Project Report

on

"Assignment 3 : Application of MPI and Map-Reduce on K Means clustering Algorithm "

Submitted By

Naman Gupta 133050012

Anshul Agarwal 143050044

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

MPI for K Means

Here the implementation of K Means Algorithm is done in C language. It uses the mpi.h library which supports all the inter process communication functions used in MPI. In this implementation , we have one main process acting as a Master which combines all the data from all the other processes , acting as Slaves. Each process gets a unique number to identify itself and is called as Rank of process. Here we assume the process with rank 0 is the master. It works in the following way :

Role of Master (rank = 0):

1. It initialises the MPI environment.
2. Reads data points from the file.
3. Takes first 5 points as initial clusters.
4. divides data points into equal sets so that each slave receives roughly equal datapoints to operate upon.
5. After slaves finish their computation , the master collects the information regarding the new Parent clusters of each point.
6. It outputs the result in new files : one has datapoints along with their parent clusters and second file has the final computed clusters.

Role of Slaves (rank != 0):

1. they receive set of datapoints.
2. they check for each point , which cluster is closest to it by calculating the euclidean distance.
3. This closest cluster is assigned as member of the point.
4. The cluster value is recomputed as average of all the points belonging to that cluster .
5. It then checks if this new value of cluster differs from the previous value by some Threshold.
6. If difference is greater than threshold , then the algorithm is run again.
7. else it stops and outputs the result to the master.

Compiling the program:
$mpicc mpi_kmeans.c mpi_main.c mpi_io.c file_io.c -o Kmeans_final

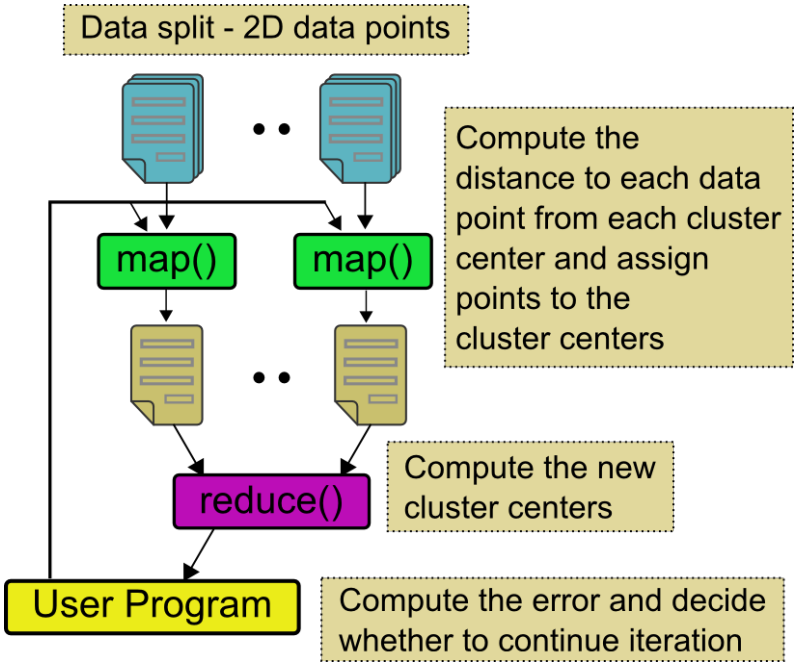
Running the program
$mpirun -np <number of processes> <path to file Kmeans_final> <path to directory containing input data> <path of output directory>

Map-Reduce For K Means

We applied MapReduce style parallel algorithm for K-Means Clustering.

Two files were given as input, first one is the file containing the data points and second one contains the centroids. Next we select k points to be the centroids of k clusters which at present have no members. The list of centroids can be selected by any method (e.g., randomly from the set of data points). We took top k data points from the input dataset as our initial choice of cluster centroids.

Figure 2.1 shows the flow of map-reduce when applied for clustering. The map function performs the procedure of assigning each sample to the closest center while the reduce function performs the procedure of updating the new centers.



