**CSE587 – Data-Intensive Computing**

**Homework - #2**

Name: Harin Rishabh

UBIT Name: harinris

UB Person#: 50540017

**Part – 1**

Implemented a basic word count application that reads two files and uses PySpark to create a dictionary of words and their frequencies sorted in descending order. It is case-insensitive and ignores stopwords using a Pyspark library’s StopWordsRemover function.

**Output –**

A screenshot of a computer

Description automatically generated

**Analysis –**

1. Here, we run use PySpark REPL to run the WordCount program on only one book –
2. A screenshot of a computer

   Description automatically generatedThe 25 most common words are as follows –
3. A diagram of a diagram

   Description automatically generatedPySpark breaks up execution into stages to achieve optimal parallelism. Here, the WordCount task is broken up into 2 stages. The first stage handles the reading of the files and creation of RDDs. The second task handles shuffling, partitioning and collection.

A diagram of a program

Description automatically generatedA diagram of a computer

Description automatically generated

**Part – 2**

In the second part, we read the two generated text files containing graph information. We run Dijkstra’s algorithm on both these graphs concurrently. After we calculate the distances from vertex 0 to every other vertex in both graphs, we sort them by descending order of distances and output them into a single text file.

2. In graph 1, vertex 0 to vertex 2 has a distance of 9. This is the largest. The distance from vertex 0 to several vertices are 1. These include vertices 23, 42, 52, 94, etc.

A screen shot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated

In graph 2, vertex 0 to vertex 12 has a distance of 7. This is the largest. The distance from vertex 0 to several vertices are 1. These include vertices 23, 49, 75, 97, etc.

A screen shot of a computer

Description automatically generated A screen shot of a black screen

Description automatically generated

3. PySpark breaks up execution into stages to achieve optimal parallelism. Here, the task is broken up into 2 jobs that each have 2 stages. Therefore, a total of 4 stages are present. In the first job handles all tasks pertaining to file 1 while job 2 handles everything pertaining to file 2. This ensures parallel processing of file 1 and 2. The first stage in either job handles the reading of the respective file and creation of RDDs. The second task handles shuffling, partitioning and collection when we are trying to convert the RDDs to graph representations using mapping and lambda functions. The 2 jobs run the algorithm on the two files seperately and merge the outputs into one file. A diagram of a diagram

Description automatically generatedA diagram of a diagram

Description automatically generated

**Part – 3**

I have implemented page rank algorithm and limited it to 10 iterations for ease of execution. Ideally, PageRank algorithms run on huge graphs in 1000s of iterations until it converges.

A screenshot of a computer

Description automatically generated2. Node 9 has the highest PageRank. Node 73 has the lowest PageRank.

A screenshot of a computer screen

Description automatically generated

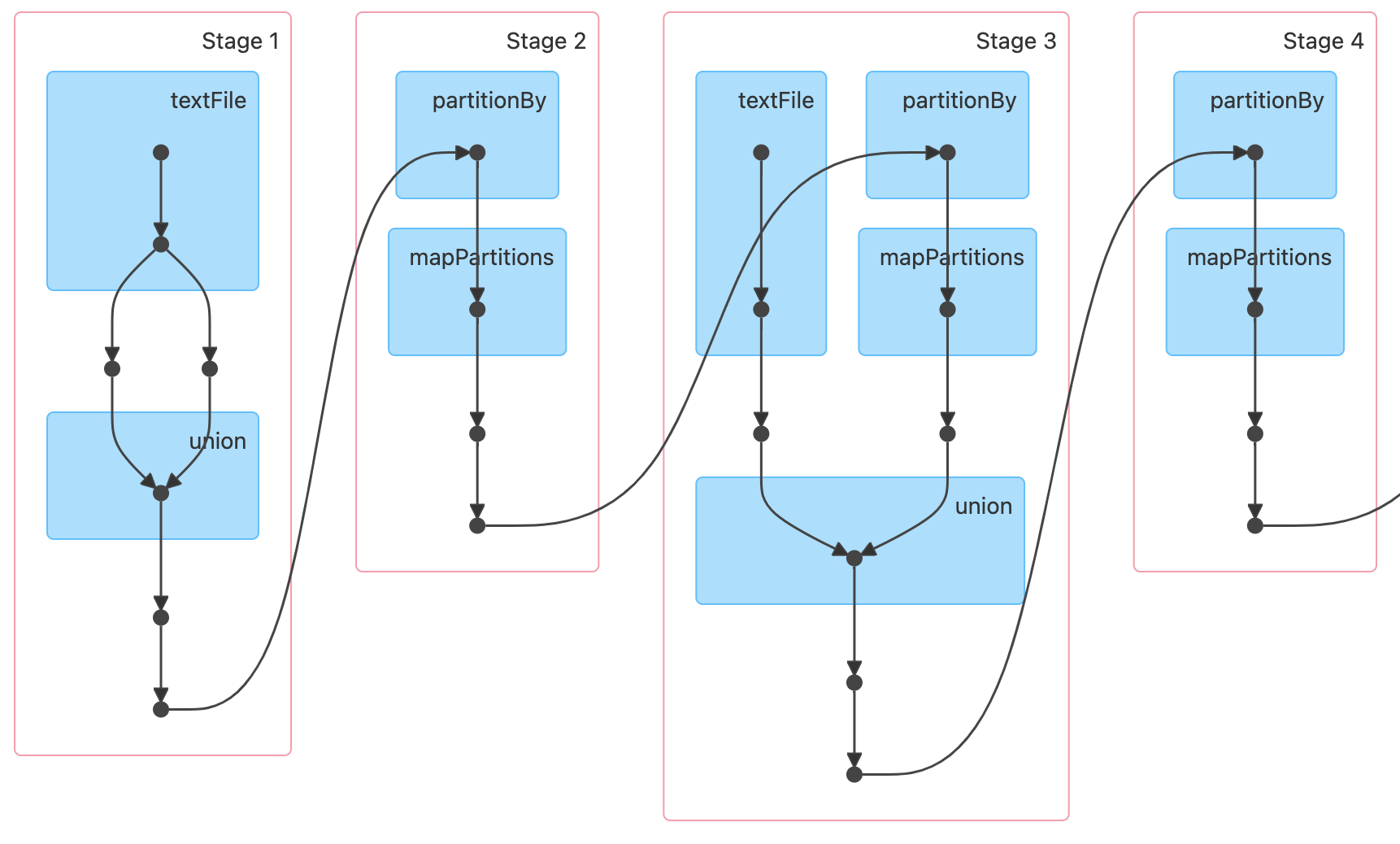
3. Spark divides the task into 4 jobs for sake of parallelization.

* Job 0 - It consists of only one stage. The input text file is read here.

A screen shot of a phone

Description automatically generated

* Job 1 – This job consists of 21 stages. Here, the text file is converted to RDDs. Partitioning, mapping and reduction occurs here as PageRank is calculated iteratively. This is the most time-consuming step as we ran 10 iterations.



A diagram of a diagram

Description automatically generated

* Job 2 – It consists of 21 stages. However, 20 stages are skipped as they are executed in previous job and values are retrieved from cache. The 42nd stage performs the sortBy() function.

A diagram of a diagram

Description automatically generated

* Job 3 - It consists of 22 stages. However, 20 stages are skipped as they are executed in previous job and values are retrieved from cache. Stage 63 and 64 basically form RDDs from the partitions after shuffling and prepare data for final output.

A diagram of a diagram

Description automatically generated