NAME : RAKESH BAINGOLKAR
UBID : 50097576

PROGRAMMING ASSIGNMENT 4

PART 1:
SIMPLE BUCKET SORTING USING MAPREDUCE

/SORT/
FILES :
RANDOM NUMBER GENERATOR : random.jar
COMMAND TO RUN : java -jar random.jar -t [PROBLEM_SIZE] [FILE_NAME]
EXAMPLE : java -jar random.jar -t 10000 README.txt
Above command will write to README.TXT 10000 random generated values.

SORTING FILE : sort.jar
COMMAND TO RUN : java -jar sort.jar [INPUT FILE] [OUTPUT_PATH]
EXAMPLE : java -jar sort.jar README.txt output
Above command will write the sorted numbers on STDOUT with the TIME TAKEN and also in the file output/part-r-00000.

SLURM FILE : SLURM_myHadoop.sh
Line 32 : I run the random.jar file i.e.
java -jar random.jar -t 1000 README.txt
(1000 → Problem Size)

RATIONALE FOR PARALLELISM :

The README.TXT file is of the format:
[NUMBER] [PROBLEM_SIZE]

MAP STEP :

In the map step I read the README.TXT file and extract the number part. According to the number I select the proper bucket that number should be put into. Key is the bucket number and the value is the number itself.

REDUCE STEP :

In the reduce step I create an ArrayList and add all the values belonging to a particular key i.e. bucket. Then I sort the arraylist i.e. a single bucket. This step executes in parallel. The key in MapReduce is already sorted so the order of buckets are already sorted.

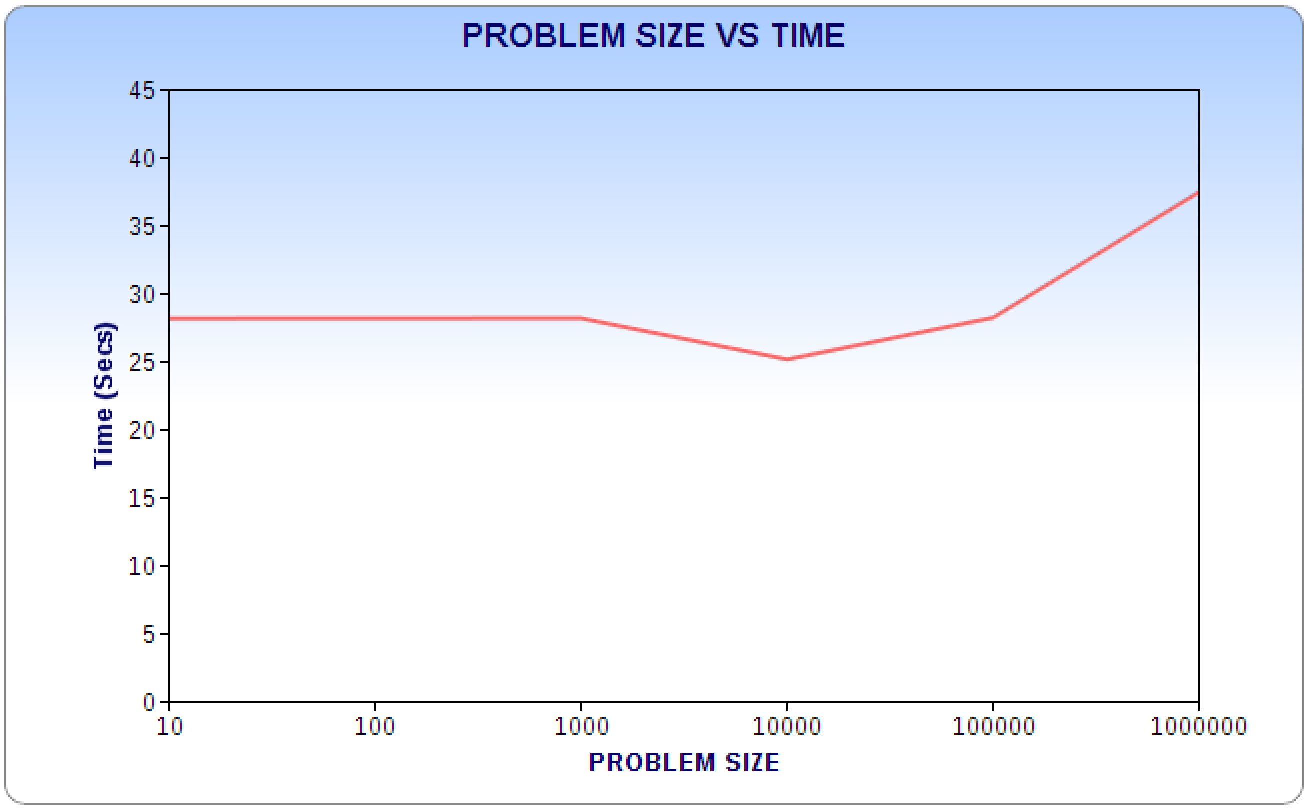
For measuring time I used the function:

```
System.currentTimeMillis();
```

For displaying the sorted values I accessed the output file in HDFS.

