ISSN: 0975-766X
CODEN: IJPTFI
Available Online through *Research Article*
www.ijptonline.com

AN ENHANCED FORD-FULKERSON ALGORITHM TO DETERMINE MAX-FLOW IN LARGE MESH NETWORK USING HADOOP

[1]Raghu B Hemanth, [2]R.Vijayan
M.Tech IT (Networking), VIT University, Vellore, India.
Asst. Professor [SG], SITE, VIT University, Vellore, India.
Email: raghubasoor@gmail.com

Received on 20-08-2016 *Accepted on 25-09-2016*

Abstract

Maximum-flow algorithm is used to determine maximum possible bandwidth between source and destination in a large mesh network where packets can travel any root independent of each other. The algorithm also helps to find spam sites which are in contact with the trusted site. The maximum flow problem arises from the large scale network graphs obtained from the internet such as Facebook, yahoo, twitter, Google, LinkedIn, spam sites, etc,. Nowadays, the massive amount of data obtained from the World Wide Web (WWW) are growing rapidly which is too expensive to store and process using the conventional storage algorithm. The solution to this problem is to effectively parallelize the max-flow algorithm using Hadoop framework with a cluster of commodity hardware. The algorithm uses HDFS-Hadoop distributed file system to store large scale network graphs and MapReduce to process huge datasets parallely with a cluster of commodity hardware. The algorithm calculates the max-flow on a large network graphs with thousands of vertices and edges using MapReduce framework with a cluster of number of machines in reasonable time. It also compares and analyses the different maximum-flow algorithms such as traditional Ford–Fulkerson algorithm and enhanced Ford-Fulkerson algorithm with a cluster of commodity hardware in a reasonable time.

Keywords: MapReduce, HDFS, cluster, datasets.

1. Introduction

The max-flow problem comes from the large network graph obtained from the World Wide Web, Social Networks such as Facebook, Gmail, Twitter, etc. and also from the human body. The max-flow algorithm give solutions to the problem faced from the graph obtained from the internet such as; to determine the maximum available bandwidth two computers irrespective of the traffic within it, to find unwanted sites which in contact with other trusted sites and also to find the fault parts in the human body [1]. Nowadays, the real world graph is growing rapidly with the increase in

number of users using the internet. These graphs are too large to the conventional storage algorithms- to store and process. In this internet world, more than billion users having social networking account and each user can have more than one account and there is no limit to it. It is very difficult to store and process the huge datasets obtained from the social networking sites. The max-flow algorithm takes nearly quadratic runtime complexity with traditional sequential Ford-Fulkerson method. Since, the graphs obtained from the internet outgrown the traditional storage algorithms. While Terabytes sized graphs exceeds the storage capacity available in the single conventional machine [1].

An expensive super computer overcomes these storage problems. But, it's economical to use multi node or single node cluster using commodity hardware's or to use cloud computing technology. This distributing approach need to look after many issues like task distribution, load equalizing, job scheduling, error resistance and inter-process communication. These issues can be overcome by Hadoop framework which uses HDFS for storing huge datasets and MapReduce for processing it.

2. Related works

In this section we briefly review some of the popular works proposed in the literature to signify the importance of handling huge datasets in MapReduce framework. The work in [1] proposed the Ford-Fulkerson algorithm which overcomes the max-flow problem to store and process huge datasets using MapReduce framework. The work in [2] [5] the MapReduce framework is used as de-facto framework for huge datasets analysis, big data and data mining. The network graph analysis is one of the prominent areas. The network graphs obtained from World Wide Web and social networking is too large in size with billion of vertices and edges. To store and process those huge datasets, programmers have been using MapReduce framework to analyse those huge datasets effectively. The work in [3] [4] proposed MapReduce divides that huge datasets and process it parallely to reduce the time complexity to reasonable time.

The work in [6] proposed the approach helps to control network congestion issues and also uses the bandwidth effectively. The work [7] has proposed the linear programming techniques to solve maximum-flow problem consumes quadratic time complexity. When these techniques are used with huge datasets it consumes much more time as expected. The work in [8] proposed the network graphs obtained from the internet are too large to analyze as it contains million of vertices and edges. MapReduce provides effective programming model to store and process large scale network graphs obtained from internet.

3. System Model: We use the same model as proposed by authors to represent the Hadoop environment that consists of numbers of clusters that includes number of Data nodes within it. The schematic of Hadoop Distributed File system architecture is shown in the figure 1.

