# Evaluating partitioning of big graphs

Fredrik Hallberg, Joakim Candefors, Micke Soderqvist
fhallb@kth.se, candef@kth.se, mickeso@kth.se

Royal Institute of Technology, Stockholm, Sweden

**Abstract.** Distributed processing is a good way of handling big data. However distributing the data introduces other complications such as higher latency when communication is needed between servers. This can be solved by clever partitioning. With the data represented as graphs, partitioning can be done by maximizing locality while minimizing edgecuts when placing vertices into components of roughly equal size. This paper partitions a graph containing 1.5 million edges and 64 thousand vertices into three partitions using a cluster running Apache Hadoop and Apache Giraph. The resulting partitioning reduces the amount of edgecuts from 66,66% to 15,2%.

**Keywords:** graph partitioning, Apache Giraph, Okapi, Spinner

## 1 Introduction

### 1.1 Graphs

A graph is a good way to represent data objects with relations. Graphs consists of vertices and edges which can hold values. These values may e.g. represent lengths of roads (edges) between cities (vertices). Graphs are also a good way of representing social networks with friends and their relationships. There is more to be gained from representing these kind of data as graphs than just the visual aid. Many good heuristics have been developed to solve problems on graphs that would otherwise be difficult or ineffective to solve with regular relational databases. This is especially true when considering the increasing sizes of data.

### 1.2 Graph partitioning

With increasing size comes the issue of managing these big amounts of data when it can no longer be stored or processed on a single server. To solve both this problem and to boost the efficiency in computations, distributed systems are utilized.

Distributing the data introduces other complications such as latency when communication is needed between nodes within a cluster or even between different clusters. Clever partitioning of the graph can minimize these problems. Instead of randomly distributing vertices of the graph across servers an algorithm could make sure that we minimize the internode communication and thus the latency.

A typical query in a social network like Facebook is to fetch information from all your friends (neighboring vertices). The latency of this query may be significantly lowered by localizing these vertices to the same server. Facebook utilizes this by first calculating a good partitioning using Giraph and then distributing the information in their relational databases according to the suggested partitioning. [1]

Another paper[2] describes how this can be done on an incremental basis using the underlying graph database system neo4j.

Graph partition problems are NP-hard and some instances of graph partitioning such as uniform or balanced graphs can be shown to be NP-complete. With partitioning the result should be a graph with components of roughly the same size and the amount of edge cuts should be minimized. More formal we need to make sure that each component does not contain more than $v * n/k$ of the graph vertices, with $n$ the total amount of vertices, a parameter $v \geq 1$ and $k$ components [3].

### 1.3 Contribution

Much data can be represented by graphs. In large scale systems with big data, the efficiency of querying the information may be dependent on a good structure of the data. This project aims to evaluate how partitioning of big data graphs may be done with the aid of distributed parallel computations using Apache Giraph.

## 2 Method

*The following section describes the components used in this paper to evaluate partitioning of big graphs.*

### 2.1 Apache Hadoop and Giraph

Apache Hadoop is a framework for distributed processing and storage with high fault tolerance and scalability. Using HDFS (Hadoop Distrubuted File System), for storage where files can be replicated throughout the cluster of machines, and MapReduce for distributed parallel processing. Later versions of Hadoop also includes a resource negotiator, YARN, which manages the distribution of the workload in the Hadoop cluster. [4]

Apache Giraph is a large scale graph processing framework which can be used in conjunction with Hadoop to process big graphs over a cluster of computers. Like Google Pregel, which Giraph has evolved from, it is based on the BSP (Bulk Synchronous Parallel) model of processing and is used instead of the more commonly used MapReduce [5]. When Giraph receives an application along with an input graph to be processed, an application master is appointed among the cluster slaves. This application master, which is an extension of BSP by Giraph, has a more global view of the graph than the rest of the processing units in the cluster, the workers. Each worker has in turn YARN containers, the innermost processing units with dedicated memory and CPU. The application master starts by splitting the input graph file among the workers using a partitioner. By default Giraph use a HashPartitioner, distributing the vertices among the workers at random.

