Department of Computer Science and Engineering,
ANNA UNIVERSITY CHENNAI - 600025

CS6301 – Machine Learning Laboratory

Mini Proj TITLE

NAME	REG.NO	AADHAR NO	E-MAIL	MOBILE NO

INSTRUCTOR & MENTOR: **Dr AROCKIA XAVIER ANNIE R**

Contents

1. ABSTRACT (300 to 500 words)

Every project needs to have a base on which the foundation is laid for future development and the base has to be built of new bricks and not any used stones. Thus we have chosen a novel project which uses the most recently released Big-Data Analytics tool "**APACHE SPARK**" and it is the present trending tool of Apache. The reason for it being so powerful is that it is 100 times faster than the Hadoop's Map-Reduce version in memory and 10 times faster than the latter on disk. Also the core of Big Data analytics is performed on unstructured data as parallelizing unstructured data is very easy, these are also called "Embarrassingly parallel problems". The more difficult part of Big Data Analytics is processing structured data and this is the crux of our project as we process large graphs. We achieved this using Google's Pregel Framework which will be dealt with below.

As to what our project does- " It will analyse a large network's traffic from a source to sink and this large network is mimicked by our Log Normal Graph generator function. The output is an integral value denoting the largest traffic flow possible without congestion between the source and the sink node". It is based on the Edmonds-Karp algorithm to compute Maximum flow on a single system. We have implemented a distributed version of this algorithm. The advantage of this project is that it will handle graphs of any scale. We have deployed it on a Stand Alone Mode cluster which is configured manually, thus the scale can be maximised to any extent and its run time is also discussed below**. Our promise was upto 10,000 nodes but now we have extended the project up to 25,000 nodes and we should be able to run higher classes as well**. The applications of Maximum flow vary from Computer Vision to Oil Companies and we are hoping to add this as a framework for flow analysis on large-graphs.

2. INTRODUCTION

While it is possible to use a very large and expensive machine with TB-sized memory, it is more economical and practical to use a cluster of machines alongside Big Data Analytics or Cloud computing. This naturally requires dealing with distributed programming issues, e.g. data distribution, load balancing, scheduling, fault tolerance, communication, etc.

To help us with this large-scale analysis, Spark is our base. It was introduced by Berkley University and very recently Spark was made open-source by Apache and from then on it has been the most active project at Apache, making it more the reason for our choice of project.

In recent years, the trend has been to use Hadoop for any kind of analysis. But the reason for our movement to Spark is that as mentioned before compared to the iterative model of MapReduce in Hadoop, Spark has almost 100x speed in graph analysis due to its In-memory Capability using GraphX library, RDD and so on.

Flow-Analysis can be done using various algorithms. Below in the references section is the Wikipedia link which provides with various algorithms and their complexities.

We will be using the Edmonds-Karp Algorithm for our flow-analysis as it has a run-time complexity of O(VE^2). The reason to choose this is that the algorithm can scale very well for large graphs and it is perfectly apt for our project as we need big-data graph analytics to be done on spark along with division of the graph as well. Having said this, the Edmonds-Karp algorithm is an implementation of the Ford-Fulkerson method along with a shortest path method added at the start. All these will be explained in detail further.

There are 5 modules in our project:-

1) Spark set-up.
2) Shortest Path Calculation in Scala.
3) Edmonds-Karp Algorithm in Scala.
4) Cluster Implementation.
5) Result Analysis.

Let us first introduce as to what is the maximum flow problem,

- A maximum flow problem is defined on a flow network.
- A flow network will consist of
    - Di-graph (directed graph)
    - Weights with each edge (Essential)
    - Two vertices marked
        - Source – No incoming edges
        - Sink – No outgoing edges.

Finding maximum flow in a network has been a vastly studied problem in networks and every day there are combinatorial optimizations that are attempted upon this intriguing problem.

The maximum flow problem is so powerful that it can be reduced to solve the maximal matching in a bipartite graph where each of the edges could either have a weight of 1 or any other priority set. This is something that must be thought of as Maximal matching is the crux of many large-scale problems that are solved i.e Matrimonial matching, Job matching, Smart Fridge optimisations etc.

Coming back to our problem of maximum traffic flow in a large-scale network, we have slightly modified the definition of our network and have used the Edmonds-Karp Algorithm.

- The sink can have incoming edges.
- The source can have outgoing edges.

We have introduced this because in a very large graph it will not suffice to just analyse only those pair of vertices which have no incoming and outgoing edges as at any point one node might become of a much higher consequence than any other node in its local sub-graph. Also our code is capable of handling self-loops and parallel edges which the original maximum flow problem will not have accounted for. Thus we bring about these two contributions to the well-defined maximum flow problem.

The details as to how maximum-flow algorithm runs and its implementation is provided below in further sections. Next we will introduce “SPARK”.

2.1 SPARK

As said by MateiZaharia - the author of Apache Spark (quoted with a few changes)

“One of the Spark project goals was to deliver a platform that supports a very wide array of diverse workflows - not only MapReduce batch jobs (there were available in Hadoop already at that time), but also iterative computations like graph algorithms or Machine Learning and also different scales of workloads from sub-second interactive jobs to jobs that run for many hours.”