Hadoop distributed file system is a specially designed distributed file system works especially on massive amount of datasets on a cluster of commodity hardware with streaming access pattern. Hadoop distributed file system is highly fault tolerant and capable of working with low- cost hardware's.

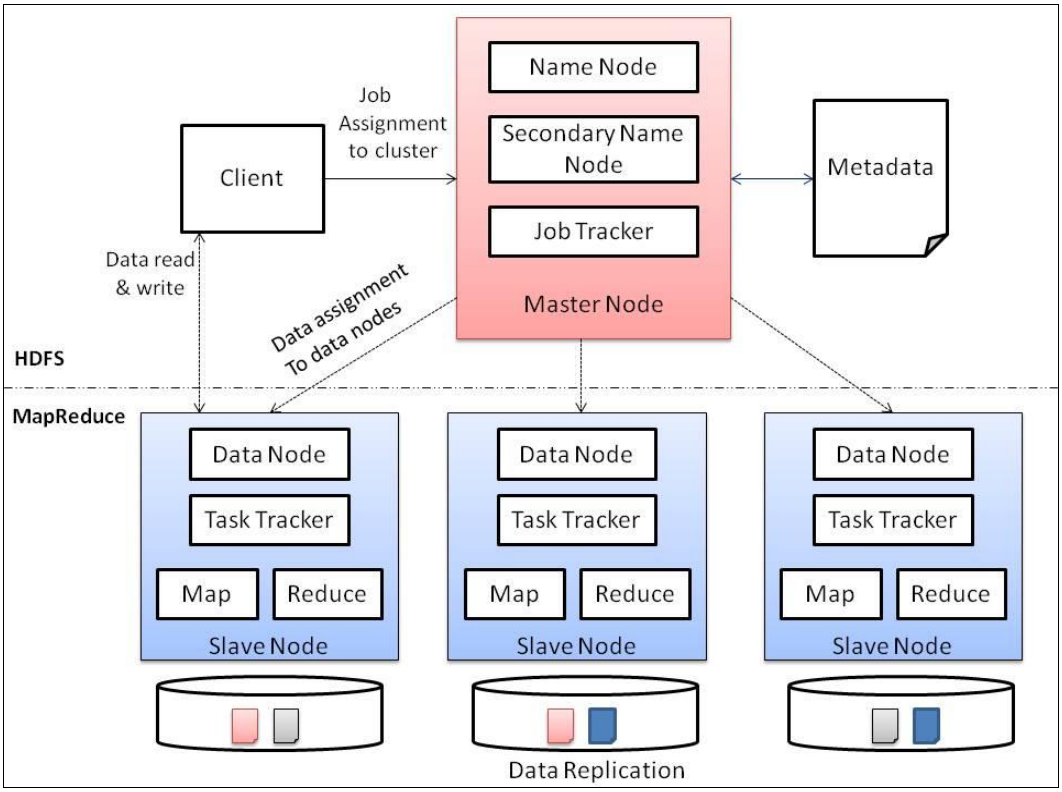

Figure 1: Hadoop Distributed file system architecture.

Hadoop distributed file system consist of major five services

- Name node.
- Secondary Name node.
- Job tracker.
- Data node.
- Task tracker.

Hadoop distributed file system works on master-slave approach. Name Node is the master node, which keeps track all the activities of the system such as storing huge data, splitting of huge datasets, assigning jobs to the data nodes, responsible for all client requests, keep track of all data assigned to data nodes in a metadata and also keep track of replication of data.

Secondary Name Node is the master node and it backup of the Name node. In case name node is lost, the secondary name node can be used as name node which provides recovery. The job tracker is also master node which keeps track of the job assigned to the task tracker. These master nodes can communicate with each other but they are unable to communicate with all slave nodes. The Name Node can only communicate with data node and same way job tracker can only communicate with task tracker.

The Data Node is the Slave node which store and manipulate the data given by name node which is in turn assigned by client. Each Data node stores and process 64MB data and it also replicate the data three times for backup in case the data is lost due to overhead.

The Large scale graph with thousands of vertices and edges is given to master node for storing and processing. The processing is done in MapReduce which consist of Mapper and reducer function where the algorithm is implemented. The master node divides the graph into sub graphs and each sub graph is chosen a local source and destination and the job is assigned to Mapper.

