Database System Implementation Assignment 3
(Shailesh Goel - sg1180) – (s3 bucket: mr-db-shailesh)

1. The name of the s3 bucket containing output is s3://mr-db-shailesh, and the folder name is s3://mr-db-shailesh/output. A Readme file has been placed in the bucket to help navigate the different folders in the bucket.
2. The Output of the Word Count Example running the word count java file provided on Sakai is stored in the bucket. It has been run of the Large Dataset as well as the Small dataset provided on Sakai. The example has been run on clusters of size 2, 4, 8, 10, 12 core and 1 Master Node. The size of data being relatively small, doesn't provide any good improvement over the running times as the cluster size increases and the job time hovers around 46-48 second.
    a. JAR file can be found in s3://mr-db-shailesh/jobs/wc.jar
    b. Source code and JAR file have been also uploaded on Sakai.
    c. Input directory for small word count data set s3://mr-db-shailesh/input/WordCountData
    d. Output directory for small word count data set s3://mr-db-shailesh/output/wordcountoutput-small-2core
    e. Input directory for Large word count data set s3://worddatasets/WordCountInput
    f. Output directory for small word count data set on cluster of different sizes.
        - s3://mr-db-shailesh/output/wordcountoutput-Large-2core
        - s3://mr-db-shailesh/output/wordcountoutput-Large-4core
        - s3://mr-db-shailesh/output/wordcountoutput-Large-8core
        - s3://mr-db-shailesh/output/wordcountoutput-Large-10core
        - s3://mr-db-shailesh/output/wordcountoutput-Large-12core

    As you will observe the number of file in each of the folders above vary. The reason being the number of reducers increases as the cluster size increases. Each reducer generates an output file.
3. The three Chess Map Reduce tasks have been run on clusters of size 2, 4, 8, 10, 12 core and 1 Master Node. Increasing the cluster sizes do provide with a greater improvement of running time and performance. Detailed analysis below the cost of running a larger cluster for a jobs being done in shorter times is approximately equal to or less than jobs being run on a smaller cluster taking a long time. Below are input path and output along with other necessary resources.
    a. JAR Files can be found in s3://mr-db-shailesh/jobs
        - Part A: s3://mr-db-shailesh/jobs/cga.jar
        - Part B: s3://mr-db-shailesh/jobs/cgb.jar
        - Part C: s3://mr-db-shailesh/jobs/cgc.jar
    b. Input Directories are given below for the following Small Input Data Sets:
        - s3://mr-db-shailesh/input/chessData
    c. Output Directories are given below for the following Small Input Data Sets:
        - s3://mr-db-shailesh/output/cgA-Small-2core
        - s3://mr-db-shailesh/output/cgB-small-2core
        - s3://mr-db-shailesh/output/cgC-Small-2core
    d. Input Directories are given below for the following Large Input Data Sets:
        - s3://chessdatasets/VeryLargeChessInput

e. Output Directories for 3 jobs run on Cluster
   - **2 cores and 1 master** node
     1. Part A: s3://mr-db-shailesh/output/cgA-Large-2core
     2. Part B: s3://mr-db-shailesh/output/cgB-Large-2core
     3. Part C: s3://mr-db-shailesh/output/cgC-Large-2core
   - **4 cores and 1 master** node
     1. Part A: s3://mr-db-shailesh/output/cgA-Large-4core
     2. Part B: s3://mr-db-shailesh/output/cgB-Large-4core
     3. Part C: s3://mr-db-shailesh/output/cgC-Large-4core
   - **8 cores and 1 master** node
     1. Part A: s3://mr-db-shailesh/output/cgA-Large-8core
     2. Part B: s3://mr-db-shailesh/output/cgB-Large-8core
     3. Part C: s3://mr-db-shailesh/output/cgC-Large-8core
   - **10 cores and 1 master** node
     1. Part A: s3://mr-db-shailesh/output/cgA-Large-10core
     2. Part B: s3://mr-db-shailesh/output/cgB-Large-10core
     3. Part C: s3://mr-db-shailesh/output/cgC-Large-10core
   - **12 cores and 1 master** node
     1. Part A: s3://mr-db-shailesh/output/cgA-Large-12core
     2. Part B: s3://mr-db-shailesh/output/cgB-Large-12core
     3. Part C: s3://mr-db-shailesh/output/cgC-Large-12core

f. Output of Part A for Small and Large Data Set have been Attached separately.

g. Output of All parts can be found in folder output-hadoop attached along with this report.