Whenever Graph algorithms are run, less saves to disk or network transfers will always lead to a much better performance and this is the most trivial advantage that spark provides us with.

To put it in a simple line Spark can be quoted as Hadoop ++ , (just like c and c++) where it provides the user with RDD's or resilient distributed database which is similar to the Collections API that is exposed in Java, Scala , Python etc. The spark shell provides you with the much needed interactions with the Spark programs you are executing and also the cluster can be controlled via the same spark shell using various script commands.

Spark can read data from many sources such as File Systems, NoSQLdata , relational databases and so on. For machine learning enthusiasts spark provides them with the Mlib library, for us the graph algorithm workers it provides the GraphX library and so on. All this can be referred from the above diagram.

Spark has the ability to store as much data as possible in memory and this provides a much greater scale of data reuse which will drastically reduce the run time of your project when compared to Hadoop.

While the entire Spark configuration setup and internals is out of bounds in this particular document we provide a high-level abstraction as to how Spark can be setup and what are the various instantiations and objects that are available to us.

3. LITERATURE SURVEY

1. Halim, Felix, Roland HC Yap, and Yongzheng Wu. "A mapreduce-based maximum-flow algorithm for large small-world network graphs." *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011

   Advantage:-

   - Effectively parallelize a max-flow
   - algorithm based on the Ford-Fulkerson method on a cluster
   - using the MapReduce framework.
   - The algorithm is the first of its kind - a parallel and distributed map-reduce version of the max-flow.
   - They have great results and also their graphs against various rounds of map-reduce is a good result. They have detailed about scaling as well.

   Our Enhancement of a kind on this paper if compared is:-

   Though we have run smaller graphs, because of the use of Spark we estimate that our run-time will be much lesser and also as we use log normal generators only our graph properties are similar to their properties as well. Thus showing how Spark will take or has taken over Hadoop**.**

2. Zhipeng Jiang, Xiaodong Hu, and SuixiangGao. A parallel ford-fulkerson algorithm for maximum flow problem. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

   In this paper, they give a parallel Ford-Fulkerson algorithm, which is simple to implement. Different from the intuitive idea, the basic idea of their algorithm is processing all the arcs simultaneously in the step of searching for a flow augmenting path. The simulation result show that the given parallel algorithm have a great improvement of the computing time.

   Our Approach:

   We have implemented a distributed version of the Edmond Karp's Algorithm which is based on the ford fulkerson method. Instead of processing all arcs simultaneously, the input itself is split among the worker nodes for parallel processing, resulting in greater effeciency

3. **Google Pregel Paper**:

   Pregel is a programming model specifically targeted to large-scale graph problems. It's useful for two reasons:
   Ease of programming: The programming model is natural when working with graphs because it makes vertices and edges first-class citizens and encourages programmers to think in those terms, rather than in terms of dataflows and transformation operators on parts of the graph.
   Efficiency on graph problem: The framework is designed to support iterative computations more efficiently than MapReduce, because it keeps the dataset in memory rather than writing it to disk after every iteration. This is useful since many graph algorithms are iterative. It also handles the fact that graph algorithms generally have poor memory access locality, by locating different vertices on different machines and passing messages between machines as necessary.
   These above points suggest the usefulness of the pregel paper which we have used to the fullest extent.

   Our improvement

We are implementing the max-flow algorithm which has not been specified in the pregel paper but it follows a similar model to the shortest path problem and we have taken this inspiration for our paper.

4.PROBLEM STATEMENT

To deploy the distributed version of the maximum flow algorithm on a cluster using a Big Data Analytic Tool (Spark) which has an iterative model of computation to find the maximum possible traffic flow in a Large graph

Given a flow network which is modelled as a directed acyclic graph with edges representing the traffic between the nodes and the vertices being unique routers across the network. We are to find out the maximum amount of traffic or packets that can be sent across the network starting from one router to another router without any packets being lost. For example, consider the graph given below :

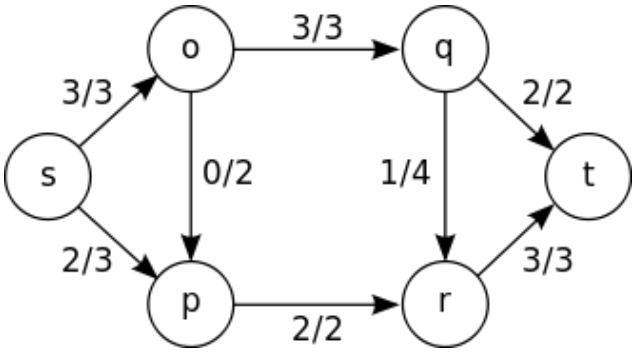


FIG 1

In the above given simple graph , the edges represent maximum number of packets that can be sent in that path. The notation a/b on the edge is used where ‘a’ tells the amount of packets to be sent and ‘b’ is the maximum number of packets that can be sent across the end routers. The output,that is the maximum flow of packets possible across the above given network is 5

5. PROBLEM SOLUTION

5.1 DATA SET:-

The Data Set has been generated by two ways:

- Random graph generated using Matlab
- A Log Normal Graph generated using the LogNormal graph generator function available in Spark

5.2 INPUT :-

The input are 2 text files given to the SparkContext to run the algorithm on it.

