**Project 2 Document**

Description of the main steps as well as the data flow has been included below:

**PageRankMap.java:**

In this program, we have created a PageRankMap class that extends the Mapper class. Within it we have created a map() function. The typical input to this map() function is the adjacency matrix, which comes in from the CreateGraph() method.

Within the map() function, we instantiate an object of class RandRecord that stores the information pertaining to a single webpage. Next, we calculate the rankValuePerUrl using the below formula:

**rankValuePerUrl = rankValue/numUrls**

Later, we note down the value of rankValuePerUrl. Note that, all this is done, if the size of targetUrlsList is less than 0. If not, then we create the pair <trgtURL, rankValueTrgtURL> . For this, we find the value of rankValueTrgtURL using the formula:

**rankValueTrgtURL = rankValue/sizeof(targetUrlsList)**

Later, this value is printed using System.out.println() as well as to the context object. Finally, we create a pair <sourceUrl, #targetURLs>.

Summary:

In short, the main aim of PageRankMap.java was to create pairs of <trgtURL, rankValueTrgtURL> or <sourceUrl, #targetURLs>. These pairs are then in turn passed onto the PageRankReduce.java program explained next.

**PageRankReduce.java:**

In this program, we have created a PageRankReduce class that extends the Reducer class. Within it we have created a reduce() function. The typical input to this reduce() function are the pairs of <trgtURL, rankValueTrgtURL> or <sourceUrl, #targetURLs> that are generated by the map() function in PageRankMap.java.

The strArray array that we have created contains values like <3,#1#2> and <3,#4#5>, wherein the first parameter is the key, while the second one is the list of values separated by #. One of the main aims of our reduce() function is to concatenate such records, so that the resultant records would be like <3,#1,#2,#4,#5>. Note that this is done, only if the size of the strArray is greater than 1 (meaning that we have many list of values separated by #s), meaning that we have a link relation tuple. We also sum up the rank values for display. Finally, we take into account the damping factor (d) by using its default value of 0.85. The formula used is:

**sumOfRankValues = 0.85\*sumOfRankValues+0.15\*(1.0)/(double)numUrls**

Summary:

In short, the main aim of PageRankReduce.java was to concatenate the values of the records with the same key. Also, we summed up the values taking into account the damping factor (d).

**Data Flow:**

The code basically has 3 MapReduce jobs:

1. CreateGraph:

It adds one column ‘initial pagerank value’ to the input pagerank adjacency matrix and passes it to the PageRank program which inturn calculates the pagerank values.

1. PageRank:

The code takes as input the transformed adjacency matrix and calculates the pagerank values for each and every page.

1. Cleanup results:

It removes the targetUrls column and outputs <sourceUrl, pageRankValue> as the final result.

1. Sorting:

This job sorts the various <key, value> pairs by the rank values in the descending order.

The PageRank input data is locally stored as an adjacency matrix. It is then uploaded to Hadoop Distributed File System (HDFS), and is thus distributed across multiple nodes. The Hadoop framework uses the InputFormat interface to read from the input and generate <key, value> pairs. Key here is the unique id or the name of the webpage and the value contains the rank and the out-value (fan-out) of the webpage. Thus, by consuming a single input <key, value> pair, the map function can generate zero or more <key, value> pairs. The map function applies the formula to each <key, value> pair. Thus the intermediate <key, value> pairs that they generate is the partial rank value of each web page. The reduce task in turn calculates the sum total of the webpages by using the below equation:

These aggregated rank values are then written back to HDFS. This is then used as the input during the next set of iterations, and so on.

The screen shot of the output is as below:

