currentLabel, candidate); // compare with neighbors labels
        }
        // propagation is necessary if we are in the first superstep or if we found a new label
        if (getSuperstep() == 0 || currentLabel != getVertexValue().get()) {
            setVertexValue(new IntWritable(currentLabel));
            sendMsgToAllEdges(getVertexValue()); // propagate newly found label to neighbors
        }
        voteToHalt(); // terminate this vertex, new messages might reactivate it
    }
}
```

Step 2: create a custom input format

- input is a text file with adjacency lists, each line looks like: *<vertex_ID> <neighbor1_ID> <neighbor2_ID> ...*

```
public class ConnectedComponentsInputFormat extends
    TextVertexInputFormat<IntWritable, IntWritable, NullWritable, IntWritable> {
    static class ConnectedComponentsVertexReader extends
        TextVertexReader<IntWritable, IntWritable, NullWritable, IntWritable> {
    public BasicVertex<IntWritable, IntWritable, NullWritable, IntWritable> getCurrentVertex() {
        // instantiate vertex
        BasicVertex<IntWritable, IntWritable, NullWritable, IntWritable> vertex = ...
        // parse a line from the input and initialize the vertex
        vertex.initialize(...);
        return vertex;
    }
}}
```

Step 3: create a custom output format

- output is a text file, each line looks like:
<vertex_ID> <component_ID>

```
public class VertexWithComponentTextOutputFormat extends
    TextVertexOutputFormat<IntWritable, IntWritable, NullWritable> {
    public static class VertexWithComponentWriter extends
        TextVertexOutputFormat.TextVertexWriter<IntWritable, IntWritable, NullWritable> {
        public void writeVertex(BasicVertex<IntWritable, IntWritable, NullWritable, ?> vertex) {
            // write out the vertex ID and the vertex value
            String output = vertex.getVertexId().get() + '\t' + vertex.getVertexValue().get();
            getRecordWriter().write(new Text(output), null);
        }
    }
}
```

Step 4: create a combiner (optional)

- we are only interested in the smallest label sent to a vertex, therefore we can apply a combiner

```
public class MinimumIntCombiner extends VertexCombiner<IntWritable, IntWritable> {  
    public Iterable<IntWritable> combine(IntWritable target, Iterable<IntWritable> messages) {  