5.3 APPROACH:-

The two text files are a vertex text file and an edge text file.which has all the edge pairs. This is a different representation of Graphs which is generally used in Big Data Analytics.This is how the data set is represented and given as an input to spark. Then we will run our Edmonds-karp algorithm implemented in Scala to find the flow of the graph and also will use shortest path calculations from the source to the sink, to compute the augmenting path and construct the residual graph.

These text files will be passed into our Spark Tool and will be processed using Pregel, GraphX and our implementation of Max-Flow Analysis.

Once the graph flow has been processed, the output will be an Integer indicating the flow across the graph without any loss or leakage.

The vertices of the graph would represent different routers or nodes in a network and the edges would represent the amount of current traffic. Applying the Edmond-Karp algorithm, We would tell the maximum number of packets that can be safely sent without causing loss of packets ie we would say what is the maximum capacity the network can handle without causing it to overflow.

6. NOVELTY

Big Data Analytics on any structured data is always a task comprising higher implementation challenges and novel ideas when compared to unstructured data. **Our deployment on a cluster provides a user with our framework to compute maximum-flow on any graph including parallel edges and self-loops**. Our novelty constitutes of relating this to real-world problem where we believe the maximum traffic-flow value can be added as a quality parameter in large networks. **Also the same output can be used for dynamic signalling in a large-network to avoid congestion mechanisms itself. The technical novelty is by looking at the run-time analysis which is portrayed as a graph where for even graphs as large as 25,000 nodes and 35,59,000 edges the run time is 1150s**. These run-times are reasonable in real-world applications and can be compared to any other implementation which provides such flow analysis results.

7. ARCHITECTURE

7.1ABSTRACT ARCHITECTURE

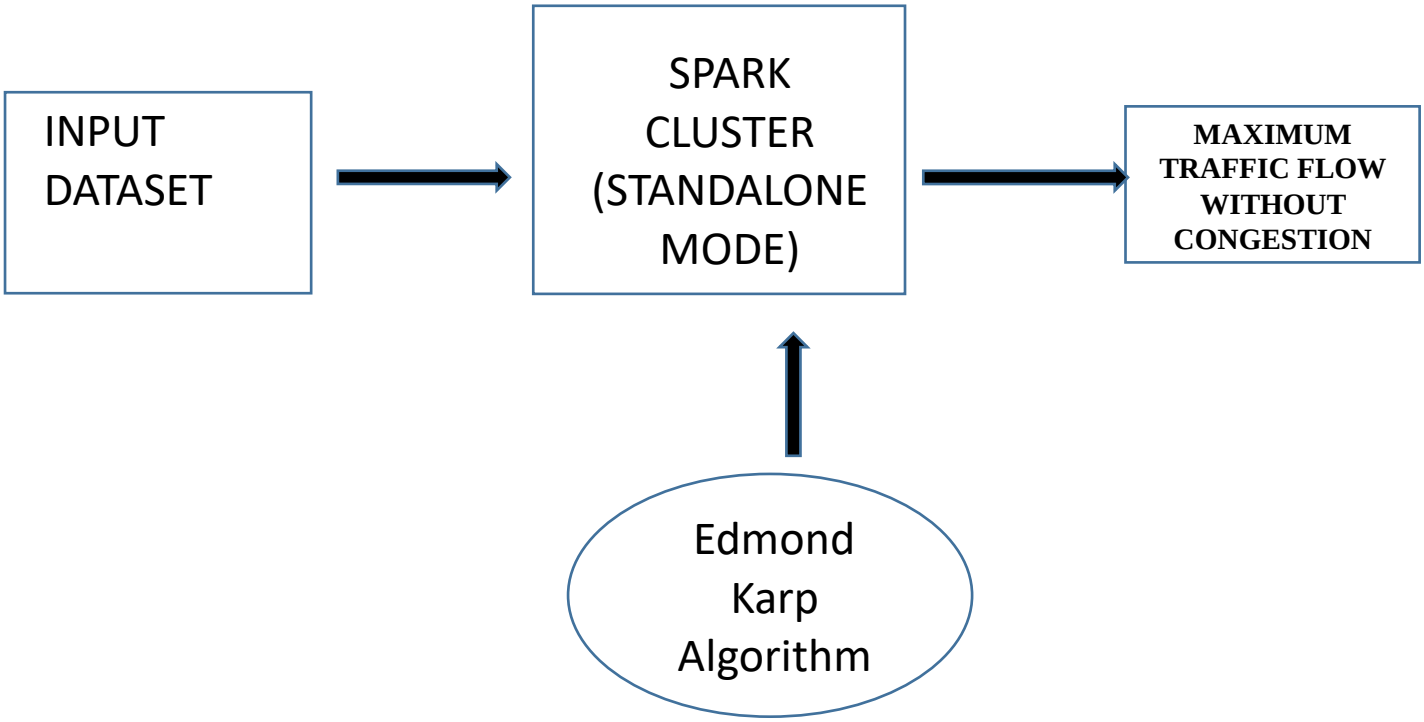


FIG 2

This is an abstract view of our project. Given the input data set as two text files of vertices and edges, we deploy the Spark Standalone Cluster mode and run our Edmonds-Karp algorithm on the given large network graph. The Edmonds-Karp algorithm will be explained in detail later. The output of the algorithm will be an Integral Value providing us with a maximum value of traffic or packets or data based upon the edge value that is given as input to us. Based upon this result we can analyse various scenarios.