Above process is executed above till either centroids point vary little (P distance unit) or number of iterations (T) reaches its threshold. For our implementation, P was taken as 0.2 and T was taken to be 100.

1. Map Function : The input dataset is stored on HDFS. The dataset is split and globally broadcast to all mappers. Each mapper computes the nearest centroid for each data point it receives. It sends <centroid,datapoint> as a key value pair to the reducers.
2. Reduce Function : It receives <centroid,List<datapoints> > from the mappers. It computes new centroid by averaging over the data points it has received.

3. Process (1) and (2) continues till new centroid vary by 0.2 in euclidean distance with the old centroid or the number of iterations is below threshold.

Experiment 1
Performance of MapReduce on Kmeans

We performed experiment on different dataset generated using script provided on moodle. Number of clusters were chosen to be 5. As mentioned earlier initial set of cluster centroid are first 5 entries in dataset file. We did 2 different setup for hadoop clusters.

SETUP 1: Hadoop Cluster of 3 Machines

Hadoop cluster was set up on 3 machines 10.12.14.54 (master), 10.12.5.16 (slave), 10.105.19.55(slave) and experiments were performed with 1L, 2L, 3L and 5L, 25L data points.

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
25 L	1	183,240	24,390	207,630
	2	196,330	23,230	219,560
	3	204,660	24,030	228,690
	4	216,510	22,960	239,470

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
5 L	1	36,280	7,250	43,530
	2	34,710	7,000	41,710
	3	34,030	8,600	42,630

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
3 L	1	24,370	6,610	30,980
	2	23,940	6,200	30,140
	3	25,340	6,180	31,520

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
2L	1	16,440	6,030	22,470
	2	16,400	6,030	22,430
	3	16,460	5,590	22,050
	4	16,370	5,220	21,590
	5	16,360	5,460	21,820
	6	16,060	6,480	22,540
	7	17,420	6,040	23,460
	8	16,660	5,890	22,550
	9	15,720	5,560	21,280

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
1 L	1	9,810	4,710	14,520
	2	9,030	5,000	14,030
	3	9,780	4,700	14,480
	4	8,710	4,720	13,430

Observations

1. Map and Reduce per iteration time increases as the size of dataset gets increased.
2. We tried to configure maximum map tasks as 10,20 and with other values but only 1 map task were getting triggered with datasets having size as 1L, 2L, 3L,5L and 2 map tasks were triggered for 25L dataset.
3. Reduce task, inspite of configuring it with different parameters, only 1 reducers was triggered.
4. Choice of initial cluster plays role in deciding convergence of the algorithm. For example with 2L datapoints, 9 iterations were required for algorithm to converge. On observation of the data points we found that initial cluster centroids were close to one another and represents data points of single cluster.

SETUP 1: Hadoop Cluster on Blade Server of InfoLab

Another set of Experiments were performed on Hadoop Server available on **Blade Server (Infolab)**. This cluster has **40 CPU cores** and capacity of **40 map tasks and 20 reduce tasks**.We performed experiments with 50L, 25L, 5L and 3L data points.

1. **50 L Data Points** : Number of Map task were set to 10 but only 4 got triggered. Number of reduce tasks were set to 10 but only 1 got triggered.

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)
50 L	1	285,040	67,570	352,610
	2	274,170	65,820	339,990
	3	268,970	66,260	335,230
	4	281,600	68,380	349,980
	5	272,560	66,900	339,460

2. **25 L Data Points** : Number of Map task were set to 10 but only 2 got triggered. Number of reduce tasks were set to 10 but only 1 got triggered.

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)	Speed Up (SETUP 1 /SETUP 2)
25 L	1	126,440	31,870	158,310	1.31
	2	123,370	31,820	155,190	1.41
	3	130,980	30,950	161,930	1.41
	4	130,630	31,310	161,940	1.47

3. **5 L Data Points** : Number of Map task were set to 10 but only 1 got triggered. Number of reduce tasks were set to 10 but only 1 got triggered.

