

Graph Processing

Thejas Raju Ravi Govind Raju

1 Introduction

Data that can be characterised by its volume, variety, velocity and flow across systems is called Big Data. The volume at which data is generated and the types of data present is more compared to traditional datasets. The velocity at which the data is produced and its variability across systems is also more than other datasets.

Hadoop uses multiple nodes across networks to perform computations across a Big Data set. Hadoop framework implements the MapReduce programming model. Hadoop uses a Distributed File System which facilitates the access storage and access of data across networks in a convenient format. The system resources are managed by the YARN module. Hadoop-Commons module provides the tools and utilities required to process Big Data.

MapReduce programming model is adopted for processing BigData. The model performs two functions mainly, mapping and reducing. In the mapper function, the model takes the input from the specified file in the Hadoop Distributed File System and outputs key-value pairs. These key-value pairs are summarised in the reduce function. The reducer performs aggregation and statistical operations on the key-value pairs that are generated from the mapper. The reducer also outputs a pair of keys and values. These are passed to the file which is specified for output data.

Apache Giraph is mainly used for graph-processing jobs on big datasets. It was developed based on Google's Pregel software. Giraph uses Hadoop's MapReduce programming model to perform graph-processing. An application of Giraph's working is the 'People you may know' feature in Facebook which is delivered using graph-processing.

2 Requirements

The data to be analysed is a citation graph from the e-print arXiv (high energy physics phenomenology papers). The data set is ideal and qualifies as a BigData set as it possess a huge volume of data which consist of a citation graph. The dataset being used covers all the citations within a dataset of 34,546 nodes (papers) with 421,578 edges (citations) between them.

The system is required to perform Big Data jobs using the Hadoop framework which implements the MapReduce programming model. The system is also required to process the dataset using Apache Giraph which utilizes Hadoop's MapReduce programming model. The system is expected to be capable of processing the dataset to analyse and provide insights into the number of times a paper has been cited. The system should be able to find all the different papers that are present in the dataset and provide the total count of the edges pointing towards these nodes. Also, indirect the system should be capable of handling indirect citations as well. Based on the result in the output file, the user can draw conclusions such as the top ten papers with largest citation count. This can be obtained by using Bash functionality.

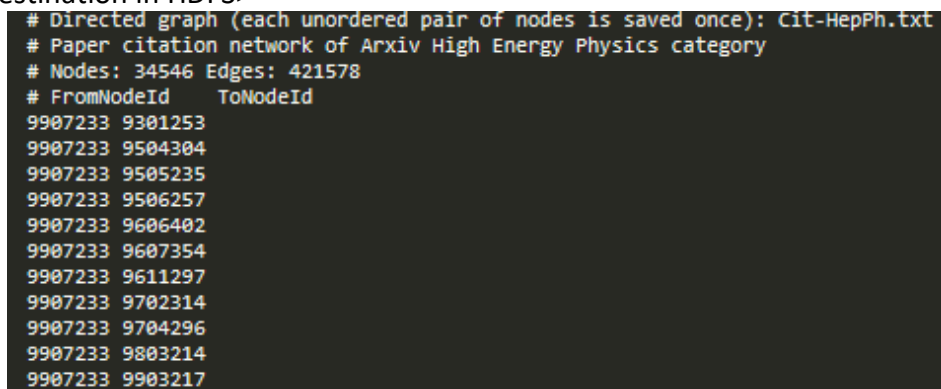
3 Design

3.1 Task-1: Hadoop MapReduce

- a) Download the dataset in HDFS

The dataset is downloaded and extracted to the local system. This data is uploaded to the Hadoop File System using the command:

```
$HADOOP_HOME/bin/hadoop dfs -put <path of dataset in local system>  
<destination in HDFS>
```



```
# Directed graph (each unordered pair of nodes is saved once): Cit-HepPh.txt  
# Paper citation network of Arxiv High Energy Physics category  
# Nodes: 34546 Edges: 421578  
# FromNodeId ToNodeId  
9907233 9301253  
9907233 9504304  
9907233 9505235  
9907233 9506257  
9907233 9606402  
9907233 9607354  
9907233 9611297  
9907233 9702314  
9907233 9704296  
9907233 9803214  
9907233 9903217
```