7.2 DETAILED ARCHITECTURE

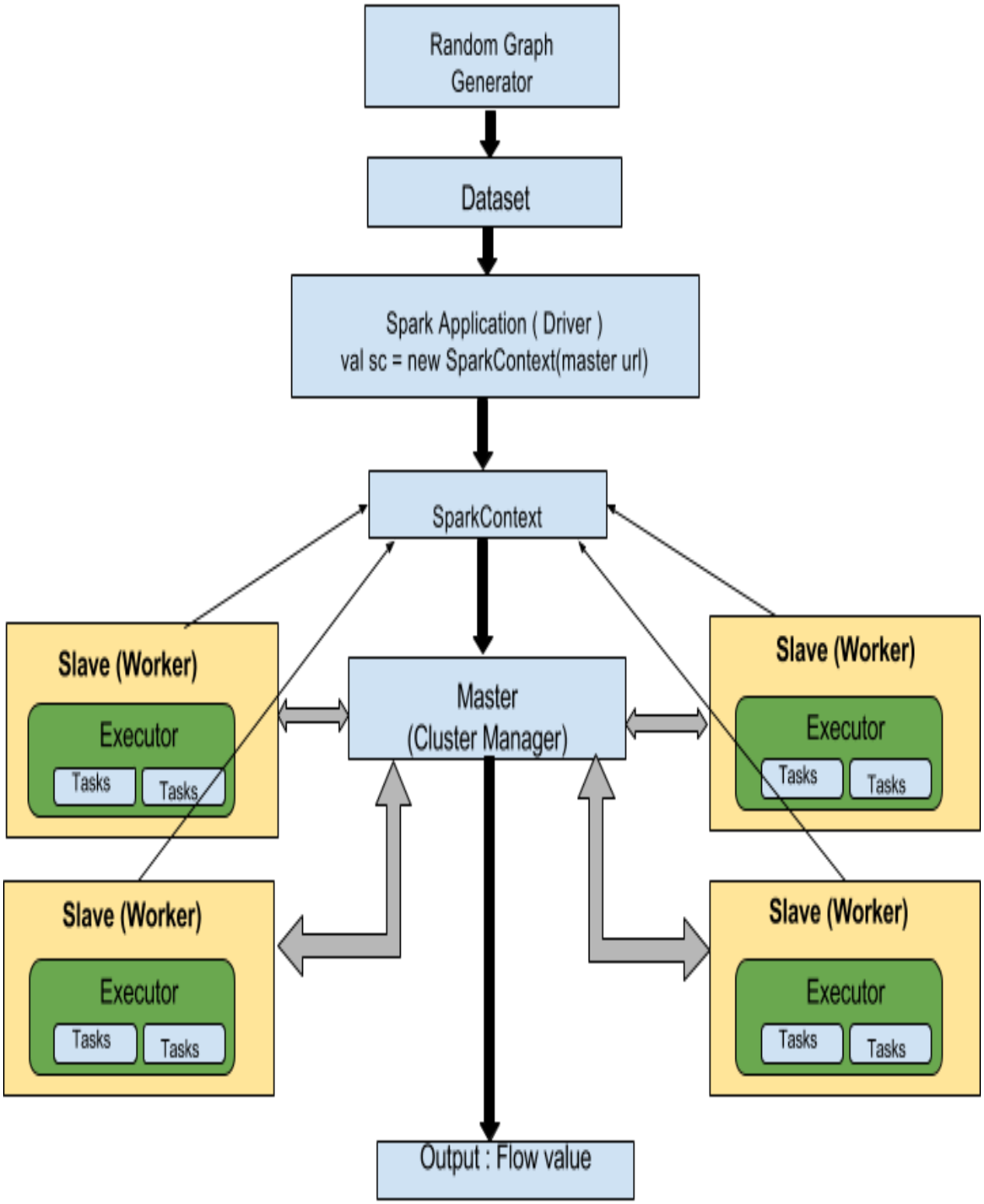


FIG 3

7.3 EXPLANATION OF THE ARCHITECTURE

Random Graph Generator:

This block represents the Generation of the data set using Matlab or the Log Normal graph generator in Spark. It is essential to generate the input which resembles a large network in the real world.

Data Set:

This represents the two text files, a vertex text file and an edge text file. These are given as input to the application. If the Log Normal Graph generator is used then the data set is created inside the application itself.