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)	Speed Up (SETUP 1 /SETUP 2)
5L	1	28,880	8,810	37,690	1.15
	2	25,170	8,090	33,260	1.25
	3	24,980	7,870	32,850	1.29

1. **3 L Data Points** : Number of Map task were set to 10 but only 1 got triggered. Number of reduce tasks were set to 10 but only 1 got triggered.

Number of DataPoints	Iteration	Map CPU Time (ms)	Reduce CPU Time (ms)	Total Time (ms)	Speed Up
3L	1	17,950	5,840	23,790	1.30
	2	18,060	5,930	23,990	1.26
	3	17,750	5,790	23,54	1.33

Observation

1. By increasing the computation power per iteration total time got reduced. We achieved speedup of 1.4, 1.23, 1.29 for 25L, 5L and 3L data points respectively.

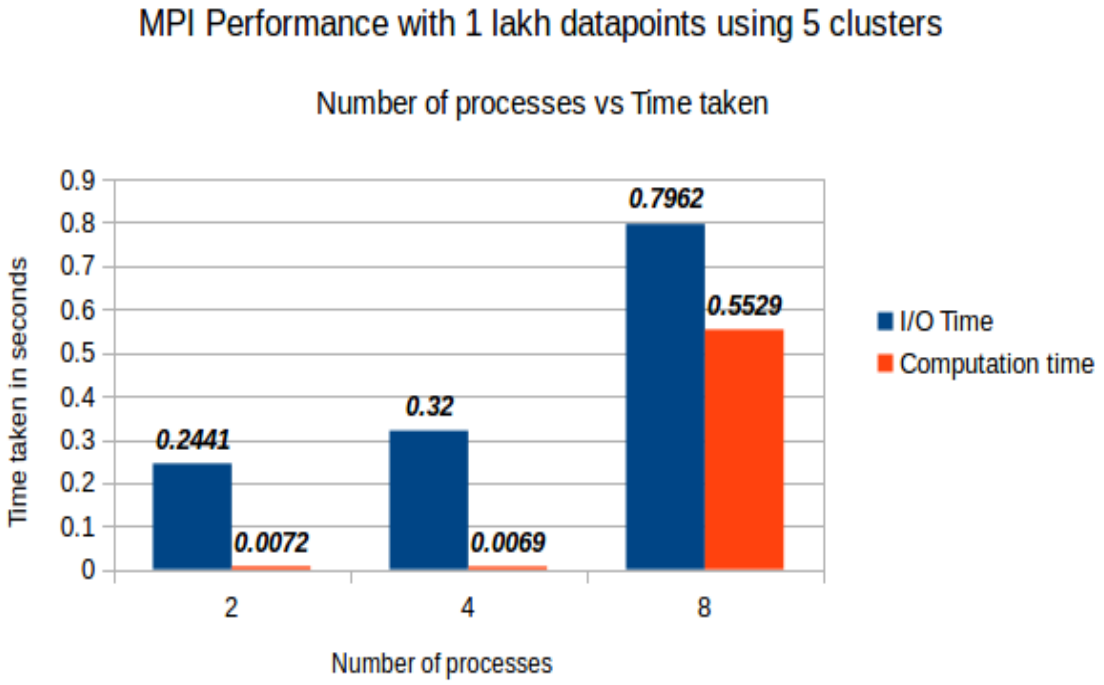
2. But inspite of having large good number of mappers (40) and reducers(20) slot available only 4 mappers and 1 reducers were triggered. We tried to change the configuration settings but there was no increase in the map or reduce slots.