        int minimum = Integer.MAX_VALUE;  
        // find minimum label  
        for (IntWritable message : messages) {  
            minimum = Math.min(minimum, message.get());  
        }  
        return Lists.<IntWritable>newArrayList(new IntWritable(minimum));  
    }  
}
```

Experiments

- Setup: 6 machines with 2x 8core Opteron CPUs, 4x 1TB disks and 32GB RAM each, ran 1 Giraph worker per core
 - Input: Wikipedia page link graph (6 million vertices, 200 million edges)

 - PageRank on Hadoop/Mahout
 - 10 iterations approx. 29 minutes
 - average time per iteration: approx. 3 minutes

 - PageRank on Giraph
 - 30 iterations took approx. 15 minutes
 - average time per iteration: approx. 30 seconds
- 10x performance improvement

hardware utilization

CPU's Total: **112**
 Hosts up: **7**
 Hosts down: **0**

Current Load Avg (15, 5, 1m):
72%, 122%, 134%
 Avg Utilization (last hour):
27%

Localtime:
2012-05-18 10:36

Hadoop Cluster Load last hour

1-min	Now: 148.4	Min: 2.6	Avg: 30.0	Max: 160.3
Nodes	Now: 7.0	Min: 7.0	Avg: 7.0	Max: 7.0
CPUs	Now: 112.0	Min: 112.0	Avg: 112.0	Max: 112.0
Procs	Now: 220.5	Min: 0.0	Avg: 33.4	Max: 256.3

Hadoop Cluster Memory last hour

Use	Now: 117.9G	Min: 23.4G	Avg: 39.2G	Max: 117.9G
Share	Now: 0.0	Min: 0.0	Avg: 0.0	Max: 0.0
Cache	Now: 109.9G	Min: 109.9G	Avg: 189.4G	Max: 205.4G
Buffer	Now: 598.9M	Min: 572.4M	Avg: 589.3M	Max: 600.6M
Swap	Now: 3.2G	Min: 3.2G	Avg: 3.2G	Max: 3.2G
Total	Now: 236.2G	Min: 236.2G	Avg: 236.2G	Max: 236.2G

Hadoop Cluster CPU last hour

User	Now: 76.0%	Min: 0.0%	Avg: 14.1%	Max: 77.6%
Nice	Now: 0.0%	Min: 0.0%	Avg: 0.0%	Max: 0.0%
System	Now: 1.3%	Min: 0.0%	Avg: 0.3%	Max: 2.3%
Wait	Now: 0.0%	Min: 0.0%	Avg: 0.0%	Max: 0.2%
Idle	Now: 22.7%	Min: 21.0%	Avg: 85.5%	Max: 99.9%

Hadoop Cluster Network last hour

In	Now: 75.3M	Min: 10.4k	Avg: 13.6M	Max: 76.3M
Out	Now: 74.6M	Min: 4.1k	Avg: 13.6M	Max: 75.8M

Show Hosts Scaled: Auto Same None | Hadoop Cluster **load_one** last hour sorted **descending** | Size: medium | Columns: 4 | (0 = metric + reports)

cloud-17

cloud-12

cloud-16

cloud-15

cloud-13

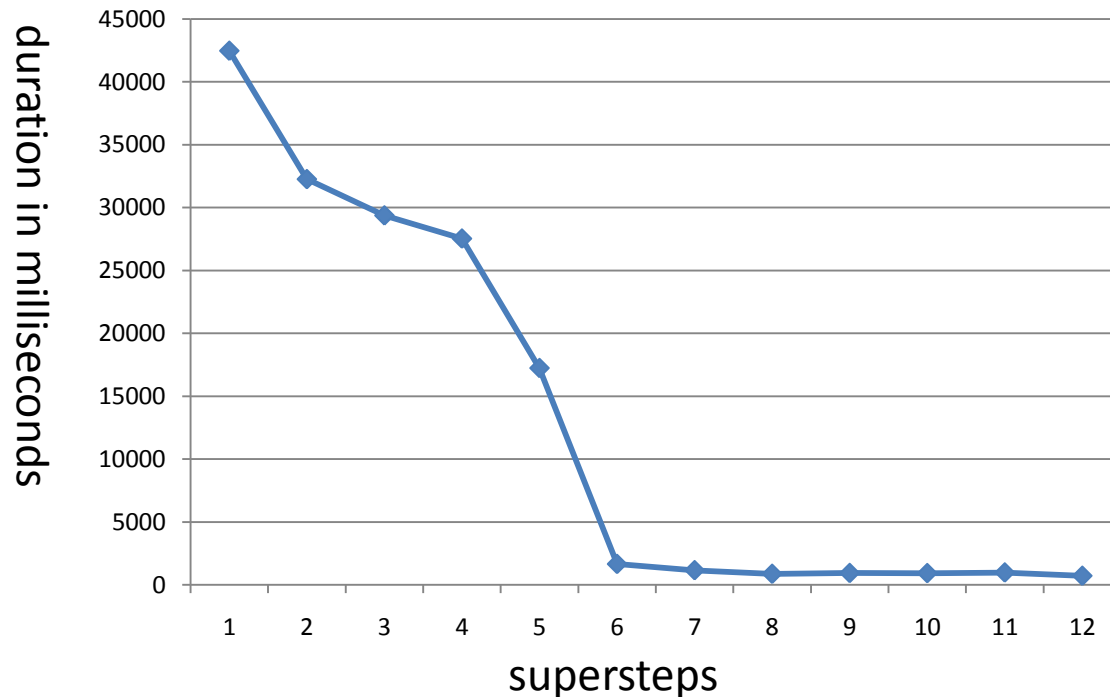
cloud-14

cloud-11

(Nodes colored by 1 minute load) | Legend

Connected Components

- execution takes approx. 4 minutes
 - subsecond iterations after superstep 5
 - Giraph exploits small average distance!



