CS6240 Map Reduce HW5: Akash Kadam (Section 2)

Pseudo Code
Row by Column approach

Main Idea
1) Pre Processing of the data
2) Create a matrix representation of data
3) Compute the page rank for 10 iterations
4) Find the top 100 pages

Job1 – Preprocessing (Map and Reduce)

```
InputParseMapper{
        map(key, value){
        It reads the data from the bz2 files. It also maintains a counter for totalNodes and
danglingNode
emit(pageName, listOfOutlinks)
        }
}

InputParseReducer{
        reduce(key, Iterable<Text> values){
        It performs the mapping of a page Name to a unique long value for each page which is
used as an index for the matrix representation
        }
}
```

Job 2 – Creation of Matrix (Map and Reduce)

```
PRI1Mappper{
        map(key, value){
        initialPr = 1 / totalNodes
        if the values are not starting with string "mapp" do the following
                split the value on tab and store in olinks array
                page = olinks[0]
                if length of olinks[1] > 1
                        outlinks [] = split olinks[1] on ','

                for(all l in outlinks )
                {
                        Create the matrix representation mm
                        emit(l, mm)
                        where we set mm(l, false, page+contri, 'M')
                }

                emit current page with all it outlinks
                create a new matrix mm(page, false, "", 'M')
```

```
            emit(page, mm)
        else the pages we get are the dangling ones{
        mm = new MMatrix(page, true, “”, ‘M’);
        write(page, mm)
        }
        }
}

PRI1Reducer{
        reduce(Text k, Iterable<MMatrix> values){
            for val in values
                if val does not have empty inlinks{
                    inlinks = inlink + , + val.getInlinks
                }
            mm = new MMatrix(key, isDangling, inlinks, ‘M’)
            emit(key, mm)
            update the dangling count
        }
}
```

Job 3 – 1st Iteration of Page Rank(Map only job)

```
Map(key, MMatrix value){
        initPR = 1/ total_nodes
        emit(key, Text(initPR))
}
```

Job4 – running for rest Iteration (Map and Reduce Job)

```
ComputePRMapper{

        Map(Text key, MMatrix val){
            If type of key is ‘M’
            emit(key, value)
        }
}

ComputePRReduce{

        A hashmap <long, double> to store in links contrib

        reduce(Text k, Iterable<MMatrix> val){
            for val in values{
                if val type is ‘M’
                    inlinkscontri[] = val.getInlinks
                for (contri in inlinkscontri)
```

```
                        if contri is not empty
                        put in hashMap (page, contribution value)

                for pageContri in mm.getInlinks splitted on ‘,’{
                        if pageContri is not empty string
                        {       page = pageContri[0]
                                If(mapping.containsKey(page){
                                        PrSum += mapping.get(page) * inlinks.get(page)
                                }
                        }
                }
                RandomJumpComputation = prsum * alpha
                finalPR = computedDF + randomHopFactor + randomJumpComputation


                }

        If input is a dangling node then increment the dangling factor
        emit(key, new Text(finalPR))
        }
}
```

Job 5 – get the top 100 page rank with their values (Map and Reduce)

```
Top100MapperJob{

        TreeMap<Double, Long> is used to store the sorted Pages
        map(key, value){
                sortedPages.put(val, key)
        }

        In clean up
        For(pr in sortedPage.keySet()){
        emit(DoubleWritable(pr), new Text(sortedPage.get(pr))
        }
}

Top100ReduceJob{
        HashMap<Long, String> pageLink stores the pagename and the number mapping
        reduce(DoubleWritable key, Iterable<Text> values){
        for val in values
                i = 0 until 100
                        emit(new Text(page), pageRank)

        }
}
```

How each matrix and vector is stored?

I have used a sparse representation where only rows with non-zero elements are stores and for those rows the columns with non -zero values are explicitly enumerated as (column, value) pair. For examp [0 : (0,1), (2,2); 2: (0,3), (1,4)]. Remaining elements are implicitly known to have value zero

Dangling Nodes

I am handling the dangling nodes before I create the matrix representation. Since I am doing this I do not have to worry about replacing the zero-columns in M and then work with the corresponding M'. I set a boolean isDangling to true if it a dangling node.

PERFORMANCE COMPARISION

For Type A (row by)

Machine Count	Step 1 and 2 (min)	Step 3 (min)	Step 4 (min)	Total(min)
6	**48**	**32**	**2**	**82**
11	**29**	**20**	**2**	**51**

Times from previous Hadoop Assignment

Machine Count	Step 1 and 2 (min)	Step 3 (min)	Step 4 (min)	Total(min)
6	**18**	**29**	**3**	**50**
11	**10**	**48**	**2**	**60**

The Matrix approach which we used should be more efficient as compared to previous hw3 approach. The reason I am getting a performance lag is because I haven't implemented intelligent partitioning and I am using a single reducer only to compute the page ranks. For this approach I could have used the intelligent partitioning by sending a specific set of rows of the matrix and the vector to the reducer to compute the page rank. Also, it would be necessary to consider the actual values inside the matrix because for dangling nodes the values would have been zero so the load balancing among the reducer would not be proper if we just partition the data based on matrix index only.

The result of this execution as compared to the previous hw3 are very similar there is a slight change in the page rank values though and that is because of the precision we take care of while computing the page rank.