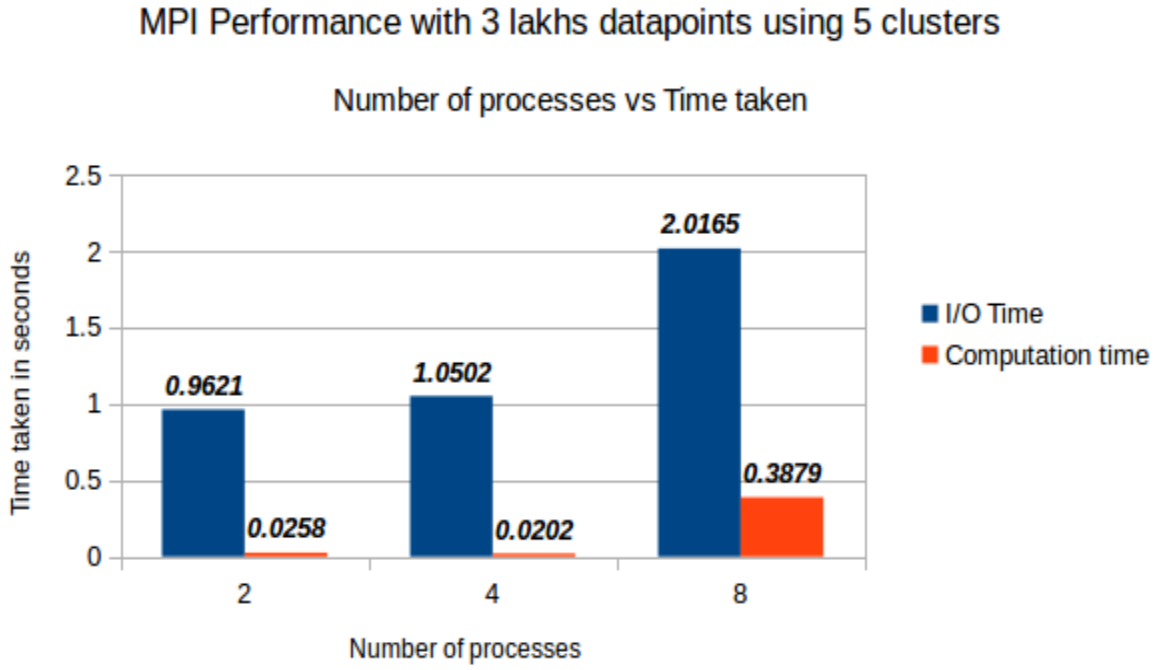
Experiment 2
Performance Analysis of MPI

We performed experiment on different dataset generated using script provided on moodle. Number of clusters were chosen to be 5. As mentioned earlier initial set of cluster centroid are first 5 entries in dataset file.