Spark Application (Driver) :

A Spark driver is the process that creates and owns an instance of SparkContext. It is your Spark application that launches the main method in which the instance of SparkContext is created. It is the cockpit of jobs and tasks execution (using DAGScheduler and Task Scheduler). It hosts Web UI for the environment.

Spark Context:

SparkContext (aka Spark context) represents the connection to a Spark execution environment (deployment mode).You have to create a Spark context before using Spark features and services in your application. A Spark context can be used to create RDDs, accumulators and broadcast variables, access Spark services and run jobs. A Spark context is essentially a client of Spark's execution environment and acts as the master of your Spark application .

Master:

A master is a running Spark instance that connects to a cluster manager for resources.

The master acquires cluster nodes to run executors.

Slave (Worker):

Workers (aka slaves) are running Spark instances where executors live to execute tasks. They are the compute nodes in Spark.

Executor:

Executors are distributed agents responsible for executing tasks. They typically (i.e. not always) run for the entire lifetime of a Spark application. Executors send active task metrics to a driver. They also inform executor back ends about task status updates (task results including).

Tasks:

Task (aka command) is an individual unit of work for executors to run. Tasks are like basic processes which must be executed on each node that has been deployed on your cluster andeven your master can chose to have a worker and run the task of synchronizing the workers as well as run the application that has been developed.

Also before we move forward we would like to provide with your directory structure as well which is very important for spark applications

YOUR_PROJECT_FOLDER/SRC/MAIN/SCALA/CODE.SCALA

YOUR_PROJECT_FOLDER/SRC/DATA/Datasets

After packaging you will also find a target sub-folder and classes and sub-folder and inside the target sub-folder you will be able to locate your packaged JAR file which is then run using the sbt run command.

The task for our application has been explained in detail below:

7.4 TASK ARCHITECTURE

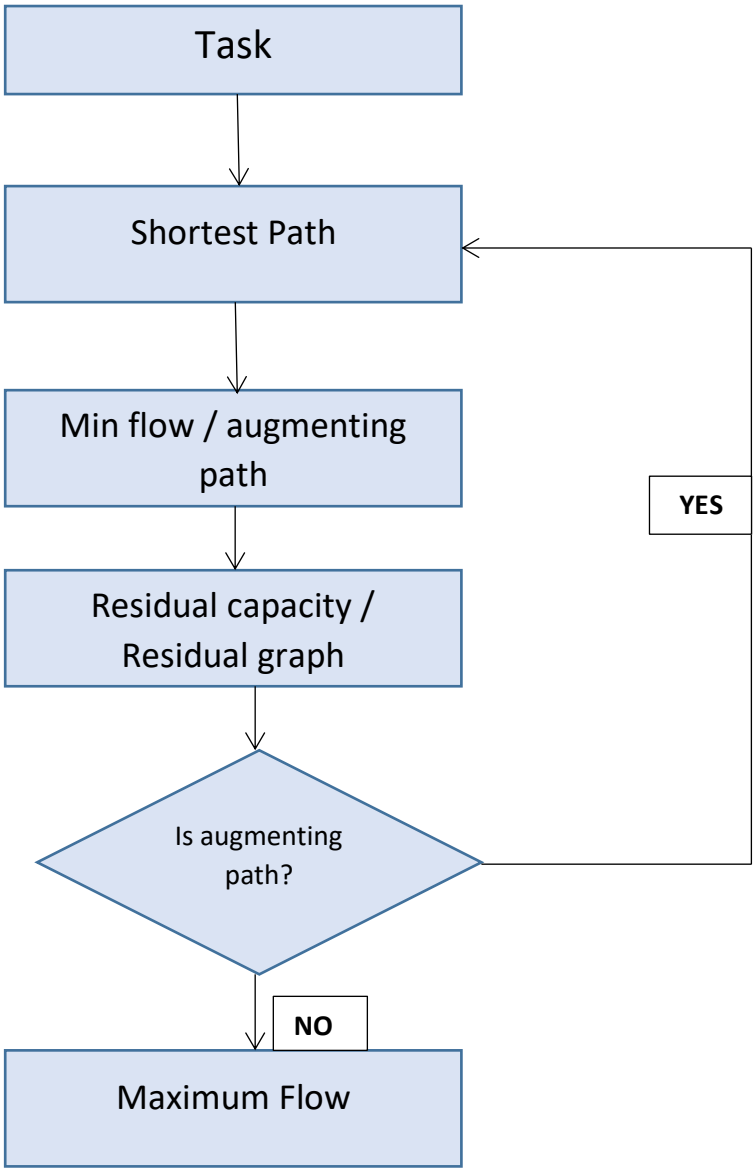


FIG 4

7.5EXPLANATION OF THE ARCHITECTURE DIAGRAM

Shortest Path:

To find the shortest path between source and sink - This is the first step of the Edmond-Karp's Algorithm to calculate maximum flow. We use Google's Pregelapi to calculate the shortest path as it is a distributed system.