The processing on all the workers is vertex centralized and done iteratively in supersteps. In each superstep, all the vertices run their compute()-method, utilizing their vertex values, outgoing edge values and messages sent to the vertex to make the computation. The compute-method results in either messages to other vertices over edges, sharing information to globally accessible pools, aggregators, or a voting to halt the application. At the start of each superstep the application master runs a masterCompute()-method to oversee and manage the distributed computation. [6]

### 2.2 Okapi Spinner

Okapi is a library for the above mentioned framework Giraph. It includes several graph algorithms and input formats to be used when processing graphs with Giraph. The library includes the application Spinner, which is an algorithm to be used for partitioning a graph to the desired number of components. Spinner is designed to scale to a ”billion vertices and beyond”[7]. Apart from creating an initial partitioning, Spinner can receive already partitioned graphs for further repartitioning. Spinner computes an edge-based balanced k-way partitioning [1] of the graph while trying to maximize locality and balancing. However, in relation to the algorithm used by Hermes[2] Spinner is less aggressive to edge cuts. The algorithm starts from a random partitioning of the graphs and migrates vertices iteratively between components. Migrations are decided by local information at the vertices, received from each vertex communicating their component to neighbors.

## 3 Evaluation

### 3.1 Setup

In order to try to simulate big data graph partitioning, a cluster of Hadoop 2.5.1 with YARN running Giraph 1.1.0 was setup. Hosted on three different virtual machines, with one master node and three slave nodes (the master also acting as a slave). As Giraphs main description is ”Large-scale graph processing on Hadoop”, this cluster is rather small compared to the intention of the framework and industry clusters, but was deemed to be sufficient for the evaluation. Each node was set up with 4 GB of RAM and 8 vCPU cores. The configuration process of the cluster was more time consuming than what was initial thought and well over 50 hours were devoted to get the cluster operational.

The dataset [8] used for this project was an anonymized sample from users in New Orleans and their relationships on Facebook. This graph was directed and consisted of 63731 vertices and 1545686 edges. The format of the file was:

<vertex> <target vertex>

Even though Hadoop supports reading data from distributed database systems we chose to import our dataset in the form of a text file on the HDFS.

### 3.2 Results

Running the Spinner Giraph application with this graph yielded a partitioning output of the format:

<vertex> <component>

The output file contained all the 63731 vertices mapping their computed component id. This map could function as a guideline of how the vertex data should be migrated. It is not only other database systems that would gain from this, Spinner includes a partitioning module to be used with other Giraph applications. This module has the purpose of partitioning the input graph onto the workers according to a pre-computed partitioning scheme.

