CS6240, Assignment 3, Rakesh Buchan Krishna Murthy

Pre-Processor job

```
map(key k, input_data value){
    parse() ;// added extra functionality to handle ~ and changed linkPageNames list to hold Text data
    emit(pageName, outLinksList);
    for each outLink in outLinkList
        emit(outLink, null);
}

combine(pageName, outLinkList){
    If (outLinkList == “”)
        emit(pageName, null);
    else
        for each outLink in outLinkList
            finalOutLinkList = finalOutLinkList + outLink; //concat all outLinks
        emit(pageName, finalOutLinkList);
}

reduce(pageName, outLinkList){
    If (outLinkList == “”)
        emit(pageName, “[] #0.0”); // emitting a value saying no outLinks and a default rank of 0.0
    else
        for each outLink in outLinkList
            finalOutLinkList = finalOutLinkList + outLink; //concat all outLinks
        emit(pageName, finalOutLinkList+ “ #0.0”); // emitting a value with all outLinks and a default rank of 0.0
}
```

Page Rank Job

(Logic is very similar to slide 3.3 PageRank in MapReduce from Graph Algorithms deck)

```
totalPages = REDUCE_OUTPUT_RECORDS; // from PreProcessorJob
current_delta = 0.0;
previous_delta = 0.0;
for each iteration
    Class Mapper{
        setup(){
            get the counter values which are set in driver class – currentIteration, totalPages
        }
        map(pageName, (outLinkList # pageRank)){

            String[] itr ← split value based on “\n”
            Loop over itr[]
            String[] pageDetails ←   split itr based on “#”
            pageName = pageDetails[0].substring(0, indexof(‘[‘));
            pageRank = pageDetails[1];
            outLinks = pageDetails[0]. substring(indexof(‘[‘), indexof(‘]‘));

            if(iteration == 1)
                currentPageRank = 1/ totalPages;
            else
                currentPageRank = pageRank;

            emit(pageName, new PageDetails(pageName, currentPageRank,
        outLinks));

            if(no outLinks present)// it’s dangling node
                emit(dummy, currentPageRank);
            else
                for each outLink in outLinks
                    emit(outLink, currentPageRank/# outLinks);
    }
}
Class Reduce{
    setup(){
        get the counter values which are set in driver class – totalPages, previous_delta,
        current_delta
    }
    reduce(pageName pName, PageDetails/DoubleWritable pObj){
        if(key == dummy)
            for all pObj
                current_delta = current_delta + pObj;
            set CURRENT_DELTA = current_delta; // CURRENT_DELTA is a counter
        else
            for all pObj
                if (pObj == DoubleWritable)
                    newRank = newRank + pObj;
                else
                    newPage = pObj;
            rank = 0.15/totalPages + (1-0.15)(previous_delta/ totalPages +
            newRank);
            newPage.setPageRank(rank);
            emit(pName, pObj.getOutLinks() # rank);
    }
}
previous_delta = current_delta;
current_delta = 0.0;
```

Top K Job

```
    Class Mapper{
        setup(){
            get the counter values which are set in driver class – top_K_count
            initialize Hashmap pageRankMap to store pageName vs pageRanks
        }
        map(pageName, (outLinkList # pageRank)){
            String[] itr ←  split value based on "\n"
            Loop over itr[]
                String[] pageDetails ←  split itr based on "#"
                pageName = pageDetails[0].substring(0, indexof('['));
                pageRank = pageDetails[1];
                pageRankMap.put(pageName, pageRank );
        }
        cleanup(){
            pageRankMap.sort // in the decreasing order of pageRank
            emit(pageRank, Page);
        }
    }
     Reduce(pageRank, Page){
        Loop over Page
            List<Result> = page P // only copy first top_k
        For all values of Result
            emit(pageName, pageRank)
    }


Partitioner(){
    Directs all records to a single reducer by returning 0;
}

keyComparator(){
    sort the keys based on pageRanks
}
```

CS6240, Assignment 3, Rakesh Buchan Krishna Murthy

Data Transfers

With 5 Workers

Iteration	Mapper to reducers	Reducers to HDFS
1	1436864442	1478368186
2	1654464643	1480934555
3	1652593742	1479589653
4	1653975499	1480971822
5	1652840603	1481089469
6	1653533791	1480969397
7	1653005332	1481045662
8	1653244393	1480991380
9	1653072235	1479060492
10	1653074275	1480986990

With 10 Workers

Iteration	Mapper to reducers	Reducers to HDFS
1	1474094428	1478368308
2	1698745619	1480935270
3	1696963603	1479594933
4	1698192696	1480972540
5	1697233606	1481098808
6	1697950656	1480979879
7	1697432640	1480071949
8	1697456836	1480993117
9	1697513887	1481016221
10	1697557406	1480996537

In both cases, as we see, the data transfer between the mappers and reducers as well as reducers and HDFS remain almost same for each iteration, as we are reading the whole data from each file and transferring and then writing them to HDFS in each iteration.

With 5 workers

pre-processing time (mm:ss)	21:33
ten iterations of PageRank (mm:ss)	30:11
top-100 pages (mm:ss)	01:17

With 10 workers

pre-processing time (mm:ss)	12:47
ten iterations of PageRank (mm:ss)	19:22
top-100 pages (mm:ss)	00:54

These timings makes sense since as the number of worker machines increase, the time taken would reduce which can be inferred from above stats

CS6240, Assignment 3, Rakesh Buchan Krishna Murthy

Time taken for page rank is more than pre processing and top-100 in both cases because it deals with transferring large amount of data (shuffle bytes from log files) for each iteration, hence transfer time would add significant time to the overall time

Top-100 in both cases is taking roughly same time as we have dedicated just a single machine to compute it in both cases.

Speed up is high in pre-processing, as with the number of machines increased for the same work, pre-processing required lesser time compared to the other two, and top_k has the least speed up, since the time taken is almost same as the reducer phase uses a single machine.

The following wiki pages are present in top 100 - Wikimedia_Commons_7b57, Wiktionary, since these pages have a lot of links referring to each other resulting in page rank shooting up, this is one of the techniques used by bloggers to increase the page rank of their websites by adding links to their pages from different pages, So it’s better to remove duplicate out links to pages while measuring page ranks.