Apache Giraph

Matúš Macko
Martin Schvarcbacher

Overview

- an iterative graph processing framework, built on top of Apache Hadoop
- Released: 2011
- Based on Google’s proprietary Pregel, open-source implementation
- Written entirely in Java

Pregel

- programming model targeted to large-scale graph problems
- message passing between vertices in graph - *supersteps*
- user specified compute function on each vertex
- PageRank implementation is only 15 lines of code in C++
- more - https://kowshik.github.io/JPregel/pregel_paper.pdf

Features Overview

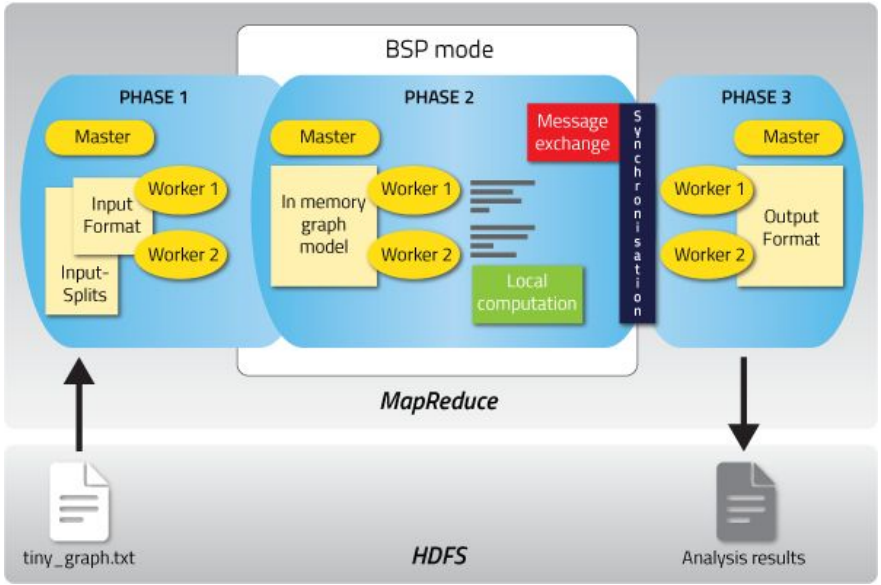
- Master computation - http://giraph.apache.org/implementation.html
- Sharded aggregators - http://giraph.apache.org/aggregators.html
- Edge-oriented input/output - http://giraph.apache.org/io.html
- Out-of-core computation - http://giraph.apache.org/ooc.html

Why select Giraph

- Used internally by Facebook on 4 trillion edges, time to process = 4 minutes
  - since version 1.0.0 Apache Giraph provides all of the described features
  - https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/101516170061539 20/
- Open-source, in active development, regular contributions from FB engineers
- You need to do data ANALYTICS on graphs and not direct graph storage
- Used by: Apache Software Foundation, Facebook