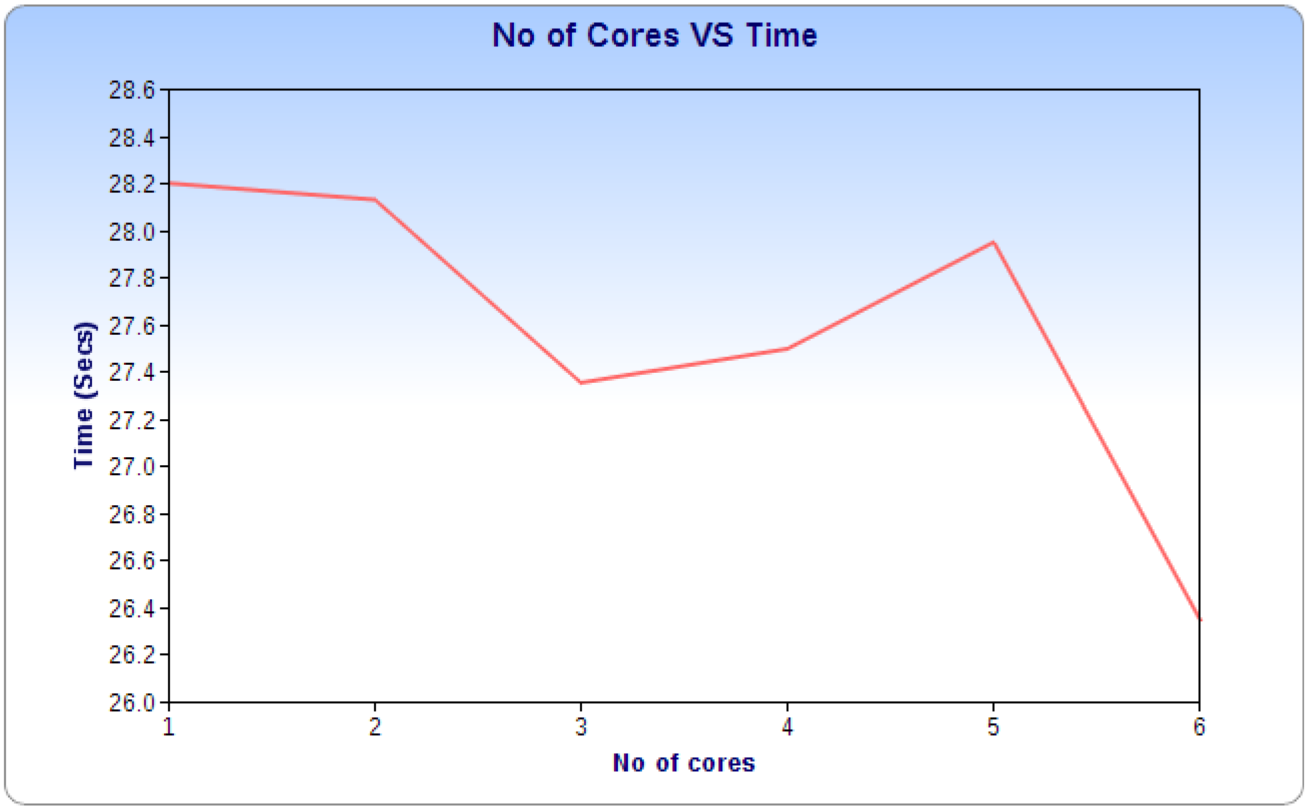
RESULTS:

INCREASING PROBLEM SIZE:



With the increase in the problem size we see that the time taken by the program to run more or less remains constant for the initial stages and then increases as the problem size increases. This is because the mapping phase will not take much time as each line in the input file is mapped at once and the reduce phase will make a difference in time such that it has to sort a large amount of numbers for large input. Also there can be a case when for the lower input there is overhead of iniiating the map reduce job which eventually results in a constant time for the large problem sizes. In either of the cases MapReduce is very powerful and it will work very fast in a real life scenario of handling large data sets because the time doesnt increase exponentially.