Min Flow/Augmenting Path:

An augmenting path is a path in a graph which only has positive capacities moving from source to the sink without cycles. Min Flow is the minimum flow available in that path as that is it's the maximum capacity that can be sent without overflowing the network.

Residual Capacity/Residual Graph:

A residual graph is almost like a complement of our input graph where all the flows are subtracted from the capacity of the edges thus providing more possible flows across our graph.

A residual capacity is the capacity left after subtracting the mn flow from each of the edges in the path.

Is Augmenting Path?

The algorithm will continue till an augmenting path exists.

Maximum Flow:

The output of the application indicating the maximum possible flow without loss of packets.

.

8. DETAILED MODULE DESIGN

8.1 SHORTEST PATH:

A typical Pregel computation consists of input, when the graph is initialized, followed by a sequence of super steps separated by global synchronization points until the algorithm terminates, and finishes with output.

Algorithm:

In each super step, each vertex receives messages from its neighbours and updates potential minimum distances from the source vertex. If the minimum of these updates is less than the value currently associated with the vertex, then this vertex updates its value and sends out potential updates to its neighbours, consisting of the weight of each outgoing edge added to the newly found min distance.

This is very similar to our normal shortest path algorithm but pregel gives it a distributed flavour by adding super-step and a message synchronization process to it.

The below given Pseudo code is picked from the Pregel Paper (Google's paper)

```
classShortestPathVertex
: public Vertex<int, int, int> {
void Compute(MessageIterator* msgs) {
intmindist = IsSource(vertex_id()) ? 0 : INF;
for (; !msgs->Done(); msgs->Next())
mindist = min(mindist, msgs->Value());
if (mindist<GetValue()) {
*MutableValue() = mindist;
OutEdgeIteratoriter = GetOutEdgeIterator();
for (; !iter.Done(); iter.Next())
SendMessageTo(iter.Target(),
mindist + iter.GetValue());
}
VoteToHalt();
}
};
```

After this a basic Combiner code must be added to be able to pick up the minimum of all values as there will be various messages with distances incoming at a particular node.

The above algorithm has been reduced into Scala Implementation by our team and its code snippet is mentioned in the implementation section.

8.2 MAXIMUM FLOW MODULE

We have to indicate two nodes as Source and Sink. Between the source and sink we have to find the maximum traffic flow possible.

The two simple constraints that should be followed in a flow computation are:-

1) The flow on the edge should be less than or equal to the capacity.
2) Flow into an edge = Flow out of the edge. (No leakage) .

The algorithm for maximum flow is:

Set Flows across all edges to null.

Let Residual Graph = Initial Graph.

While(Augmenting Path)
{
    Find shortest path augmenting path in Residual graph.
    Find min. flow across the path such that there will be no overflow.
    Broadcast this flow across all nodes.
    Update the flows and residual graph.
}

The above algorithm is a very high-level overview of the idea and each line of this algorithm will include many other detailed steps implemented in Scala.

8.3 CLUSTER

The cluster module will have two main parts – master and slaves. The master URL is a very important parameter which must be noted and then the slaves must be provided with their data to process, and this processing is done by the same algorithm written in the master. The setup and many of the launch scripts used in the cluster implementation will be explained in the next section. To get a better view of the cluster implementation, the screenshots of the web UI have been added below:

9. IMPLEMENTATION

9.1 INITIAL SET-UP:

Spark Set up:

- Download Spark, Scala and Java in your Linix based system.
- Install git as Spark requires git for its dependencies
- Install sbt and open-ssh-server for your project compilation and execution on a cluster.
- After running the sbt/sbt assembly command on your terminal, and the spark-shell command you must be able to see the spark-context and sql-context.
- The spark context constitutes your spark application and after this you can begin coding your application
- The deploy mode will provide you with either local deployment or cluster deployment

.9.2CLUSTER SETUP:

1. Configure the "spark-env.sh" file

2. Create conf/slaves.sh add all the hostnames of spark slave nodes to it like mentioned below.

<HOSTNAME OF YOUR MASTER NODE>

<HOSTNAME OF YOUR SLAVE NODE 1>

...

...

<HOSTNAME OF YOUR SLAVE NODE n>

[Repeat same above step 1 and 2 on other slave nodes]

3. Start/Stop Spark using below commands

$ sh /opt/spark-1.5.1-bin-hadoop2.6/sbin/start-all.sh

$ sh /opt/spark-1.5.1-bin-hadoop2.6/sbin/stop-all.sh

9.3 TOOLS:

- Spark
- SBT

9.4 CODE SNIPPETS:

9.4.1 Shortest Path Logic

```
(id, dist, newDist) => {
if (dist._1 < newDist._1) dist
elsenewDist
   },

   // send message
triplet => {
     // If distance of source + 1 is less than attribute of destination => update
     // If capacity along edges <= 0; do not propagate message
if (triplet.srcAttr._1 + 1 < triplet.dstAttr._1 &&triplet.attr> 0) {
Iterator((triplet.dstId, (triplet.srcAttr._1 + 1, math.min(triplet.srcAttr._2, triplet.attr), triplet.srcId)))
     }
else {
Iterator.empty
     }
   }
```

```
val v = sssp.vertices.take(vNum) // set shortest path to an array
```