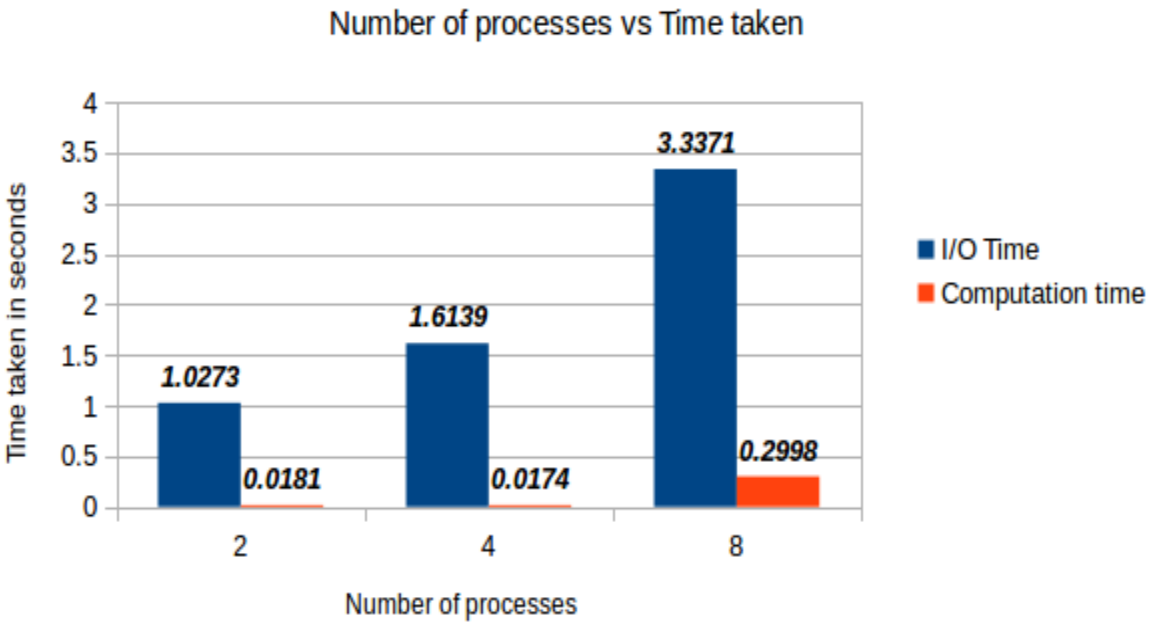
Setup: MPI on Machine with 4 cores

Number of Datapoints (in Lakhs)	I/O Time | Computation Time (seconds) # Process = **2**	I/O Time | Computation Time (seconds) # Process = **4**	I/O Time | Computation Time (seconds) # Process = **8**
1L	0.2441 | 0.0072	0.32 | 0.0069	0.7962 | 0.5529
3L	0.9621 | 0.0258	1.0502 | 0.0202	2.0165 | 0.3879
5L	1.0273 | 0.0181	1.6139 | 0.0174	3.3371 | 0.2998
25L	6.1070 | 0.0938	9.4552 | 0.0876	19.7417 | 0.5278
50L	12.5435 | 0.2352	23.9480 | 0.2282	44.9417 | 0.5964
75L	17.7286 | 0.4145	30.9759 | 0.4107	67.7534 | 1.3035

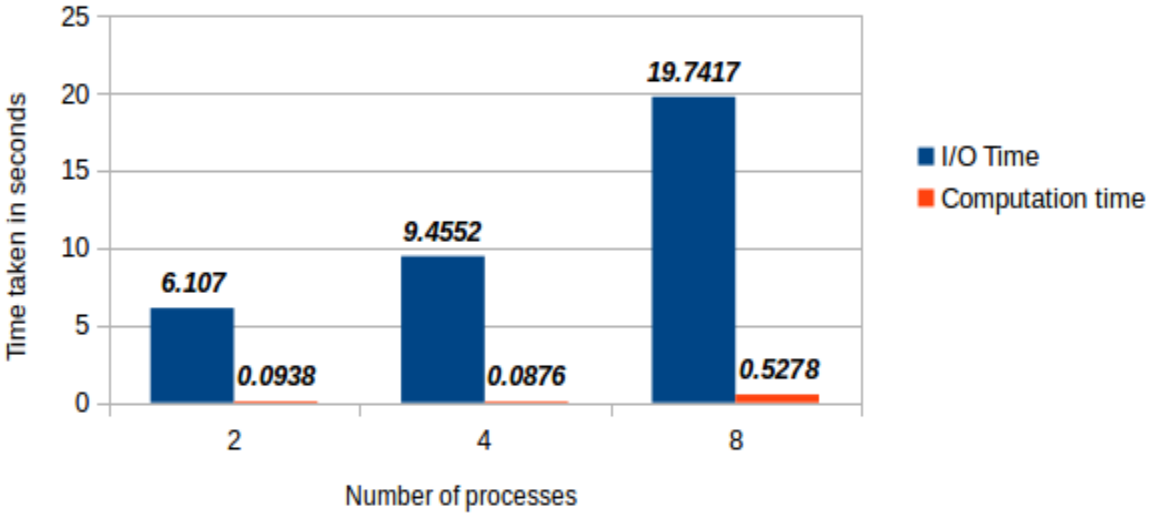

MPI Performance with 1 lakh datapoints using 5 clusters
Number of processes vs Time taken
Time taken in seconds
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.2441
0.0072
0.32
0.0069
0.7962
0.5529
2
4
8
Number of processes
I/O Time
Computation time