INCREASING NUMBER OF CORES:



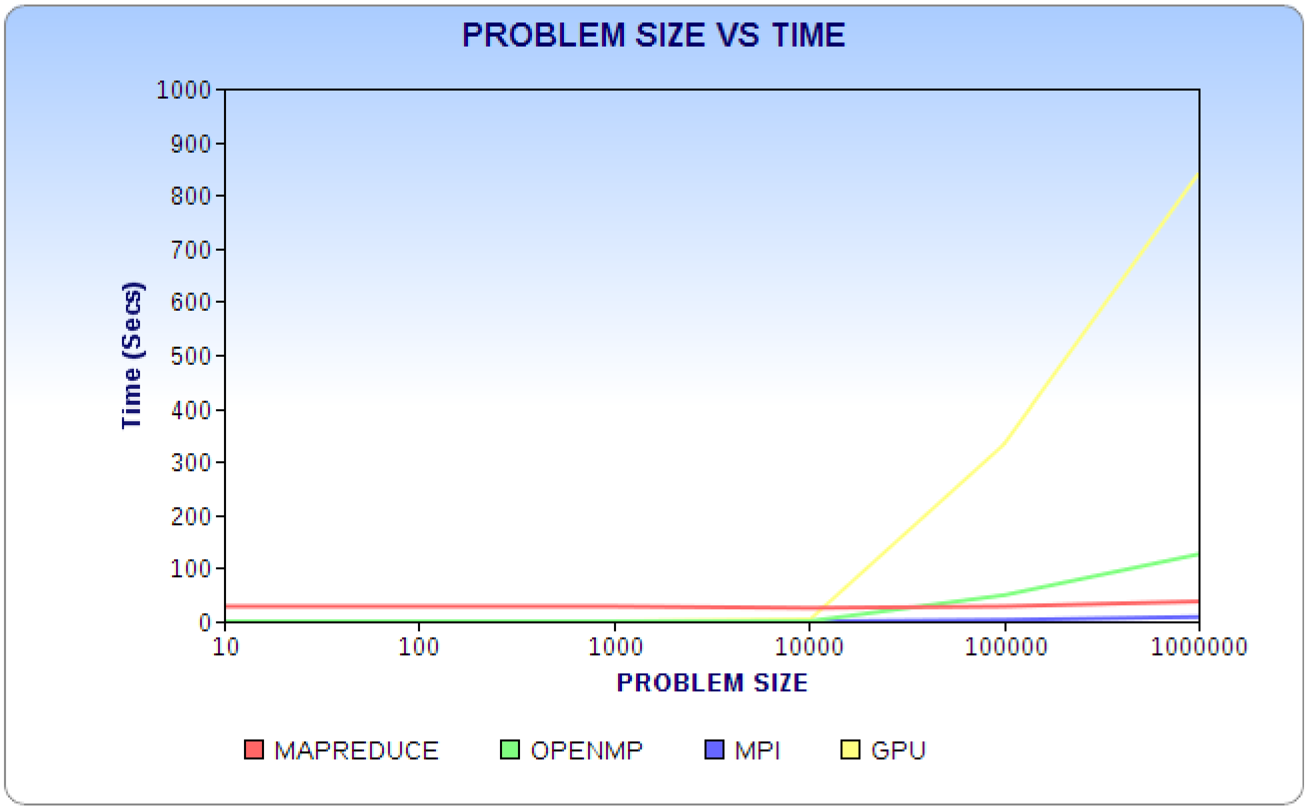
When we increase the number of cores we see that the time decreases by almost 2 seconds from core 1 to core 6. This is because different nodes run parallely on the MapReduce thus reducing the total time taken by the entire program to run. Multiple nodes consider separate line and accordingly put them in the buckets parallely while at the reduce step we see that according to the key the numbers are sorted parallely and then writen to a output file in a sorted order. Increasing number of nodes meaning less work for all nodes and thus fast execution.