4. Implementation

An enhanced Ford-Fulkerson algorithm consists of four modules:

- Graph Creation
- Main function
- Mapper Function
- Reducer Function

The Large scale network graph is created using the text input format with 5051 nodes with weighted edges. This large scale graph is input file which is being processed in rest of the modules. Main function takes the input file and stores in the Hadoop distributed file system and process that input file and stores in output file. The Mapper function divides the input file and process it individually and the reducer function combines the resultant from the Mapper and gives the resultant into output file.

4. 1 Graph creation

The input format is

ID WEIGHT | EDGES | DISTANCE | COLOR | Path_Taken_EDGES |

Where

ID = node identifier: integer value.

WEIGHT = capacity between any two edges: integer value.

EDGES = neighboring nodes: 1's neighboring nodes 2 & 3.

DISTANCE = the distance from one node to another after visiting.

COLOR = it's an indicator to check whether the node is visited or not.

- **Psuedocode of Weighted Node**

a. Public class WeightedNode {

b. Initialize WHITE, GRAY, BLACK;

c. Initialize id, weight, distance, edges, color, Path_taken_edges;

d. Public WeightedNode (String str) {

i. Split the input string into Key and Value;

ii. Split value into tokens;

iii. WEIGHT|EDGES|DISTANCE|COLOR|Path_Taken_Edges

iv. String [] tokens = value.split ("|");

v. Save node's key;

vi. Weight = tokens [0];

vii. Edges = tokens [1];

viii. Distance from origin = tokens [2];

ix. Color tokens = [3];

x. Path taken edges tokens = [4];

}

b. Returns a array of string ;

}

4. 2. Main function

The main function of the enhanced Ford-Fulkerson algorithm is similar to traditional Ford-Fulkerson algorithm. The main function defines the input file path and also the location of the output file of the each iteration as well as final result.

a. Public class EnhFordFulkerson {

b. Initialize

numberOfRowsToProcess=0;

SetOneReducer=false;

c. Set the input/output Path

{

d. inputPathIteration=”EnhFordFulkerson/input/input-graph”

e. inputPathOtherIteration=”EnhFordFulkerson/output/output-graph”

f. OutputPathIteration=”EnhFordFulkerson/output/output-graph”

}

g. Set the Job configuration

{

h. JobConf conf= JobConf(getConf(), EnhFordFulkerson.class);

i. Conf.setJobName(“EnhFordFulkerson”);

j. Set the Mapper and Reducer Classes Conf.setMapperClass(EnhFordFulkersonMapper.class)

Conf.setReducerClass(EnhFordFulkersonReducer.class)

Conf.setCombinerClass(EnhancedFordFulkersonCombiner.class)

}

}

4.3 Mapper Function

The Enhanced Ford Fulkerson Mapper divides the input file and processes it parallely. Initialize it creates configuration log file of the Mapper Function and it start processing from the source with colour GRAY which an indicator. The neighbouring node of the source is visited and marked as GRAY. Initialize the weight of the node to zero until the weight is known. The weight is set during the reducer phase. Once the node visited the previous node is marked to black.

a. Public class EnhFordFulkersonMapper{

b. Logger LOG = Logger.getLogger(FordFulkersonMapper.class);

// Initialization log file

c. Public void Mapper()

{

d. WeightedNode node= WeightedNode(value.toString());

   // The node is the gray node, visit that node and mark it

e. If(node.color=gray){

f. For(v: getEdges){

   // initialize the weight of the node to zero until weight is known.

   // The weight will be set during reducer phase.

g. weightedNode vnode= weightedNode(v)

h. vnode.setWeight(0);

i. vnode.setMaxFlow(node.getDistance()+node.getWeight());

j. for(pathTakenEdge=getPathTakenEdges){

k. vnode.addPathTakenEdge(PathTakenEdges)

   }

l. node.setcolor=black

   }}}

4.4 Reducer Function

The input file from the specified path is store in Hadoop distributed file system. The input file is split into number of sub files and stored in the file system. The sub- graphs are processed parallely in the Mapper phase of the MapReduce framework. Initially the weight and distance is declared as zero. The graph starts with its neighbour node of the source. If the neighbour node is not visited, check for the weight of the neighbour node. If the weight of the neighbour node is more than the present node weight, add weight to the maximum flow.

a. Public class EnhFordFulkersonReducer{

b. Initialize

   i. Weight=0, distance=0, color= MaxFlow.color

c. Loop through all the values for this key

d. While(values.hasNext()){

   i. Text value= value.next();

   ii. Save the Maximum Weight

e. If(getWeight()>weight){

i. weight=getWeight;