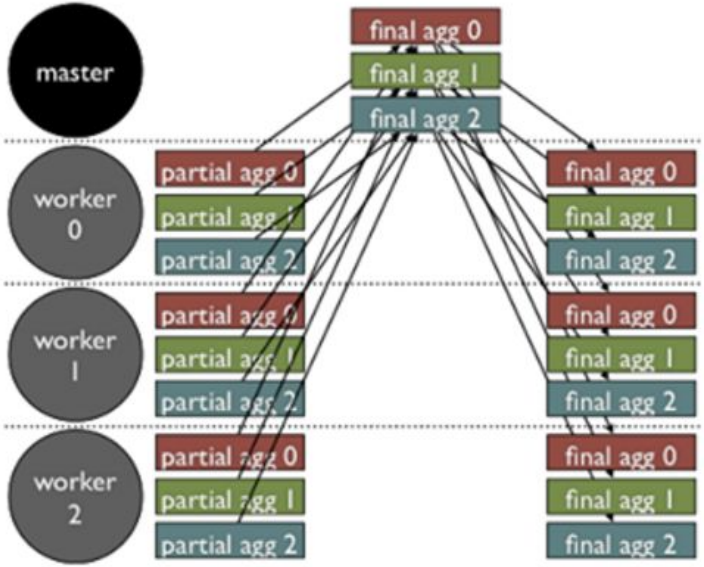
Technology Underneath

- Apache Hadoop:
  - Hadoop File System
    - All data needs to be in HDFS for processing
  - Allows easy task parallelization
  - Giraph jobs run as a MapReduce job
- Java as a primary language for processing



How Apache Giraph works

- all data and workload distribution related details are hidden behind an easy-to-use API
- a worker node or a slave node is a host performing computations and storing data in HDFS
- Giraph algorithm is an iterative execution of “super-steps”
- BSP - Bulk synchronous parallel



Vertex-centered approach

- Iterative model for each connected(nearby) vertices
- All data divided into partitions for iterative and parallel approach
- IBM researchers looking into graph-centric approach (Giraph++)
    - Published a theoretical implementation paper in 2014
    - Progress stalled in 2015
    - Paper link: http://researcher.watson.ibm.com/researcher/files/us-ytian/giraph++.pdf

Downsides of Giraph

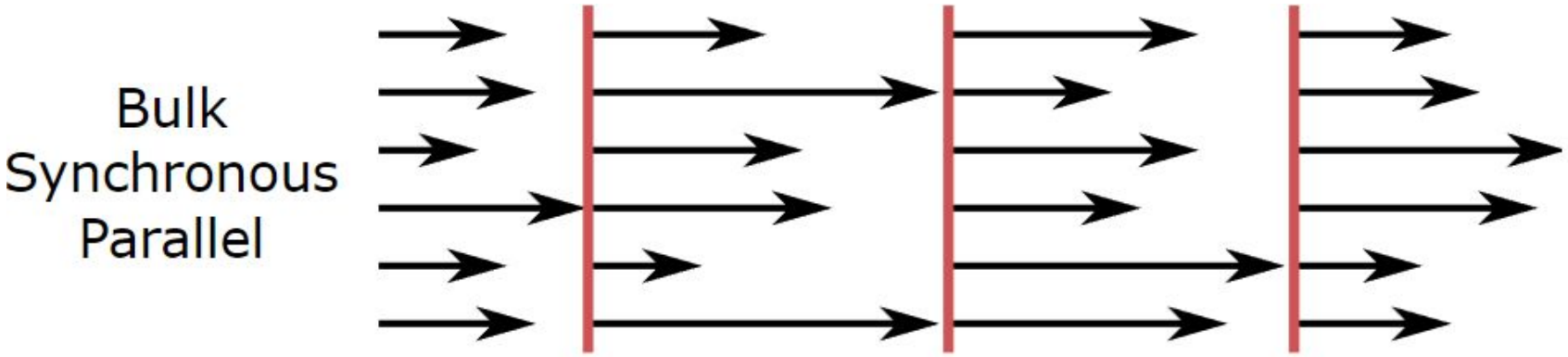
- No custom/embedded query language separate from API, nothing like Neo4J Cypher (yet)
  - **Have to do everything from Java, including the algorithm design**
- Due to MapReduce nature, all data needs to be known beforehand
  - Cannot add data to an ongoing MapReduce job!
- Each new run starts with importing data into Giraph
  - Extra processing time when compared to storing data directly in a graph DB with querying support
- No realtime responses, not interactive
  - Even simple traversals take at least 10-20 seconds
    - Add data to HDFS and distribute
    - Run MapReduce job
    - Get output from HDFS
    - Parse output
- Low quality documentation, JavaDoc often only 1 line to describe a class
- No GUI / web interface

Computation step:

- Computation step:
  - In each superstep, each active vertex invokes the *Compute method* provided by the user.
  - The method implements the graph algorithm that will be executed on the input graph
- The Compute method:
  - receives messages sent to the vertex in the previous superstep,
  - computes using the messages, and the vertex and outgoing edge values,
    - can result in modifications to the values of edges or current vertex
  - may send messages to other vertices.
  - does not have direct access to the values of other vertices and their outgoing edges.
  - Inter-vertex communication occurs by sending messages.