COMPARISON:

SIZE	**OPENMP**	**MPI**	**GPU**	**MAPREDUCE**
10	0.000014	0.000113	0.062681	28.127
100	0.000072	0.00014	0.065972	28.135
1000	0.007803	0.000324	0.103853	28.143
10000	0.500747	0.026498	3.394545	25.118
100000	49.426869	2.645066	333.67084	28.188
1000000	126.421532	8.412576	842.42151	37.424

NO_OF_CORES	**OPENMP**	**MPI**	**GPU**	**MAPREDUCE**
1	43.7687	2.681468	333.67084	28.198
2	10.6372	4.156325	129.48756	28.128
3	2.8053	3.412563		27.351
4	6.0465	3.512534		27.495
5	9.74284	3.456867		27.947
6	1.162111	3.624513		26.348

INCREASING PROBLEM SIZE



MPI vs MAPREDUCE

When we consider a comparison between MPI and MAPREDUCE in terms of increasing the problem size for all the considered problem sizes MPI gives better performance than MapReduce. The Initialization for the Map-reduce job has its own overhead that can be considered bad for small problem sizes. But one observation we can make is that the fracion of time by which the time increases in MPI is more than the fraction of time by which the time increases in Mapreduce .In MPI as we increase the problem size same number of nodes have to deal with the large problem size which results in increase of the time taken. In case of MAPREDUCE a single line is considered in the MAP phase which is done parallely thus the time more or less remains constant. Also in MPI the blocking operations of SEND and RECV can be blamed as the Parent thread has to wait until all the other threads send the output back and if the processing time taken by the other threads is more then it might affect the overall performance of the program.

