Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

ASSIGNMENT – 3 REPORT

Design Discussion:

1. Pre-processing pseudo code:
   a. Data structure used for maintaining Page details:
      Page ← Text pageName, DoubleWritable pageRank, IntWritable noOfOutlinks, List<String> outlinks

```
map (offset o, recordLine l) {
    //The key o is the offset of the line in the input
    //l is a line of record entries from the input file
    splitLine[] ← l.split(":") //array of tokens from line delimited on :
    pageName ← splitLine[0]
    htmlContent ← splitLine[1]
    //Changes made to parser provided in HW to replace all occurrences of "&" with "&amp;"
    //to handle errors arising due to this
    htmlContent ← html.replace("&", "&amp;")
    //parses html content and provides results as a list of pageNames
    listOfOutlinkPageNames ← parse(htmlContent)
    //Remove duplicates and self links from listOfOutlinkPageNames
    //Create new page with the output of the parser
    Page page ← new Page(pageName, listOfOutlinkPageNames)
    emit(page, page)
    //For every outlink of page emit a Page structure containing only the pageName of the
    //outlink as a value. This is done to account for dangling nodes
    for every outlink in page
        outlinkPage ← new Page(outlink);
        emit(outlinkPage, outlinkPage)
}

//Combiner is defined to combine the results of the mapper so as to provide a merged input
//to the reducer
combiner(Page p, List<Page> pages) {
    for each page in pages
        p.outlinks.addAll(page.getOutlinks)
        p.noOfOutlinks ← p.noOfOutlinks + page.noOfOutlinks
    emit(p,p)
}
```

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

Note: Secondary sort is applied to the results of Mapper+Combiner to sort the results based on the noOfOutlinks. We know that the mapper emits a) a page with all the details of the page as produced by the parser and b) for every outlink of the page - a page with only the page name corresponding to the outlink page name. So, in the reducer we can intuitively predict that the incoming list of values corresponding to a key would have one page with all values corresponding to the page and other pages with only the page name. Since the pages are sorted based on the noOfOutlinks we will have the only relevant page that we need to emit as the first available value in the reducer. Therefore the reducer only has to take the first value and emit it

```
reduce (Page p, List<Page> pages) {
      for each page in pages
            emit(page.toString, null)
            break;
}
```

Key Comparator:
This class extends the WritableComparator class and we override the compare method to sort the pages in descending order of noOfOutlinks.

compare (key1, key2)

- Compare pageName of both keys, if both are equal, compare noOfOutlinks of both keys and return the result of comparison
- If pageName of keys are not equal return result of comparison of noOfOutlinks

Group Comparator:
This class extends the WritableComparator class and we override the compare method to group the pages by pageNames.

compare (key1, key2)

- Compare the two keys based on the pageNames and return the result of comparison to provide the grouping factor.

HashPartitioner:

This class extends the Partitioner class and we override the getPartition class to ensure that pages with the same pageName are sent to the same reducer irrespective of the noOfOutlinks which is part of the composite key. The getPartitioner method is the default one used by Hadoop.

2. Page-Rank pseudo code:
   a. Data structure used for maintaining Page details:

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

Page ← Text pageName, DoubleWritable pageRank, IntWritable noOfOutlinks,
List<String> outlinks
EmitMultiValueWritable ← Class[] ← Page.class, DoubleWritable.class

b. enum DanglingCounter ← DANGLING_NODE_MASS (Long)

```
map (offset o, recordLine l) {
    //The key o is the offset of the line in the input
    //l is a line of record entries from the input file
    splitLine[] ← l.split("~") //array of tokens from line delimited on ~
    pageName ← splitLine[0]
    pageRank ← splitLine[1]
    noOfOutlinks ← splitLine[2]
    outlinks ← splitLine[3]

    //The algorithm after this point is provided in Module 6, in section 3. Lesson 1 > PageRank
    //sub-section 3.3

}

reduce (Text pageName, List< EmitMultiValueWritable > pageOrRanks) {
    //The algorithm followed at reducer is provided in Module 6, in section 3. Lesson 1 >
    // PageRank, sub-section 3.3 PageRank in MapReduce
}
```

Note: There is use of counters in the PageRank evaluation to obtain the total number of pages in the dataset and to factor in the contribution of dangling nodes towards page rank of all the pages in the dataset. The dangling factor is used in the formula for evaluating pageRank as provided in Module 6, in section 3. Lesson 1 > PageRank, sub-section 3.5 Dangling Nodes and is evaluated as provided in Solution 2 in Module 6, in section 3. Lesson 1 > PageRank, sub-section 3.6 Computing δ.