MPI Performance with 3 lakhs datapoints using 5 clusters
Number of processes vs Time taken
Time taken in seconds
2.5
2
1.5
1
0.5
0
0.9621
0.0258
1.0502
0.0202
2.0165
0.3879
2
4
8
Number of processes
I/O Time
Computation time

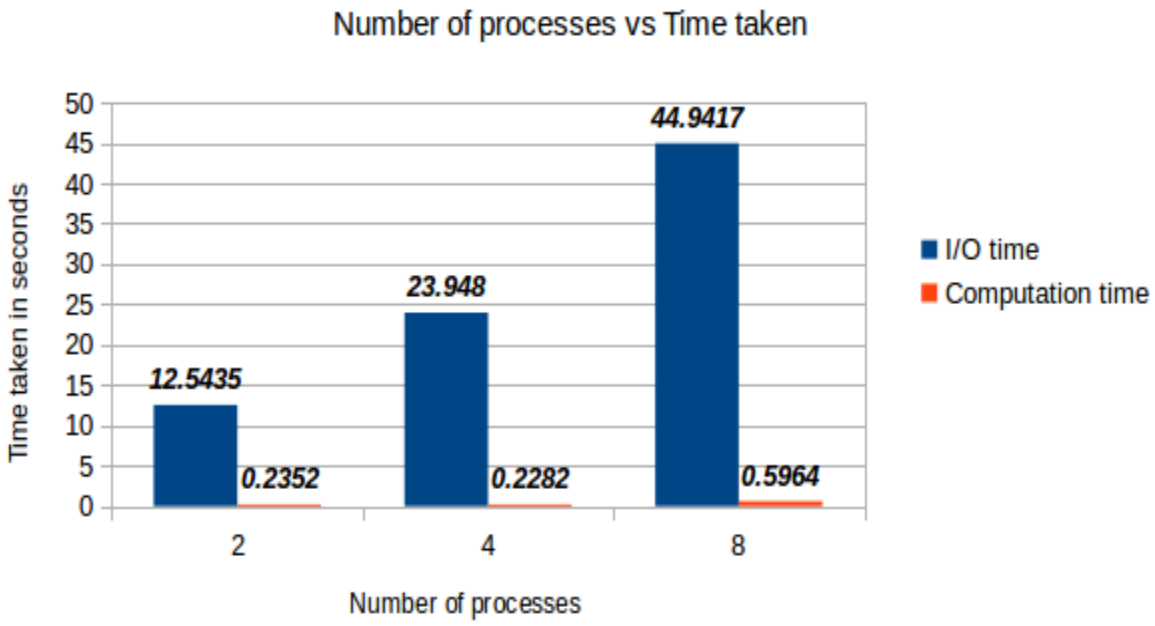
MPI Performance with 5 lakhs datapoints using 5 clusters



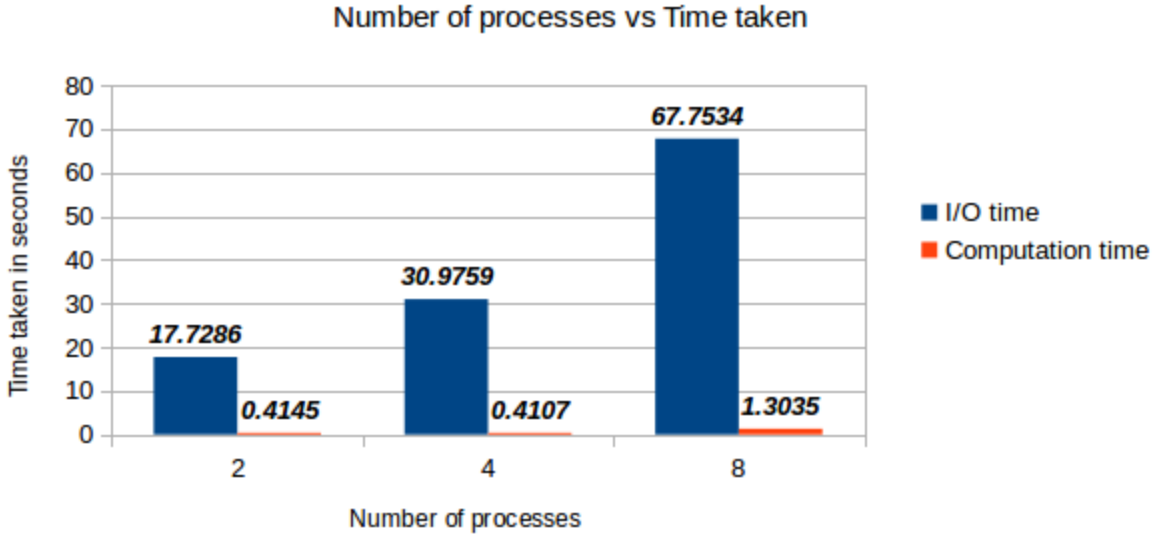
MPI Performance for 25 lakhs datapoints using 5 clusters