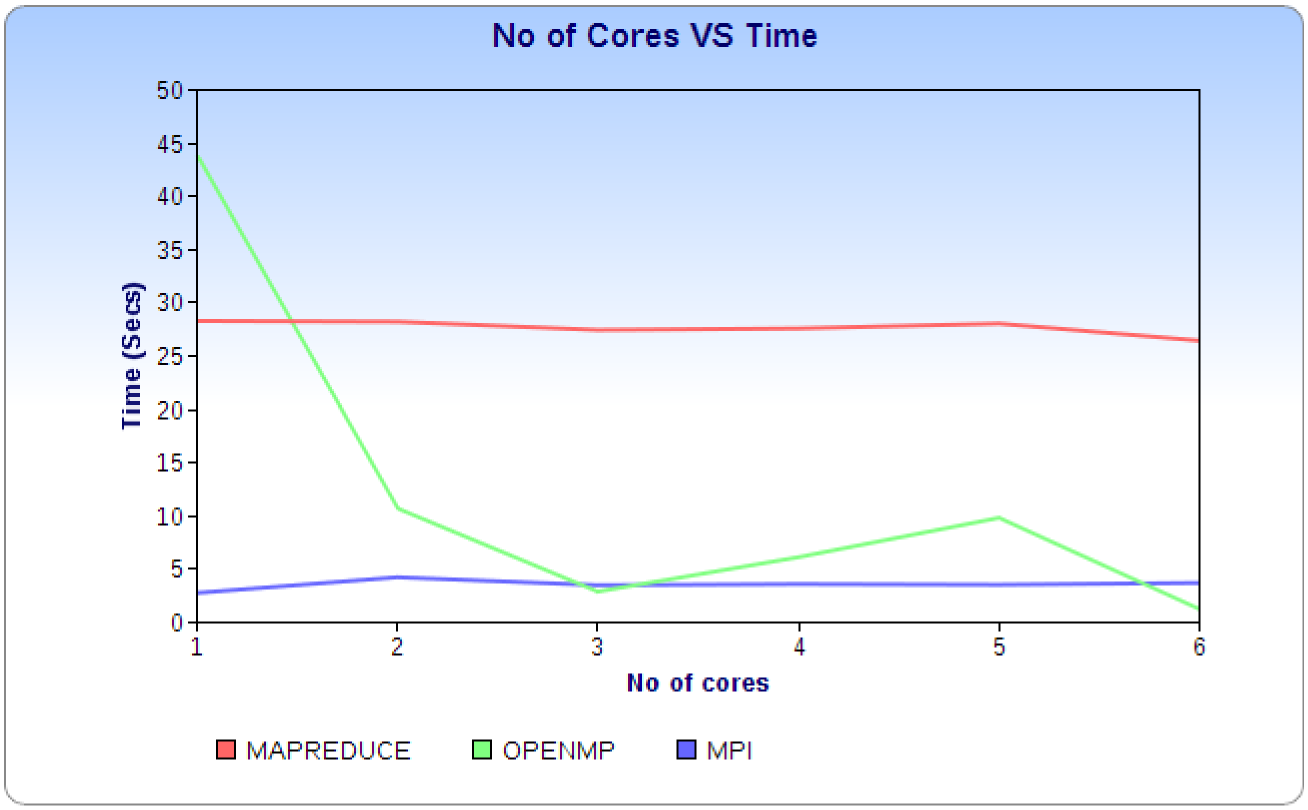
OPENMP vs MAPREDUCE

When we compare OpenMP and MAPREDUCE in terms of increase in the problem size we see that for the first four readings that is till problem size 10000 OpenMP gives good performace than MAPREDUCE but as we increase the problem size we see that he time taken by OpenMP increases by a fairly large amount whereas MAPREDUCE increases by a less facor. The reason for OpenMP to take large amount of time is that the number of threads were kept constant while the amount of work assigned per thread was increased which in turn increased the overall time taken by OpenMP. In case of MAPREDUCE the entire algorithm was parallel and apparently the MAP and REDUCE phase are faster than actual distribuion of task to threads as the increasing factor is less than OpenMP.

GPU vs MAPREDUCE

For initial values GPU gives a good performance than MapReduce but for higher values GPU takes a lot of time and the time increases by a great factor.For the larger problem sizes above we can say that GPU takes more time than MAPREDUCE. The main reason is the memory allocation of the bucket matrix. If the size of the bucket matrix is too high the GPU takes a large time to allocate the memory in the GPU. Also CUDA is limited by the amount of shared memory and the global memory while HADOOP provides a file system HDFS for storing data on the nodes which are going to compute.