Jobs/Nodes	**2C+1M**	**4C+1M**	**8C+1M**	**10C+1M**	**12C+1M**
Word Count (wc.jar)	52 seconds	48 seconds	46 seconds	46 seconds	48 seconds
	(1) 9 M, 7 R	(1) 9 M, 15 R	(1) 9 M, 31 R	(1) 9 M, 39 R	(1) 9 M, 47 R
	7 Output Files	15 Output Files	31 Output Files	39 Output Files	47 Output Files
Chess Game A (cga.jar)	17 minutes	9 minutes	4 minutes	4 minutes	3 minutes
	(1) 339 M, 7 R (2) 7 M, 1 R	(1) 340 M, 15 R (2) 15 M, 1 R	(1) 340 M, 31 R (2) 31 M, 1 R	(1) 339 M, 39 R (2) 39 M, 1 R	(1) 340 M, 47 R (2) 47 M, 1 R
	1 Output Files	1 Output Files	1 Output Files	1 Output Files	1 Output Files
Chess Game B (cgb.jar)	23 minutes	14 minutes	6 minutes	5 minutes	4 minutes
	(1) 339 M, 7 R	(1) 339 M, 15 R	(1) 340 M, 31 R	(1) 339 M, 39 R	(1) 340 M, 47 R
	7 Output Files	15 Output Files	31 Output Files	39 Output Files	47 Output Files
Chess Game C (cgc.jar)	17 minutes	9 minutes	5 minutes	4 minutes	3 minutes
	(1) 329 M, 7 R (2) 7 M, 1 R	(1) 340 M, 15 R (2) 15 M, 1 R	(1) 340 M, 31 R (2) 31 M, 1 R	(1) 339 M, 39 R (2) 39 M, 1 R	(1) 339 M, 47 R (2) 48 M, 1 R
	1 Output Files	1 Output Files	1 Output Files	1 Output Files	1 Output Files
Total Time	58 minutes	33 minutes	16 minutes	14 minutes	11 minutes
Cost/Cluster	24 nhrs/hr	40 nhrs/hr	72 nhrs/hr	88 nhrs/hr	104 nhrs/hr
Total Cost	23.2 Hours	22 Hours	19.2 Hours	20.53 Hours	19.06 Hours

Detailed Analysis of the 3 Chess Game Tasks

Table above gives the comparisons of time taken to run each job on cluster size 2, 4, 8, 10 and 12 core nodes and a single master node. The table below also gives the cost per cluster for running it for one hour and the expected cost of running one set of 4 jobs on one of the cluster (nhrs). Using this we calculate the expected cost in hours. The table also gives the

number of mappers (M) and reducers (R) used in each job according to the analysis of the log files. (1) and (2) labels signify the 2 map reduce phases of a job.
Jobs Chess Game Task A and Chess Game Task C are using 2 Map Reduce Phases chained together to get the output.

Chess Game Task A is a straight forward by getting the result of each game. In the first phase we collect the total number of games won by black & white and the number of the game that drew. The reducer also acts as a combiner in this case. The intermediate output is stored in Temporary Memory. The Second phase is just to compute the percentages and has. Although, the number of mappers provisioned in the second phase is about 15, only 1 mapper and reducer is used, rest of the mappers are idle.

Chess Game Task B uses only one phase. Chess Game Task B uses a Custom Input Format, which allows it to fetch the complete game record and then parse it using ChessPresso PGN Parsing Library. This custom input format also overrides the default file splitter, avoiding corrupt data and also provides for the logic to read the split data. We use FileInputFormat class and override its getRecordReader Method for the above-specified purpose. [1] A combiner has also been implemented to reduce the computation during the reduce phase. The data is not sorted across all files according to the key, i.e the name of the player as it was not given requirement and doesn't affect the correctness of the output. To implement sorting across all the files of the output we will have to implement TotalSortOrdering using a custom partitioner and comparator similar to what has been done Chess Game Task C.

Chess Game Task C uses a Custom Partitioner, Group Comparator and Record Comparator to sort the values according to decreasing frequency of the number of moves made in a game. As in the case Chess Game Task A, we use the first Map Reduce Phase to collect the Total count for each of the number of moves played for Chess Games in the input. The reducer again acts a combiner in this phase. The second phase is used to calculate the percentage and sort the number of moves in the order of their decreasing frequency i.e sorting based on value. This is also known as secondary sorting. Partitioner ensure only the natural key is considered when determining which reducer to send the data to, we need to write a custom partitioner. Once the data reaches a reducer, all data is grouped by key. Since we have a composite key, we need to make sure records are grouped solely by natural key, which is accomplished by writing a custom GroupPartitioner. Finally Comparator implements the logic the order in which the individual records are to be sorted, that is by the secondary key. [2]

"The number of maps is usually driven by the number of DFS blocks in the input files."[3]
From this statement we can infer that the number of initial maps task is approximately the same that is 340. This helps us infer that as the cluster size increases the average number of map tasks for each node decrease i.e. **parallelization increase**. In here the approximate size of input data is approximately 22 GB, and the block size is 64 MB, hence the number of mappers is about 340, which are reasonable enough.
Another to thing to remember as the size of the input data increase we can increase the input split block size to reduce the number of mappers.

BIBLOGRAPHY

[1] File Input Format Class – Input Splitter and Reducer
http://hadoop.apache.org/docs/r2.6.1/api/org/apache/hadoop/mapreduce/lib/input/FileInputFormat.html
[2] Reducer –Secondary Sort
https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/mapreduce/Reducer.html
[3] How many Map and Reduce Tasks created?
http://wiki.apache.org/hadoop/HowManyMapsAndReduces