3. Top-K pseudo code:
   a. Data structure used for maintaining Page details:
      Page ← Text pageName, DoubleWritable pageRank, IntWritable noOfOutlinks,
      List<String> outlinks
   b. Data structure used for combining intermediate results:
      Map<Page, Double>

```
Class Mapper {
        HashMap<Page, Double>
        setup () {
```

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

```
                initialize the HashMap structure H
        }
        map (offset o, recordLine l) {
                //The key o is the offset of the line in the input
                //l is a line of record entries from the input file
                splitLine[] ← l.split("~") //array of tokens from line delimited on ~
                pageName ← splitLine[0]
                pageRank ← splitLine[1]
                noOfOutlinks ← splitLine[2]
                outlinks ← splitLine[3]
                page ← new Page(pageName, pageRank, noOfOutlinks, outlinks)

                Map.put(page, pageRank)

        }
        cleanup () {
                //Sort the map based on pageRank values
                for each entry in HashMap
                        emit(H .pageRank, H.page)
        }
}


Class Reducer {
        List<Page>
        setup() {
                initialize the list structure L
        }
        reduce(DoubleWritable rank, List<Pages> pages) {
                for each page in pages
                        if list.size() < 100
                                L.add(page)
        }
        cleanup() {
                for each page in pages
                        emit(page.toString(), null)
        }

}
```

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

Note: There is no partitioner defined for this phase as the number of reduce tasks is set to 1 and therefore all records end up in the same reducer in the reduce phase.

Data Transfer in each Iteration:

Machines	Iteration	Data Transferred from Mappers to Reducers	Data Transferred from Reducers to HDFS
6 machines (1 Master, 5 Workers)	1	1278575752	1182730068
	2	1505729011	1185338738
	3	1279751782	1187132572
	4	1509667553	1185063702
	5	1510928170	1185038789
	6	1512301824	1185042740
	7	1513673341	1185036394
	8	1514937303	1185033283
	9	1516036645	1185019195
	10	1517011283	1185029231
11 machines (1 Master, 10 Workers)	1	1289167174	1182729615
	2	1518669149	1182666870
	3	1518702405	1182665707
	4	1518695148	1182653979
	5	1518788836	1182650454
	6	1518852689	1182651592
	7	1518905224	1182646356
	8	1518924109	1182647686
	9	1518936705	1182640147
	10	1519005706	1182646242

As seen in the table given above, the amount of data transferred between Mappers and Reducers and between Reducers and HDFS remains almost similar for the 10 iterations. There are more bytes transferred in iterations following the first iteration. This could be due to the dangling factor coming into picture in the second iteration for page rank evaluation with user defined counters values being accessed and updated. The bytes transferred more or less remain the same as we read similar data over the 10 iterations with the number of pages in the dataset being the same.

Performance Comparison:

Following table contains the running times for pre-processing, page rank evaluations over 10 iterations and top-k evaluation:

Class Number: CS6240 Parallel Data Processing in Map Reduce
HW Number: 3
Name: Neha Pradhan

Time Factor	Running Time on 6 machines	Running Time on 11 machines
Pre-processing	23 mins 33 seconds	12 mins 53 seconds
PageRank	19 mins 43 seconds	13 mins 49 seconds
Top-K	47 seconds	39 seconds

Questions:

1. Which of the computation phases showed a good speedup?

- The pre-processing phase shows the best speed up as the number of machines increase as seen in the above table. The speed up factor here shows a lot of reduction in time for the same amount of work as the number of machines increase.

2. If a phase seems to show fairly poor speedup, briefly discuss possible reasons—make sure you provide concrete evidence, e.g., numbers from the log file or analytical arguments based on the algorithm's properties.

- Based on the findings of the above table the top-k evaluation shows poor speed up. This could be due to the fact that the number of reduce tasks are set as 1 in order to evaluate the top-k ranks. So, even with many machines at our disposal the entire burden of the reducing task lies only on one machine.
- After checking the logs, we can see that the number of records to be processed by the reducer is almost the same in both 6 machines and 11 machines (2200 and 1900 respectively) and since we have only one reducer working on these inputs, the time taken for the evaluation would also be almost the same. There is no scope for speedup in the reducer end due to allocating only one reduce task in the reduce phase.

3. Report the top-100 Wikipedia pages with the highest PageRanks, along with their rank values and sorted from highest to lowest, for both the simple and full datasets. Do they seem reasonable based on your intuition about important information on Wikipedia?

- The top-100 pages sorted based on page ranks are provided in the Outputs folder for different machines as deliverables.
- The top-100 links contains links like Wiktionary, Wikimedia_Commons_7b57 pertaining to some site that is relevant to Wikipedia. This could mean that these sites are present as outlinks for many pages on the data set which based on other pages in the top-100 are not as relevant. It could be that these links are present on the important pages due to which the its page rank accumulation is more due to higher ranked pages pointing to it.