**Introduction**

In this example, we will try to extract the communication network of Enron from a large Enron email corpus.

More details of the implementation are given in the respective sections.

**Setting up**

A group directory named SUV is created in /user/shared/ on HDFS.

The split email data are copied to the location /user/shared/SUV/splits/

A Java project is implemented with the name ‘enron’ and jar file for the same is available

in /user/shared/SUV/ named enron.jar.

**Extracting the Email Network**

The email data goes through mainly three stages where in the initial stage, *from*, *to*, and *timestamp* are extracted from the email headers. In the second stage, we extract the social networking from the parsed data from the stage1 and represent the data as a directed weighted graph. In the third stage we calculate the in-degree and out-degree distribution of the above data. Following are the classes we designed for the above mentioned stages:

**Extracting Email Network: (Stage 1)**

* **EmailExtractor:**

This the driver class. This class takes the email corpus and extracts *from*, *to*, and *timestamp* as tab separated value to output files. It goes through a Mapper-Reducer function where mapper emits the tab separated *from*, *to*, and *timestamp* as key and a null as value. The reducer then just picks up the key and simply ignores the values (as this will be a list of null values) and writes the values to output files. This way we eliminate all the duplicates at this level itself.

* **EmailExtractorMapper:**

This class takes the pre-processed email messages and does the necessary jobs to extract the “from”, “to,” and “timestamp” from the message header. The “to” consists of a list of recipients from To, Cc and Bcc fields of a message. The mapper emits a tab separated string of from, to, and timestamp for each of the “to” values in the list as key and a null value as value.

* **EmailExtractorReducer:**

The reducer then just picks up the key and simply ignores the values (as this will be a list of null values) and writes the values to output files.

**Extracting Social Network**

**Extracting Social Network: (Stage 2)**

The implementation consists of a driver class SocialNetworkingExtractor which handles all the map/reduce tasks to extract the social networking from the data output by the Stage1. This class has nested classes for other operations for which the following classes are designed.

* **SocialNetworkExtractor:**

This is the main driver class for our job and it handles all the other necessary sub tasks associated with the extraction. The resultant data of the first stage goes through four jobs of Mapper-Reducer function, where in the first job it again eliminates the duplicate and in the second job we do some clean up by filtering out the non-enron emails. There are some dubious looking email accounts (e.g. -4c9060@enron.com) which are not removed. In the third job, we identify the nodes (persons, or email accounts) and edges\arcs(emails) and assigns weights to the edges. In the fourth job, we again re-normalize the weights considering the maximum arc weight or maximum edge weight in our data from the second job's output. At each stage, necessary configuration data is set for the jobs to carry out calculations. Following are the classes implemented for these MapReduce sub tasks.

* **Duplicate removal job:**
  + **NodeEdgeDuplicateRemoverMapper:**

This class takes the data from the first stage and emits from, to, and timestamp as tab

separated string as key and null as value.

* + **NodeEdgeDuplicateRemoverReducer:**

Reducer picks only the key and ignores the list of null values and writes the results to output files. This job is a redundant job, but still we implemented this again to make sure that we don't treat any duplicate entries.

* **Filtering job:**
  + **EnronEmailFilterMapper:**

Here we filter out non-enron email accounts from our previous job result set. The filtering is only done for the “from” field. The mapper emits the filtered “from”, “to”, and “weight” as key and null as value.

* + **EnronEmailFilterReducer:**

Reducer just writes the key received from the mapper as it has tab separated from, to and weight to output file.

* **Weight Assigning job:**
  + **EdgeWeightAssignerMapper:**

This class takes the input from the EnronEmailFilterReducer. Each line will be a unique tab separated from, to, and timestamp value. This class splits this data and emits the “from” value as key and “to” value as value.

* + **EdgeWeightAssignerReducer:**

This reducer gets the “from” as key and a set of “to” values. It then calculates the weight (total number of emails between the “from” and “to”). We are normalizing the weight by dividing the total number of arcs(emails). The total number of arcs is set in the job configuration from the previous job. We then rescale this between 0 and 100, by multiplying the weight by 100. The results are written to output files. Here we also determine the maximum arc weight in order to carry out the calculation in the next sequence.

* **Re-Normalizing sequence:**
  + **NormalizerByMaxArcWtMapper**

This takes the data output from EdgeWeightAssignerReducer and splits the data to emit a tab separated string of “from” and “to” as key and “weight” as value.