MPI Performance with 50 lakhs datapoints using 5 clusters



MPI Performance with 75 lakhs datapoints using 5 clusters



Analysis of MPI Performance:

The following points can be inferred from the graph:

1. As number of processes increases , the Time taken for I/O to and from files also increases as there are more processes, so more reading and writing of more number of files occur.

2. As number of processes increases from 2 to 4 , the computation time decreases as more processes bring more parallelism into computation so time is reduced.

3. But when number of processes increases from 4 to 8 , the computation time increases as now message passing between more number of processes dominate the time reduction due to parallelism . So the overall computation time goes up.

MPI vs Map Reduce

MPI is a message passing library interface specification for parallel programming.
MapReduce is a Google parallel computing framework. It is based on user-specified map and reduce functions.

In general, MapReduce is suitable for non-iterative algorithms where nodes require little data exchange to proceed (non-iterative and independent); MPI is appropriate for iterative algorithms where nodes require data exchange to proceed (iterative and dependent) (Chen et. al.)[1]

Since kmeans is an iterative algorithm, MPI performs much better in comparison to Map Reduce paradigm. We experimented on datasets with 25 L and 50 L 2D Points.

Configuration

1. MPI :
    a. Number of Process : 8 on same machine
2. Map Reduce
    a. 1 Reducer
    b. Number of Map Task : 4 for 50L, 2 for 25L

Results

Number of Data Points	Iteration	MPI I/O time (ms)	MPI Computation Time (ms)	Total Time (ms)	Map Time (ms)	Reduce time (ms)	Total Time
25L	1	5,019	129	5,148	126,440	31,870	158,310
	2	4,273	115	4,388	123,370	31,820	155,190
	3	5,260	143	5,403	130,980	30,950	161,930
	4	5,189	140	5,329	130,630	31,310	161,940
50L							
	1	9266	120	9,386	285,040	67,570	352,610
	2	8913	116	9,029	274,170	65,820	339,990

	3	8744	118	8,862	268,970	66,260	335,230
	4	9155	122	9,277	281,600	68,380	349,980
	5	8863	120	8,983	272,560	66,900	339,460

Experience in applying MPI and MapReduce to K-Means clustering algorithm

Installing and running MPI does not cause much problems as it requires less number of resource handling(unlike hadoop). Here each process is concerned with handling of messages from other processes and performing local computations . When multiple MPI processes were run on a single 4 core machine , the performance was still very good and could be used for analysis of data points ranging to millions of points easily , within seconds.

The main challenge in applying map-reduce is the setting up of hadoop cluster. Initially we executed our algorithm on a single node cluster. Inspite of algorithmic and parametric changes this setup was not able to scale up with > 10L points. Algorithm took around an hour to converge on single system when tested with 50L points.

We set up a small hadoop cluster of 3 machines (Our Laptops, lab machine). This setting was able to scale with large datasets. Lastly we experimented with hadoop cluster available at INFOLAB, KRESIT.

Map Reduce paradigm when applied to KMeans did not perform very well as compared to MPI. The main reason we could deduce is because of iterative nature of KMeans where each iteration is dependent on cluster centers computed by previous iterations.