Figure 1: Excerpt of Dataset

The above image is an excerpt from the dataset being used for graph-processing. The column contain nodeIDs where the first column specifies fromNode and the second column species toNode. A pair of fromNodeID and toNodeID shows the presence of an edge between the nodes. For example, 9907233 and 9301253 are fromNodeID and toNodeID respectively. These two nodes show individual papers where paper 9907233 cites the paper 9301253.

b) Write a MapReduce job in Hadoop that computes the citation number of papers.

The MapReduce job to compute the citation number of papers implements two instances of mappers and reducers each. This is done as indirect citations must also be taken into account.

PaperCitationMapper1: The file containing the graph nodes is taken as input and split into tokens using the separator (space) present between them. The map produces two key-value pairs where in the first pair, key is the toNodeID and value is the fromNodeID. In the second pair, the key is the fromNodeID and the value is the toNodeID with a '-' symbol appended to it initially.

```
public void map ( Object key , Text value , Context context
) throws IOException , InterruptedException {
String[] tokens = value.toString().trim().split("\\s");

fromNode=tokens[0];
toNode=tokens[1];
context . write (new Text(toNode), new Text(fromNode));
context.write(new Text(fromNode), new Text("-"+toNode));
}
```

Figure 2: PaperCitationMapper1

PaperCitationReducer1: The key-value pairs generated from Mapper1 are taken as input to this reducer where a comma is appended between each cited node. The reducer returns a key-value pair containing the citing node as key and all the nodes it cites as the values which are separated by commas.

```

public void reduce ( Text key , Iterable < Text > values ,Context context
) throws IOException , InterruptedException {

String all = new String();
for ( Text val : values ) {
all += ","+val.toString();
}

all=all.substring(1,all.length());
context . write ( new Text(key) , new Text(all) );
}

```

Figure 3: PaperCitationReducer1

PaperCitationMapper2: The output of the Reducer1 is stored in an intermediate file and is passed as the input to Mapper2. The values are split based on the commas present between them and for each instance of a paper citing that does not begin with '-', the citation count is increased. The, for all tokens which start with '-' symbol, the key-value pair where key is the token without '-' and value is count is generated as output.

```

public void map ( Text key , Text value , Context context
) throws IOException , InterruptedException {
String[] tokens = value.toString().trim().split(",");
int count=1;
for(String key1:tokens)
{
if(!key1.startsWith("-"))
{
count+=1;
}
}

for(String key1:tokens)
{
if(key1.startsWith("-"))
{
context.write(new Text(key1.substring(1,key1.length())), new IntWritable(count));
}
}
}

```

Figure 4: PaperCitationMapper2

PaperCitationReducer2: The output of Mapper2 is passed as input to this reducer. This function produces the summary where for each key, the value of all counts associated with the key is summed. This summation is output as the value of the key from Reducer2 to the output file.

```

public void reduce ( Text key , Iterable <IntWritable> values ,Context context
) throws IOException , InterruptedException {

    int sum=0;

    for(IntWritable v:values)
    {
        sum+=v.get();
    }

    context . write ( key ,new IntWritable(sum) );
}

```

Figure 5: PaperCitationReducer2

DriverClass: As there are two instances of Mappers and Reducers, the main driver class takes an input of three arguments where the first argument specifies the file containing input data nodes, the second is the intermediate output file and the third specifies the final output file. The first job performs MapReduce using Mapper1 and Reducer1. The output of this is passed to another MapReduce job which implements Mapper2 and Reducer2.

```

public static void main ( String [] args ) throws Exception {

    Configuration conf = new Configuration();
    Job job= Job.getInstance(conf, "PaperCitation at distance 2");
    job.setJarByClass(PaperCitation.class);
    job.setMapperClass(PaperCitationMapper1.class);
    job.setReducerClass(PaperCitationReducer1.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    job . waitForCompletion ( true );

    Job job2 = Job.getInstance(conf, "PaperCitation Continued");
    job2.setJarByClass(PaperCitation.class);
    job2.setMapperClass(PaperCitationMapper2.class);
    job2.setReducerClass(PaperCitationReducer2.class); job2.setMapOutputKeyClass(Text.class);
    job2.setMapOutputValueClass(IntWritable.class);
    // to ensure job2 mapper takes the input as key,value as produced by Reducer
    job2.setInputFormatClass(KeyValueTextInputFormat.class);
    FileInputFormat.addInputPath(job2, new Path(args[1]));
    FileOutputFormat.setOutputPath(job2, new Path(args[2]));
    System . exit ( job2 . waitForCompletion ( true ) ? 0 : 1);

}