INCREASING NUMBER OF CORES



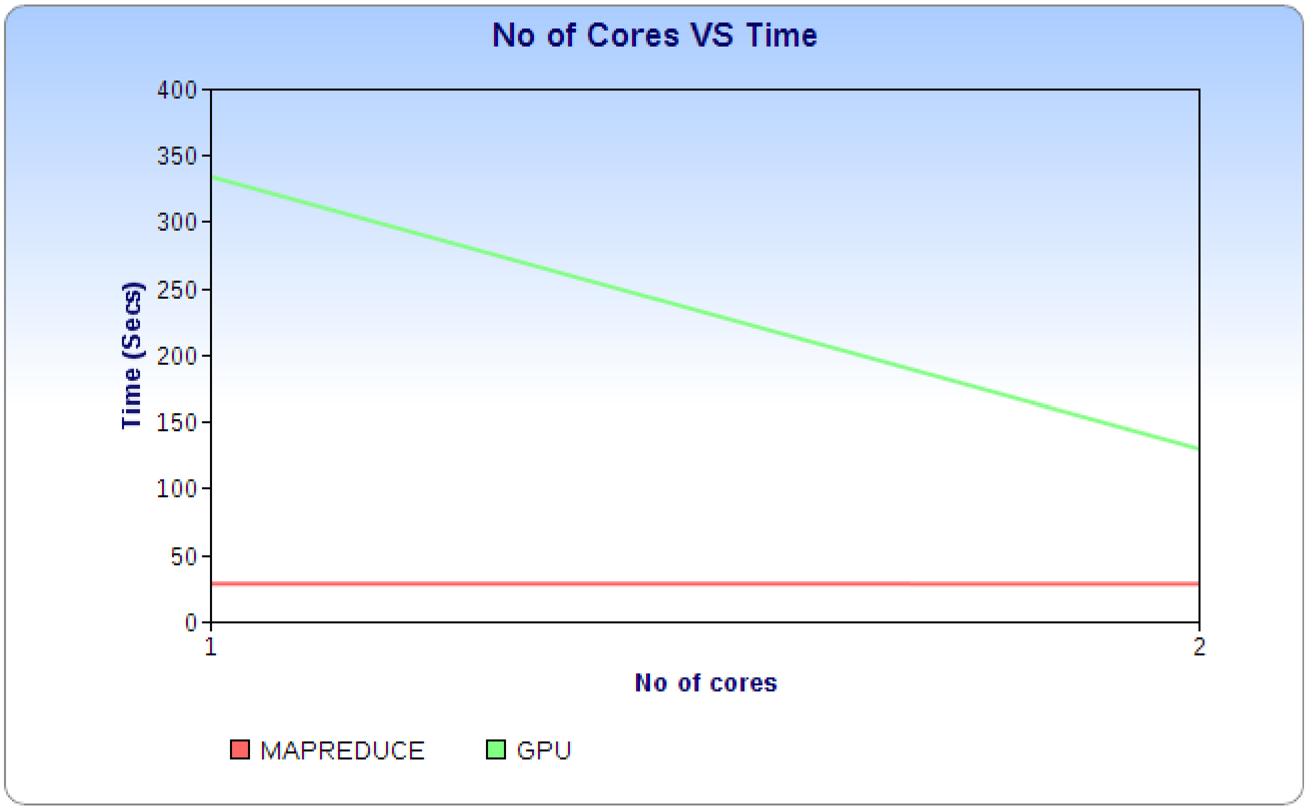
MPI vs MAPREDUCE

Above we see that for the same input as we increase the no of cores in both MPI and MAPREDUCE we see that MPI takes much less time than MAPREDUCE for same number of cores for same size of input. This indicates MPI's performance is much better than the MAPREDUCE. One reason for this is that to start a map reduce job on a particular node has its overhead and as we increase the number of nodes the overhead time multiplies itself in return of less work per node.

OPENMP VS MAPREDUCE:

In comparing OpenMP and MAPREDUCE we see that initially MAPREDUCE takes less time but as we increase the number of cores OpenMP overtakes MapReduce . As we increase the cores for OpenMP we see that the time decreased to 7->1.16 secs,8->1.82 secs and then it increased slightly i.e. 9->2.43 secs. Performance wise OpenMP is commendable. Also I believe that for a large data set on a single machine it is better to use OpenMP and in a distributed environement when we want CPU to spend as small time as possible and the data is unstructured it is better to go with MAPREDUCE.

GPU VS MAPREDUCE



The amount of time taken by GPU decreases as we increase the number of GPUS which indicates that for a huge problem we can say that if we increase the number of GPUs we can solve the problem in less time. Increasing the number of GPU indicates more memory for the operation and the processing time will be less as the task will be divided in 2 GPUs. When compared with MapReduce I believe that increase in the number of GPU might decrease the time even further and thus it is comparable to MAPREDUCE the only constraint being the memory.

READINGS:

PROGRAM	NUMBER OF NODES	PROBELEM SIZE	TIME
Sequential	1 Node	100,000	0.016586
Sequential	1 Node	1,000,000	0.179556
MapReduce	2 Nodes	100,000	28.128
MapReduce	2 Nodes	1,000,000	29.34
MapReduce	4 Nodes	100,000	27.495
MapReduce	8 Nodes	100,000	33.475
MapReduce	4 Nodes	1,000,000	40.2146
MapReduce	8 Nodes	1,000,000	37.1421
MapReduce	16 Nodes	1,000,000	43.1456

PART 2:
CATEGORIZE TWITTER MESSAGES BY USING MAHOUT NAIVE BAYES CLASSIFIER AND K-MEANS

Dataset used for both the methods is the Twitter data set given in course documents.

/Bayes/
Files
/data/chunk-0 : Hadoop Sequential File
/script/SLURM_NAIVE_BAYES.sh : SLURM FILE
Command to run : sbatch SLURM_NAIVE_BAYES.sh

Naive Bayes involves three stages namely preprocessing, training and testing.
Preprocessing is done using the seq2sparse command which creates the vectors from a sequential file. The generated vectors are divided into two sets namely train-vectors and test-vectors.
The train-vectors are used to train the set to generate a classifier model. This generated classifier model is then tested on both the

training set and testing set.

Training Set:

```
=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.9353
Accuracy                                   98.2249%
Reliability                                85.6527%
Reliability (standard deviation)            0.3469
```

Tesing Set :

```
=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.5369
Accuracy                                   69.5402%
Reliability                                61.1746%
Reliability (standard deviation)             0.271
```

Accuracy for training set : 98.2249 %
Accuracy for testing set : 69.5402 %

/Kmeans/
Files
/data/chunk-0 : Hadoop Sequential File
/script/SLURM_Kmeans.sh : SLURM FILE
Command to run : sbatch SLURM_Kmeans.sh

In kmeans we use the command seq2sparse to generate the vectors. We use the tfidf vectors from the generated vectors and random initial clusters to form output clusters. Then we use the clusterdump command to generate the output in a text file. We parse this file to get the accuracy.

The following is the output of the text file i.e. the file hat we ge after cluserdump.

```
:CL-378{n=0  c=[bargains:4.840,  deal:2.363,  deals:1.259,  http:1.012,
lt:6.407, professional:5.447, t.c
        Top Terms:
                    lt                                              =>
6.40677547454834
                     professional                                   =>
5.4465651512146
                    bargains                                        =>
4.840429306030273
                    deal                                            =>
2.3631272315979004
                    deals                                           =>
1.2594388723373413
                    http                                            =>
1.011788010597229
                    t.co                                            =>
1.009813666343689
```