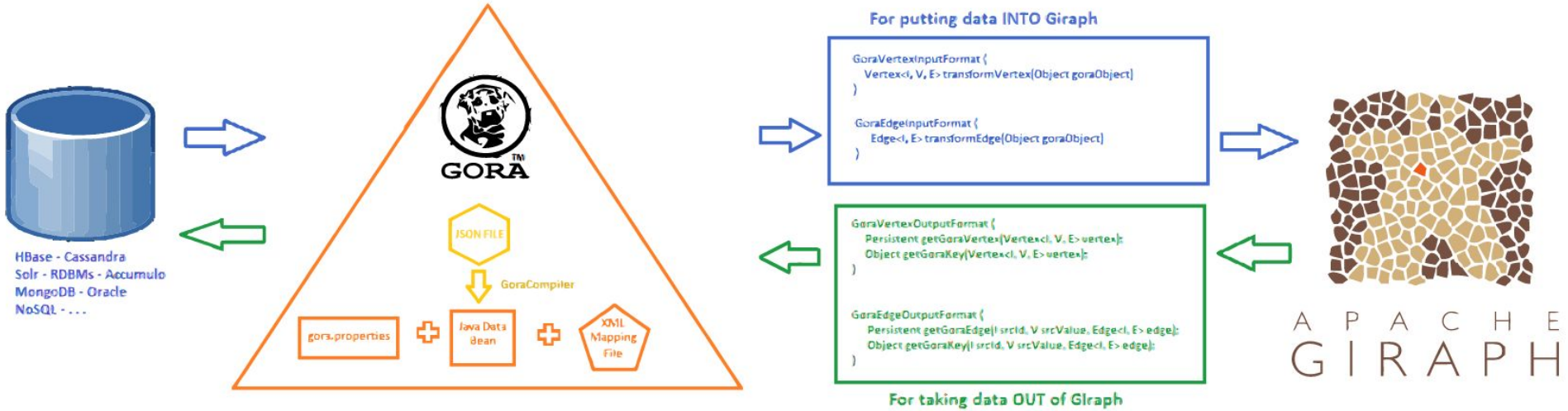
Superstep barrier

- Every operation happens as part of a superstep
- Inside each superstep block all active vertices are processed in parallel
- can process different areas simultaneously
- Vertex updating happens via message passing
- After all active vertices are processed, a new superstep can begin
- Only vertices which have received a message in previous step are activated again
- Once all vertices have halted, overall computation halts and the result is returned
- Uses Bulk Synchronous Parallel Model for synchronization of all vertices



Inputting data into Giraph (1)

- Giraph by itself is **not meant for graph data storage**, only processing and then outputting the resulting data somewhere else
- Data stored in (No)SQL database
- Using Apache Gora for data retrieval and pre-processing:
  - Supports: Column stores, Key-Value stores, Document stores, RDBMS
  - Requires JSON schema for data
  - Processes data from DB into usable format for Giraph



Inputting data into Giraph (2)

- Hadoop input/output formats:
  - Adjacency list with data about the vertex and outgoing edges stored as a string
    - can represent anything: integers, CSV or JSON
  - Requires processing string data at runtime
    - In Java: writing your own InputFormat class to parse the file
    - Returns Java primitives/objects
  - More code to write than Gora, but less technological overhead (only text files needed)
  - Example of input on next slide

Example Input Data

`FORMAT: [Vertex ID, Vertex Data, [[Connected vertex ID, Edge Data]]]`

Adjacency list representation.