Performance trade-offs of MPI and MapReduce with K-Means

With **MPI** , it was observed that there were two main factors of consideration that affected the two issues of performance. The two main factors are: Number of messages passed and amount of parallelism achieved. The two performance issues are : Input-Output (IO) time and total Computation time. They are related as follows:

1. As number of processes increases , the Time taken for I/O to and from files also increases as there are more processes, so more reading and writing of more number of files occur.
2. As number of processes increases from 2 to 4 , the computation time decreases as more processes bring more parallelism into computation so time is reduced.
3. But when number of processes increases from 4 to 8 , the computation time increases as now message passing between more number of processes dominate the time reduction due to parallelism . So the overall computation time goes up.

In **map-reduce**, we found that with increase in the computation power helps to speed up the entire process. But inspite of high computing capabilities of BLADE server (40 Mappers. 20 Reducers) very few numbers of mappers and reducers were triggered. We still need to investigate the exact reason for this behaviour.

Applicability of K-Means to MPI and MapReduce

On and all , K-Means is implementable by both MPI and Map-Reduce . It gives different performance behaviour depending on different types of systems , size of data and ways of implementation (MPI and Map-Reduce). There are three situations that can arise:

1. If the processes always run on a single system , then MPI is preferable as compared to Map-Reduce as MPI is simple to implement , simple to debug and simple to analyse by changing the parameters.
2. If the processes always run on different systems , then we can use hadoop/Map-Reduce as it has a very systematic in-built monitor to control , observe and take actions on a distributed environment . Also it can easily handle Huge amounts of data unlike MPI by adding massive amount of parallelism. Here control through MPI is not that flexible and

robust and it may not make use of the total distributed environment as well as hadoop can make use of , like Google uses hadoop for managing its never ending data in data centers.

3. If processes may or may not run on different systems , then we can use hadoop if systems are high performing systems and have to handle huge amount of data , else we can use MPI for small systems and not very large data as it is simple and has limited applicability.

Recommendations regarding the usage of MPI and MapReduce for algorithms similar to K-means.

As per our understanding from the assignment and other literature, MPI is best suited for problems that require a lot of interprocess communication.

When Data scales to petabytes, and there is little interprocess communication, MPI becomes a pain because, the processes will spend all the time sending data to each other (bandwidth becomes a limiting factor) and CPUs will remain idle. Perhaps an even bigger problem is reading all that data.This is the fundamental reason behind having something like Hadoop. where data is not passed but distributed across multiple nodes

K Means is an iterative algorithm where each iteration depends on previous iteration, such kind of algorithm does not fit for Map Reduce paradigm. These are suited for algorithms that are sequential like word count. Literature proposes hybrid setup combining MPI and Mapreduce for solving iterative algorithms applied to very large dataset.

Acknowledgments

We would like to thank Ashish Mittal (http://www.cse.iitb.ac.in/~ashishm/) for his support on hadoop installation and setup.

References

1. Chen, Wen-Yen, et al. "Parallel spectral clustering in distributed systems."*Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.3 (2011): 568-586.
2. http://web.eecs.utk.edu/~dongarra/ccgsc2010/slides/talk12-fox.pdf

3. Hadoop Installation and Setup
    a. http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
    b. http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/
    c. https://github.com/thomasjungblut/thomasjungblut-common/tree/master/src/de/jungblut/clustering/mapreduce (Modified for our implementation)

4. MPI download and setup
    a. http://www.mpich.org/
    b. https://github.com/serban/kmeans (Modified for our implementation)

5. William Gropp, Ewing Lusk, Nathan Doss, Anthony Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, Volume 22, Issue 6, September 1996, Pages 789-828, ISSN 0167-8191, http://dx.doi.org/10.1016/0167-8191(96)00024-5.(http://www.sciencedirect.com/science/article/pii/0167819196000245)
6. MPI--the Complete Reference: The MPI core - Marc Snir