```

Figure 6: DriverClass for PaperCitation job

c) Output the top ten papers (with largest citation number):

The following snapshot shows the Bash command used to obtain the final output containing the top ten papers with largest citation count:

```

user1@user1-VirtualBox:~/Desktop/NewWC$ $HADOOP_HOME/bin/hadoop dfs -cat /output10/p* | sort -k2 -r -n | head -n10
Warning: $HADOOP_HOME is deprecated.

9803315 16371
9804398 12781
9408384 11359
9512380 10775
9807344 10510
9507378 10362
9606399 10269
9207214 9750
9803466 9679
9807216 9505

```

Figure 7: Output of TASK-1 Top ten papers using Hadoop MapReduce

The output file is pipelined to the sort command which selects the second column containing citation count using '-k2', '-r' directs the results to be in descending order, and the '-n' performs numerical sorting. This output is piped to the head command where '-n10' limits the output to top ten papers with largest citation count.

3.2 Task-2: Giraph

a) Installing Giraph

Apache Giraph set-up can be verified by running a sample shortest-path program on a tiny_graph.txt file that contains graph nodes as input. The shortest-path from source node is calculated. In the following snapshot, it can be seen that the distance of nodes 0, 2, 3, and 4 from 1 is 1, 2, 1, and 5 respectively.

```

user1@user1-VirtualBox:~$ $HADOOP_HOME/bin/hadoop dfs -cat /user/hduser/output/shortestpaths2/p*
Warning: $HADOOP_HOME is deprecated.

0      1.0
1      0.0
2      2.0
3      1.0
4      5.0