Example input (integer weighted edges):

```
[0,0,[[1,1],[3,3]]]
[1,0,[[0,1],[2,2],[3,1]]]
[2,0,[[1,2],[4,4]]]
[3,0,[[0,3],[1,1],[4,4]]]
[4,0,[[3,4],[2,4]]]
```

Java data type: `Vertex<LongWritable, DoubleWritable, DoubleWritable>`

You can always write your own custom (JSON) parser for the data

Edge and Vertex Data

- Every edge and vertex can carry any information
- Graph components must be of homogeneous data types
  - Data type must subclass Writable
  - Vertex, Edge and Message can be of a different type
- Graph data type must be declared in Java before running jobs
  - Specified in Java or as Hadoop input parameter

Java API: VertexInputFormat + VertexOutputFormat

- VertexInputFormat [3]
  - Abstract superclass
  - You need to specify how to handle input data file
  - One line is one vertex, encoded as UTF8 string
  - HDFS: file can be split into multiple chunks, therefore each line needs to be independent
  - All other information encoding left to the user

- VertexOutputFormat [3]
  - Abstract superclass
  - Determines how each vertex with its data and connected edges will be outputted

Message passing

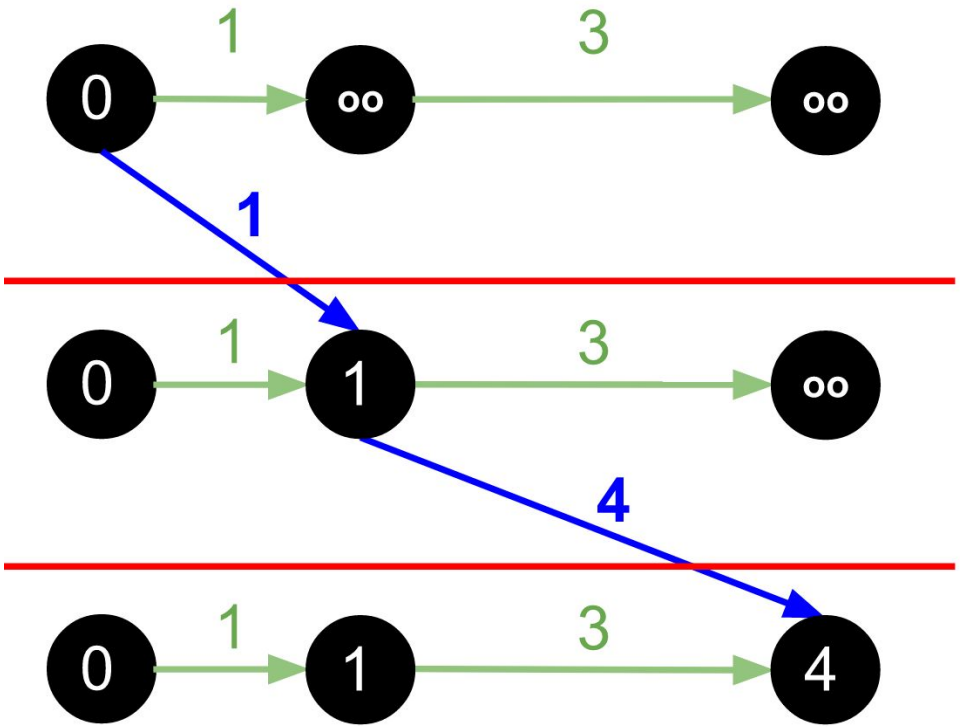
- Sometimes you need to send / share data with vertices not directly connected
- You only have direct access to outbound connected vertices and edge data
- Message passing transparently solves this issue
- Messages are passed for a specific vertex ID
- How it works internally:
  - Edge may be marked as “done” by voteToHalt(), that is it will not be re-computed again unless needed
  - By sending a message it is again marked “not done”
  - In the next computation superstep, all “not done” vertices are computed again
  - All vertices must be halted for Giraph to halt