9.4.2 Maximum Flow main logic

```
while (path != empty) {
valbcPath = sc.broadcast(path)

   // Update the flows
valupdateFlow = minCap
valnewFlows = flows.map(e => {
if (bcPath.value contains e._1) {
       (e._1, e._2 + updateFlow)
     }
else {
e
     }
   })
```

9.4.3 Residual Graph

```
valnewEdges = residual.edges.map(e => ((e.srcId, e.dstId), e.attr)).
flatMap(e => {
if (bcPath.value contains e._1) {
Seq((e._1, e._2 - minCap), ((e._1._2, e._1._1), minCap))
     }
elseSeq(e)
   }
     ).reduceByKey(_ + _)
```

9.4.4 Cluster Linker

```
valsc = new SparkContext("spark://192.168.1.9:7077","Complex-Max-FLow","/home/spark-
1.6.0",List("target/scala-2.10/spark-maxflow_2.10-1.0.jar"))
```

9.4.5 Cluster commands:-

./start-all.sh for the master

./start-slave.sh spark://master-url:port

./stop-all.sh

sbt package

sbt run

9.4.6 Configuring the scripts:

Assign your master_ip in SPARK_MASTER_IP in the spark-env.sh file

In the slaves.sh file assign all the IP'S of the worker nodes

Add the spark master URL to the constructor of SparkContext in the code.

This should run your application and yield your expected result on a cluster.

These above codes are just code snippets and contain the main functionality of our application.

10. SUPPORT INFORMATION

https://snap.stanford.edu/data/

http://www.scala-sbt.org/0.13/docs/Installing-sbt-on-Linux.html

http://nishutayaltech.blogspot.in/2015/04/how-to-run-apache-spark-on-windows7-in.html?view=sidebar

http://blog.prabeeshk.com/blog/2014/10/31/install-apache-spark-on-ubuntu-14-dot-04/

http://blog.knoldus.com/2014/06/04/a-simple-application-in-spark-and-scala/

11. SURVERYED CONTENT

http://spark.apache.org/

https://en.wikipedia.org/wiki/Maximum_flow_problem

http://www.tutorialspoint.com/scala/

http://www.cs.princeton.edu/courses/archive/spr04/cos226/lectures/maxflow.4up.pdf

http://stackoverflow.com/questions/23700124/how-to-get-sssp-actual-path-by-apache-spark-graphx

https://www.quora.com/What-is-the-difference-between-Apache-Spark-and-Apache-Hadoop-Map-Reduce - hadoopvs spa

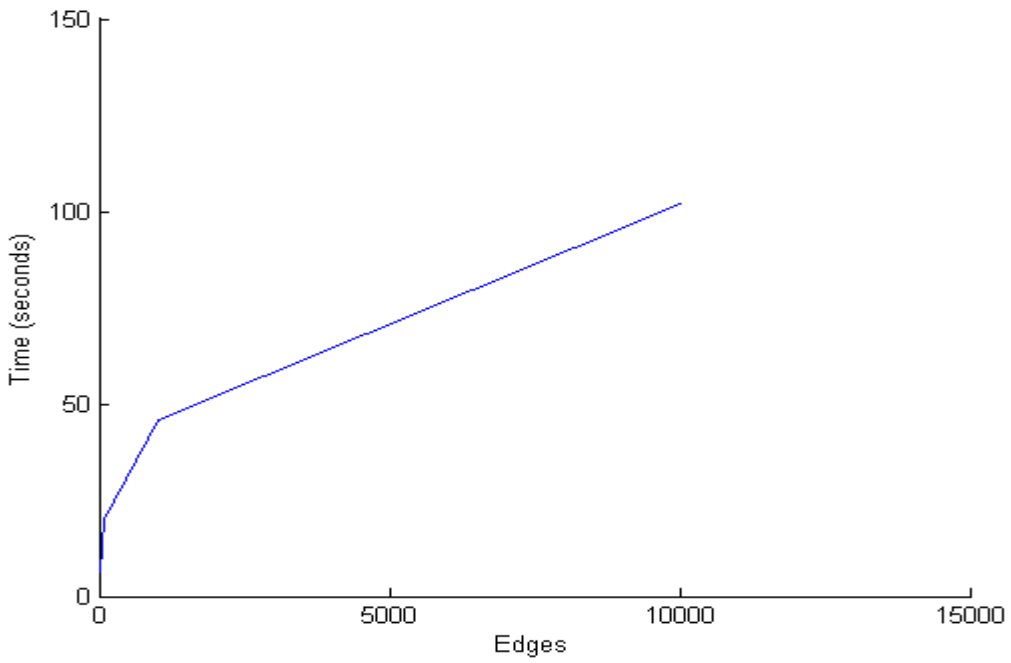
12. RESULTS AND COMPARISON

The below graph provides details of various graphs that were tested. Both matlab generated random graphs and lognormal graphs were tested and the run time has been plotted against the number of edges.