```

Figure 8: Output of ShortestPath example program

b) Graph processing job to compute citation number of papers

The graph-processing job takes input of file containing fromNode and toNode. This data is used to create a graph and compute the citation numbers. The citation number can be calculated using the in-degree count of a particular node. For super-step 1, the in-degree count of edges pointing to a particular node is calculated and stored in the vertex. The in-degree count is also stored in the message to be used in the next super-step. In super-step 2, the values stored in the messages are summed up which provides the count of indirect citations. This count is added to the vertex value initialised in super-step 1. This provides the total count of indirect citations present at each vertex.

```
public class PapersSignificance extends BasicComputation<
    LongWritable, LongWritable, NullWritable, DoubleWritable> {

    @Override
    public void compute(
        Vertex<LongWritable, LongWritable, NullWritable> vertex,
        Iterable<DoubleWritable> messages) throws IOException {
        if (getSuperstep() == 0) {
            Iterable<Edge<LongWritable, NullWritable>> edges = vertex.getEdges();
            for (Edge<LongWritable, NullWritable> edge : edges) {
                sendMessage(edge.getTargetVertexId(), new DoubleWritable(1.0));
            }
        } else if (getSuperstep() == 1){

            Iterable<Edge<LongWritable, NullWritable>> edges = vertex.getEdges();
            Long sum = 0;
            for (DoubleWritable message : messages) {
                sum++;
            }

            for (Edge<LongWritable, NullWritable> edge : edges) {
                sendMessage(edge.getTargetVertexId(), new DoubleWritable(sum));
            }

            LongWritable vertexValue = vertex.getValue();
            vertexValue.set(sum);
            vertex.setValue(vertexValue);

        } else if (getSuperstep() == 2){

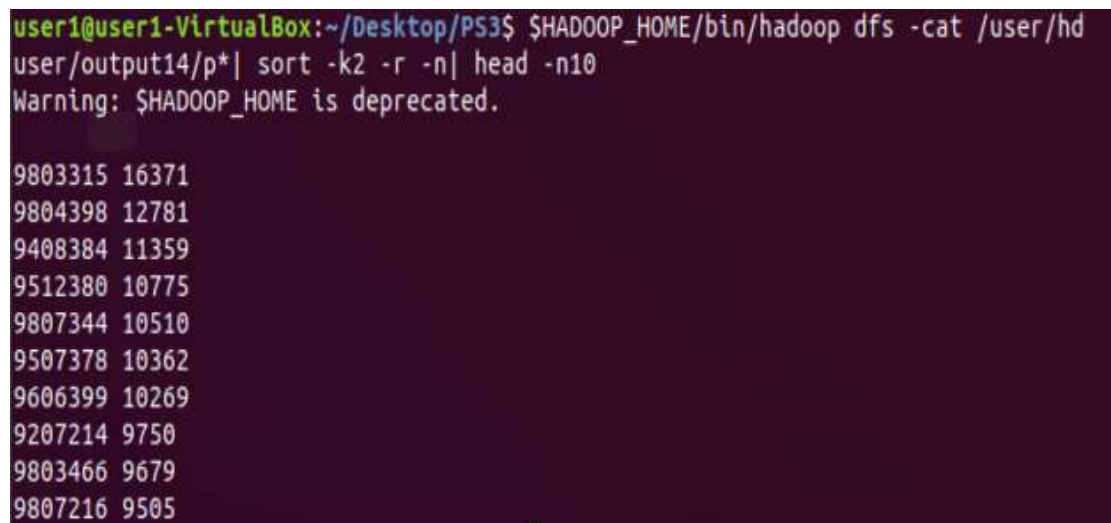
            Long sum = 0;
            for (DoubleWritable message : messages) {
                sum+=message.get();
            }
            LongWritable vertexValue1 = vertex.getValue();
            Long vertexValue2 = vertexValue1.get();
            vertexValue2+=sum;
            vertexValue1.set(vertexValue2);
            vertex.setValue(vertexValue1);

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

Figure 9: PapersSignificance.java

c) Output top ten cited papers:

The following snapshot is the output obtained when the graph processing job is executed. It shows the top ten papers which have been cited.



```
user1@user1-VirtualBox:~/Desktop/PS3$ $HADOOP_HOME/bin/hadoop dfs -cat /user/hd
user/output14/p*| sort -k2 -r -n| head -n10
Warning: $HADOOP_HOME is deprecated.

9803315 16371
9804398 12781
9408384 11359
9512380 10775
9807344 10510
9507378 10362
9606399 10269
9207214 9750
9803466 9679
9807216 9505
```

Figure 10: Output of Task-2 Top ten papers using Giraph

4 Conclusion

The dataset containing graph data was analysed to find the top papers with highest citation counts. The Hadoop framework implements the Map-reduce programming model where there is no information passed between steps but only key-value pairs are sent from Mappers to Reducers and then to output file. The input data required to be split appropriately before analysis. This model required the implementation of two instances of Mappers and Reducers each. This means that more disk space and network transactions take place during execution.

The Giraph-model also implements the map-only functionality. It is more practical for iterative processing when compared to Hadoop as there is message-passing involved and does not take up a lot of disk-space. The computation at each step can be controlled using the super-step feature. The message-passing feature allows storage of data which can be used at later stages of computation. The implementation has mechanisms to store vertex values, edge values and other features. This makes Giraph more desirable for graph-processing jobs. Hadoop can be used for small-scale graph processing jobs whereas Giraph usually performs better and has more robust features which can be utilised for graph-processing.

References

<https://github.com/manuelcoppotelli/giraph-demo/blob/master/InDegree/src/InDegree.java>

<http://hadooptutorial.info/mapreduce-programming-model/>

<http://snap.stanford.edu/data/cit-HepPh.html>

<http://bigdataprogrammers.com/get-distinct-words-file-using-map-reduce/>