Simple graph traversal: distance from source

```
1  public void compute(Iterable<DoubleWritable> messages) {
2      double minDist = isSource() ? 0d : Double.MAX_VALUE;
3      for (DoubleWritable message : messages) {
4          minDist = Math.min(minDist, message.get());
5      }
6      if (minDist < getValue().get()) {
7          setValue(new DoubleWritable(minDist));
8          for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
9              double distance = minDist + edge.getValue().get();
10             sendMessage(edge.getTargetVertexId(),
11                 new DoubleWritable(distance));
12         }
13     }
14     voteToHalt();
15 }
```

Source: [4]

Simple Graph Traversal



vertices with values

edges with values

messages

superstep barriers

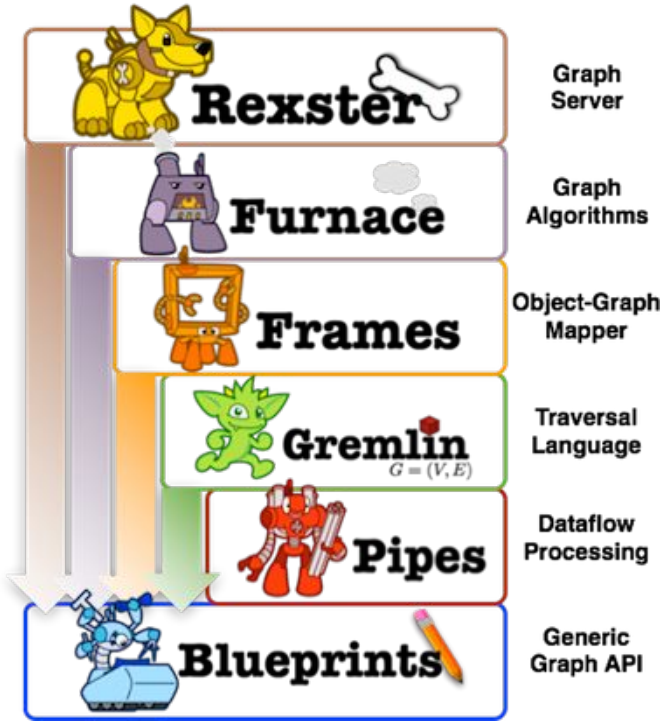
Source: [4]

Giraph Setup

- Docker setup (full stack):
  - https://hub.docker.com/r/uwsampa/giraph-docker/
  - Contains:
    - Apache Hadoop
    - Apache Giraph
    - Java SDK for compiling and deploying Giraph jobs
- Native installation (non-docker):
  - http://giraph.apache.org/quick_start.html
  - Install Java, Hadoop, Giraph

Apache TinkerPop

- Provides a common API for all supported Graph Databases and processors [5]
- Core component is the Gremlin traversal language
- Giraph supports Gremlin via Hadoop-Gremlin
  - Allows querying data sent to Giraph in an interactive Gremlin shell
  - Still vertex-centered approach, but some ideas are easier to express in Gremlin than in Java
- Example: **v.outE('knows').inV.filter{it.age > 30}.name**



Summary

- Giraph is a graph processing engine
- Backed by Hadoop
- Written in Java, has API support for other languages
  - TinkerPop (Gremlin) for non-Java queries
- Primary use case:
  - Fast big data processing when storage is backed by another DB

Sources

[1] http://tinkerpop.apache.org/providers.html

[2] http://synsem.com/SeaNode-2014-06-25/images/BSPvsForkJoin.svg

[3] http://giraph.apache.org/io.html

[4] https://cwiki.apache.org/confluence/display/GIRAPH/Shortest+Paths+Example

[5] http://tinkerpop.apache.org/docs/3.0.1-incubating/

[6] https://research.googleblog.com/2009/06/large-scale-graph-computing-at-google.html