}

f. Save the Max Flow Distance

g. If(distance!=0){

h. If(getdistance()<distance && getdistance()!=0){

i. distance= getdistance();

ii. PathTakenEdge.clear();

iii. For(pathTakenEdge=getPathTaknEdge()){

iv. pathTakenEdges.add(pathTakenEdges);

}

else{

i. If(getdistance()>distance){

i. Distance=getdistance();

ii. Clear the Path Taken Edges

iii. PathTakenEdge.clear();

j. For(pathTakenEdge=getPathTakenEdge())

{

i. pathTakenEdges.add(pathTakenEdges);

}

}

5. Results and Discussion

The large scale network graph obtained over the internet can be analysed to find the maximum possible bandwidth required to reach destination from the source and also finds the spam site which is in contact with the trusted site. The Ford-Fulkerson algorithm is used for that purpose. The paper compares traditional Ford-Fulkerson algorithm and an enhanced Ford-Fulkerson algorithm using Hadoop framework and also compares and analyses these algorithms with respect to various metrics. The comparison of Ford – Fulkerson and an enhanced Ford-Fulkerson algorithm is based on various metrics such as – feasible weight from source to destination, number of iterations, run time complexity, map task, shuffle task and reduce task performances. The figure 2 shows the analysis of both the algorithm with

respect to number of iterations and feasible weight in each iteration. The analysis of the graph is Enhanced Ford-Fulkerson is efficient while finding the path to destination. It takes the minimum feasible distance from source to destination.

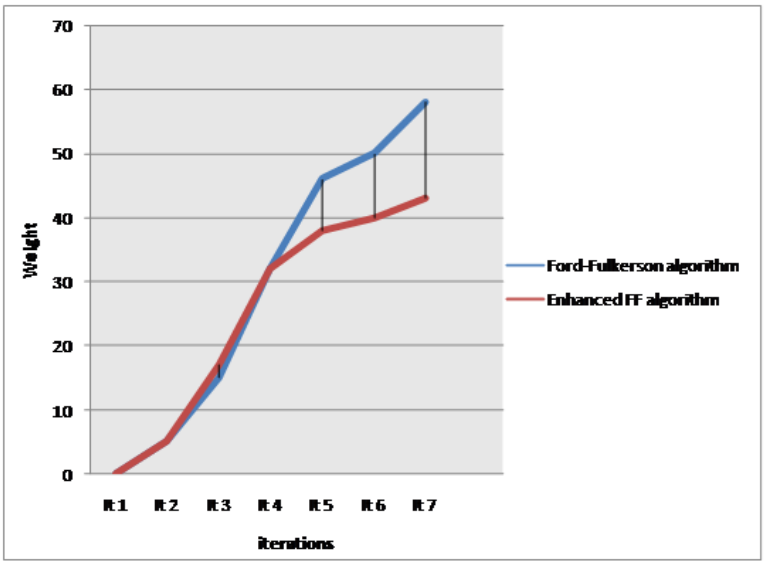

Figure 5.1. Comparison based on number of iterations.

The algorithm can also be analysed with the analysis of MapReduce job performance. The graph in the figure 3 shows the analysis of Ford-Fulkerson algorithm and an Enhanced Ford-Fulkerson algorithm in MapReduce job tracker. In the analysis, the traditional Ford-Fulkerson algorithm performance is better than enhanced Ford-Fulkerson algorithm