* + **NormalizerByMaxArcWtReducer**

The reducer takes the weight and recalculate it based on the maximum arc weight and rescale the weight on 0-100. The maximum arc weight is set in the configuration from the previous job.

The reducer outputs the results as csv(comma separated value) files.

* **Enhancements:**

We use Gephi tool to visualize the Enron email communication network. Gephi recognizes csv files and one of the requirements for Gephi to correctly identify our data is to provide headers to the csv file. For this specific requirement, we have implemented a separate FileOutputFormat class.

* + **CSVOutputFormat:**

This class extends the FileOutputFormat class from hadoop.mapred package and sets the file extension as .csv.

* + **CSVRecordWriter**:

This class extends the RecordWriter class from the hadoop.mapred package and does the actual job of writing the results to a file. Here in the constructor we set the headers as “from”, “to” and “timestamp”. The results are output by the *write* method as comma separated values.

We have used the Hadoop MapReduce framework to calculate the degree distributions for both in-degree and out-degree. The details of the implementation are provided below.

**Degree Distribution (Stage 3)**

We calculate both in-degree and out-degree distribution of the email communication network data output by our Stage 2. There are two Mapper-Reducer sequences to do this job. In the first job we calculate the degree associated to each nodes and in the second job, we calculate the distribution of these degrees. We have two separate driver classes to carry out these two Mapper-Reducer functions for both in-degree and out-degree.

* **InDegreeDistribution:**

This is the driver class and it manages two jobs in turn, first to calculate the degree associated with each nodes and then to calculate the distribution for each of these degrees.

* **InDegreeMapper:**

Reads the input from the Stage 2 and emits the node (“to”) as key and a count 1 as value.

* **InDegreeReducer:**

The reducer calculates the degree by iterating through the values and sums up the counter. The result is written to output file as node(“to”) and degree (“total emails received”).

* **InDistributionMapper:**

Reads the input from the InDegreeReducer and emits the degree as key and a counter 1 as value.

* **InDistributionReducer:**

The reducer calculates the distribution by iterating through the values and sums up the counter. This is divided by the total number of nodes (“to email accounts”) to get the distribution. The total number of arcs is set in the job configuration from the results of previous job.

* **OutDegreeDistribution:**

This is the driver class and it manages two jobs in turn, first to calculate the degree associated to each node (“from”) and then to calculate the distribution for each of these degrees.

* **OutDegreeMapper:**

Reads the input from the Stage 2 and emits the node (“from”) as key and a count 1 as value.

* **OutDegreeReducer:**

The reducer calculates the degree by iterating through the values and sums up the counter. The result is written to output file as node(“from”) and degree (“total emails received”).

* **OutDistributionMapper:**

Reads the input from the InDegreeReducer and emits the degree as key and a counter 1 as value.

* **OutDistributionReducer:**

The reducer calculates the distribution by iterating through the values and sums up the counter. This is divided by the total number of nodes (“from email accounts”) to get the distribution. The total number of arcs is set in the job configuration from the results of previous job.

**Steps to run the jobs:**

Main folder: /user/shared/SUV/

Input data for the program: /user/shared/SUV/splits/

Jar file: enron.jar

**Stage 1- Email Extraction:**

hadoop jar enron.jar enron.EmailExtractor /user/shared/SUV/splits/<<destination folder>>

**Stage 2 – Social Network Extraction:**

hadoop jar enron.jar enron.SocialNetworkExtractor <<destination folder of the Stage 1>> <<destination folder>>

(This job will out two folders, one is *temp* and *final.* The temp folder contains the intermediate results of the job, final folder contains the results. Two results sets are produced according to the normalization methods, which are named *by\_max\_wt* and *by\_total\_arcs* respectively)

**Stage 3 – Degree Distribution:**

hadoop jar enron.jar enron.InDegreeDistribution <<destination folder of stage 2>>/final/by\_total\_arcs/ <<destination folder>>

hadoop jar enron.jar enron.OutDegreeDistribution <destination folder of stage 2>>/final/by\_total\_arcs/ <<destination folder>>

\*Important: Make sure to use the output files in the by\_total\_arcs folder for Stage 3. The folder by\_max\_wt

contains .csv files maily meant for the Gephi visualization.