We have selected a few cases that we tested and plotted a graph for better clarity. The run times shown here are those obtained by running on a cluster of 3 machines with each of them running a worker. **The total RAM in our cluster was 12.2 GB and it had been deployed on 12 cores with 4 cores on each system**. Also it is worth mentioning that the cluster time is greater than a single machine system due to the communication cost between the edges of the graph which theoretically can scale upto the number of edges but practically is much lesser.

Analysing this output we can see that when our model is deployed on a much larger and powerful cluster the run time can be brought down much more and all that will matter is some amount of communication cost**. This can be applied to large networks to either dynamically signal routers on what path to choose (signalling) or can be used as a bandwidth performance quality measure.**

It should be mentioned again that run time is highly dependent on the structure of the graph and that is why we have run our application on log normal graphs too so that our model can cope up to very large network properties as well. Also, as we can see from the graph, the run time will scale up almost linearly as the number of edges increase



The times are averages of many small graphs and thus is almost linear but in practice it will scale with the edges and can be exponential based on the graph structure.

FIG 5

13. CONCLUSION AND FUTURE ENHANCEMENTS

The contribution of this project is a model suitable for large-scale network traffic flow analysis and a description of its fault-tolerant implementation via spark. Dozens of log -normal graphs of variuos sizes have been generated and deployed through our algorithm and spark-interactive shell. The largest graph tested had **25000 Vertices and 35,57,776** Edges which is in sound comparison with any other such model. Partitioning of the input graph based on topology may suffice if the topology corresponds to the network traffic and thus the use of pregel framework is also justified. **The problem scales as the number of edges increase as the nodes fit on a machine**. The complexity from the sequential version of o(cm) has been reduced to **o(cm/k) + communication cost**. (**where 'c' is max-flow and 'k' is machines**). **The communication cost in practical terms is much lesser than the order of edges and it depends largely on the structure of the graph generated** .In the future we are wanting to add a blog on Spark applications and especially on GraphX library usage as through this project the development of code was one part and the understanding of Spark and its cluster deployment scripts was another big part and thus a blog will be helpful to future users. Also if our model sustains many more inputs from other users with their datasets, then with a few modifications and input parameters from the user including - Master URL, Slave IP'S , Graph as vertices file and edge file, we can add this as a function in the graphX library of spark. All said this process would still require immense testing as we have handled random graphs with integral weights.

14. LIST OF REFERENCES

- Halim, Felix, Roland HC Yap, and Yongzheng Wu. "A mapreduce-based maximum-flow algorithm for large small-world network graphs." *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011.
- Ahlswede, Rudolf, et al. "Network information flow." *Information Theory, IEEE Transactions on* 46.4 (2000): 1204-1216.
- Lee, Youngseok, Wonchul Kang, and Hyeongu Son. "An internet traffic analysis method with mapreduce." *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE, 2010.
- https://spark.apache.org/ - spark home page
- http://spark.apache.org/downloads.html- download spark (pre-built) / (sbt build)
- http://spark.apache.org/docs/latest/spark-standalone.html- standalone mode
- cakesolutions - graphx/pregel
- http://www.datasense.ninja/spark-scala-graphx-first-graph-program/- scalagraphx
- http://blog.knoldus.com/2014/06/04/a-simple-application-in-spark-and-scala/ - scala application blog
- https://www.cs.purdue.edu/homes/ayg/CS251/slides/chap10c.pdf

15.APPENDIX A:

The undersigned acknowledge they have completed implementing the project “Traffic Analysis in large network graphs in spark” and agree with the approach it presents.

Signature: _____ Date: 04/04/2016

Name: Sanjana Jain S

Signature: _____ Date: 04/04/2016

Name: Shivanee N

Signature: _____ Date: 04/04/2016

Name: Varun R

16. APPENDIX B: REFERENCES

The following table summarizes the research papers referenced in this document.

Document Name and Version	Description	Location
A mapreduce-based maximum-flow algorithm for large small-world network graphs	Hadoop based max-flow analysis and results of various stages and rounds.	ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5961676
An internet traffic analysis method with mapreduce	Traffic is analysed based on various features using mapreduce stages.	ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5486551
Network information flow	Work indicating on how information would flow in a large network.	people.csail.mit.edu/arasala/papers/networkFlow
Pregel: Large Graph Processing System by Google	The first of a kind, graph processing system which works at a higher efficiency when compared with mapreduce.	https://kowshik.github.io/JPregel/pregel_paper

17.APPENDIX C: KEY TERMS

The following table provides definitions for terms relevant to this document.

Term	Definition
Big Data	**Big data** is a term for **data** sets that are **large** or complex that traditional **data** processing applications are inadequate
RDD	Resilient Distributed Database is a format used for storage in Spark
Maximum Flow	Given a network with A **flow** network, with source s and sink t, maximum flow is the largest value of units that pass across without any overflow.
LogNormal	Logarithm of the Normal Curve is taken (bell-shaped curve).
Augmenting Path	An **augmenting path** is a simple **path** - a **path** that does not contain cycles - through the graph using only edges with positive capacity from the source to the sink.