[1] `http://en.wikipedia.org/wiki/Partition_problem`

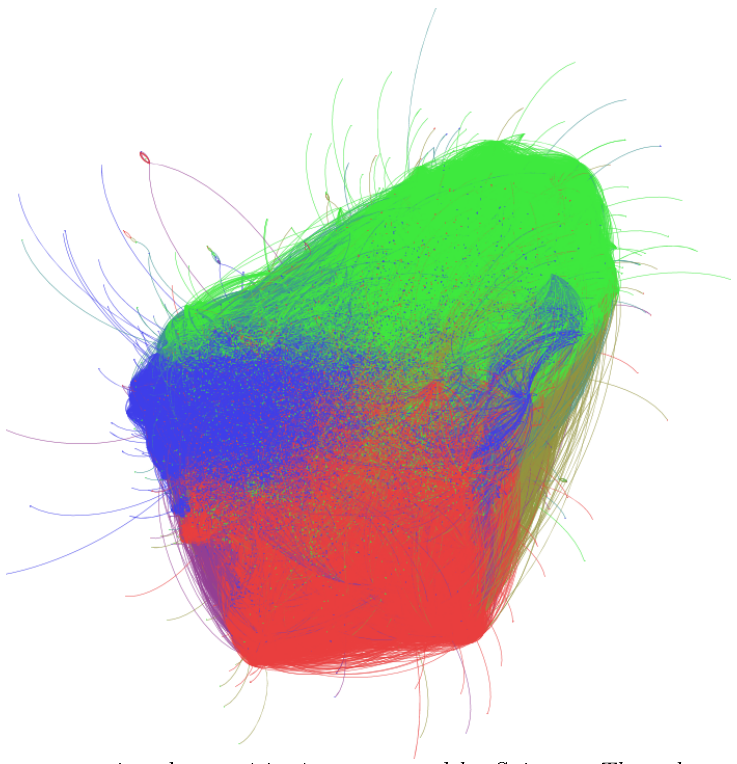

Fig. 1: *Graph representing the partitioning generated by Spinner. The colors of the vertices represent their assigned components computed by Spinner, while the layout was calculated using "Force Atlas 2" in the graph visualizing tool Gephi. This visualization took 45 minutes to produce using a system with 32GB RAM and 12 CPUs compared to the 40 seconds the partitioning took using Spinner on our cluster.*

As we intended to use this partitioning to evaluate with Giraph applications we let Spinner partition the graph into 3 components, one for each worker. The components held 16008, 31828, 15895 vertices respectively, with 234701, 15,2% edgecuts between components. This may be compared to performance of Hermes[2] partitioning on a similar dataset, with about 11 % edgecuts. It should be noted that the edge cut rates are affected by the different number of components, as 16 were used by the Hermes-paper. These rates can be compared to the theoretical rates aqcuired by using the default random partitioning generating edgecut rates of 66,66% and 93,75% respectively.

### 3.3 Conclusions

Even though evaluated in a rather small scale, partitioning big data graphs can be done efficiently with Apache Giraph, in a similar manner as described by Nicoara et al [2]. Other conclusions from this project is that working with Giraph and Hadoop is not as straight forward as the first impression would suggest. With many different modules follows many sources for errors, lots of configuration and research required. The project changed direction from implementing the Hermes partitioning algorithm in Giraph to evaluating the concept of partitioning big data graphs efficiently. This was partially decided based on the time limit and scope of this project, but also on the high complexity in writing and running Giraph applications.

## References

1. Alessandro Presta, Alon Shalita, Brian Karrer, Arun Sharma, Igor Kabiljo, Aaron Adcock and Herald Kllapi: Large-scale graph partitioning with Apache Giraph [ONLINE], https://code.facebook.com/posts/274771932683700/large-scale-graph-partitioning-with-apache-giraph/
2. Nicoara, Daniel; Kamali, Shahin; Daudjee, Khuzaima; Chen, Lei : Hermes: Dynamic Partitioning for Distributed Social Network Graph Databases Published in Proc. 18th International Conference on Extending Database Technology (EDBT), March 23-27, 2015
3. Konstantin Andreev, Harald Racke : Balanced Graph Partitioning, Springer-Verlag, 1433-0490 (2006)
4. White, Tom.: Hadoop: The Definitive Guide O'Reilly Media, Inc., 9780596521974 (2009)
5. Malewicz, Grzegorz and Austern, Matthew H. and Bik, Aart J.C and Dehnert, James C. and Horn, Ilan and Leiser, Naty and Czajkowski, Grzegorz.: Pregel: A System for Large-scale Graph Processing, Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 135–146 (2010)
6. Giraph, The Apache Software Foundation [ONLINE], http://giraph.apache.org/
7. grafos.ml. Okapi: Most Advanced Open-Source ML Library: for Apache Giraph. [ONLINE] Available at: http://grafos.ml/index.html#Okapi. [Accessed 25 May 15]. (2014)
8. Bimal Viswanath and Alan Mislove and Meeyoung Cha and Krishna P. Gummadi : WOSN 2009 Data Sets, Facebook links, On the Evolution of User Interaction in Facebook, Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (2009)