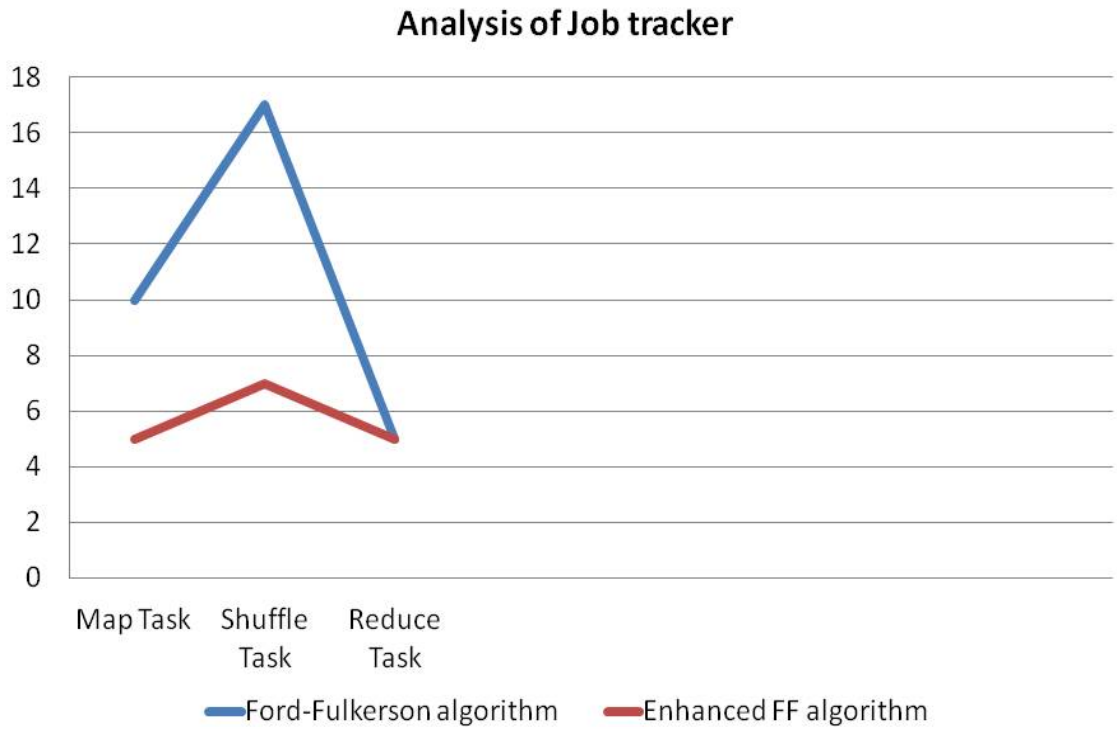

Figure 3. Analysis of job tracker.

The analysis of Ford-Fulkerson and an Enhanced Ford-Fulkerson with respect to various metrics in table 5.3.

Table 6.1: Analysis based on various metrics.

Metrics	Ford-Fulkerson Algorithm	Enhanced FF Algorithm
Execution Time	10 minute, 4 seconds	4 minute, 44 seconds
Job Analysis	Average time	Average Time
Mapper Task	10 seconds for each job	5 seconds for each job
Combiner Task	17 seconds for each job	7 seconds for each job
Reducer Task	5 seconds for each job	5 seconds for each job
feasible flow	58	43
Run Time		

6. Conclusion

In this paper, the Ford – Fulkerson algorithm and an Enhanced Ford-Fulkerson algorithm are implemented and compared based on various metrics. Finally, thesis provides an efficient way to parallelize the Ford – Fulkerson algorithm using Hadoop framework. The algorithm works effectively with large scale network graph with thousands of vertices and edges. The traditional algorithm gives quadratic run time complexity, while the proposed gives less than quadratic run time complexity.

5. References

1. F. Halim, R H.C. yap and Yougzheng Wu "A MapReduce Based Maximum flow Algorithm for large small world network graph" National University of Singapore, IEEE, Vol. 2, pp 6-7, 2014.
2. U. Gupta and L. Fegars "MapBased Graph Analysis on MapReduce" University of Texas at Arlington ,IEEE International Conference on Big Data, Vol. 10, pp 2-8, 2013.
3. KyongHa Lee, Hyunsik Choi and Bongki Moon "Parallel Data Processing with MapReduce: A Survey" SIGMOD Record, Vol. 40, pp. 12-15, 2011.
4. Euclides Pinto Neto and Gustavo Callou "An approach based on Ford Fulkerson algorithm optimize network bandwidth usage" IEEE, vol-2, pp. 12, 2015.
5. Andrew V. Goldberg and Robert E. Tarjan "Efficient maximum flow algorithm communications of the ACM", IEEE conference vol. 57, pp.2- 8, August 2014.
6. Jimmy Lin and Michel Schatz "Design Patterns for efficient graph algorithms in MapReduce", university of Maryland, vol. 5, pp. 1-12, 2014.

7. Chris Dyer and Jimmy Lin "Data – intensive text processing with MapReduce" International conference on bigdata, vol. 1, pp. 2, 2010.
8. Chintan jain, Deepak Garg "An Improved EdmondsKarp algorithm for network problem", international conference on network security, vol. 3, pp. 2, 2013.
9. Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Cluster", ACM digital library, vol.6, pp. 12, 2010.
10. Tomasz Lajdanowicz, Przemyslaw Kazienko, Wojciech Indyk "Parallel Processing of large Graphs", science direct, vol. 3, pp. 12, 2014.

Corresponding Author

Raghu B Hemanth

Email: *raghubasoor@gmail.com*