Overview

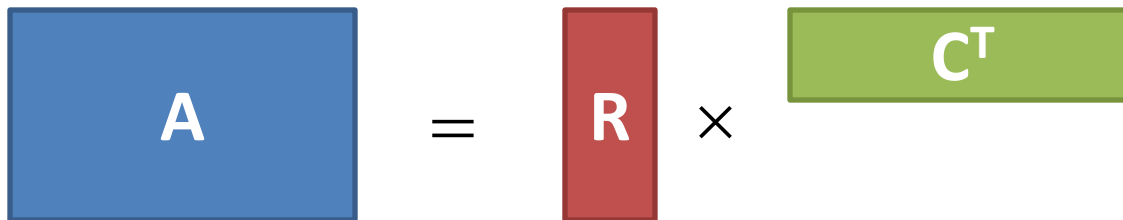
- 1) Graphs
- 2) Graph processing with Hadoop/MapReduce
- 3) Google Pregel
- 4) Apache Giraph
- 5) Outlook: everything is a network**

Everything is a network!



distributed matrix factorization

- **decompose matrix A** into product of **two lower dimensional feature matrices R and C**



The diagram illustrates the matrix factorization equation $A = R \times C^T$. Matrix A is represented by a blue rectangle. Matrix R is represented by a red vertical rectangle. Matrix C^T is represented by a green horizontal rectangle. The equation is shown with an equals sign and a multiplication symbol.

- **master algorithm:** dimension reduction, solving least squares problems, compressing data, collaborative filtering, latent semantic analysis, ...

Alternating Least Squares

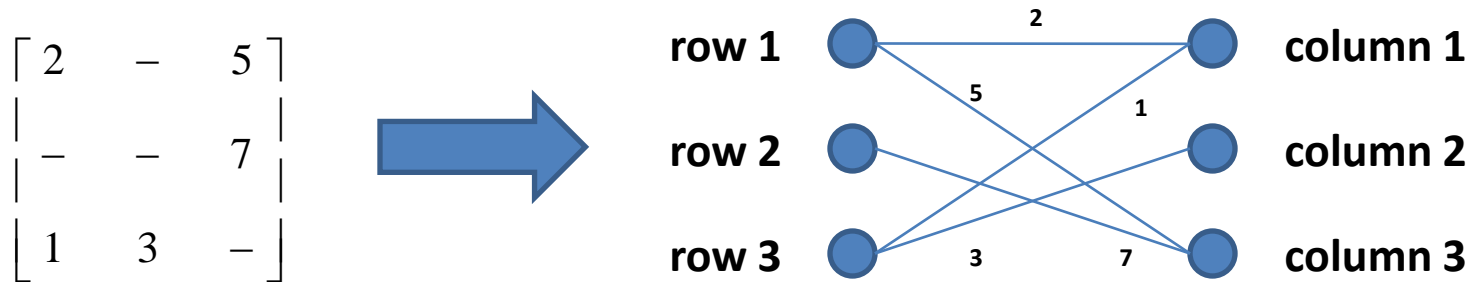
- minimize squared error over all known entries:

$$f(R, C) = \sum_{i, j \in A} (a_{ij} - r_i^T c_j)^2$$

- **Alternating Least Squares**
 - fix one side, solve for the other, repeat until convergence
 - easily parallelizable, iterative algorithm
- what does this all have to do with graphs?

matrices can be represented by graphs

- represent matrix as bipartite graph



- now the ALS algorithm can easily be implemented as a Giraph program
 - every vertex holds a row vector of one of the feature matrices
 - in each superstep the vertices of one side recompute their vector and send it to the connected vertices on the other side

What's to come?

- Current and future work in Giraph
 - out-of-core messaging
 - algorithms library

Thank you. Questions?

Database Systems and Information
Management Group (DIMIA), TU Berlin

