STUDY OF SCALABILITY – HADOOP

&

Mahout Classification

Contents

Problem Statement

Using GPUs to speed up the programs that have been built so far. The prime number generator and the Bucket sort application. We have seen that both the programs are highly parallelizable.

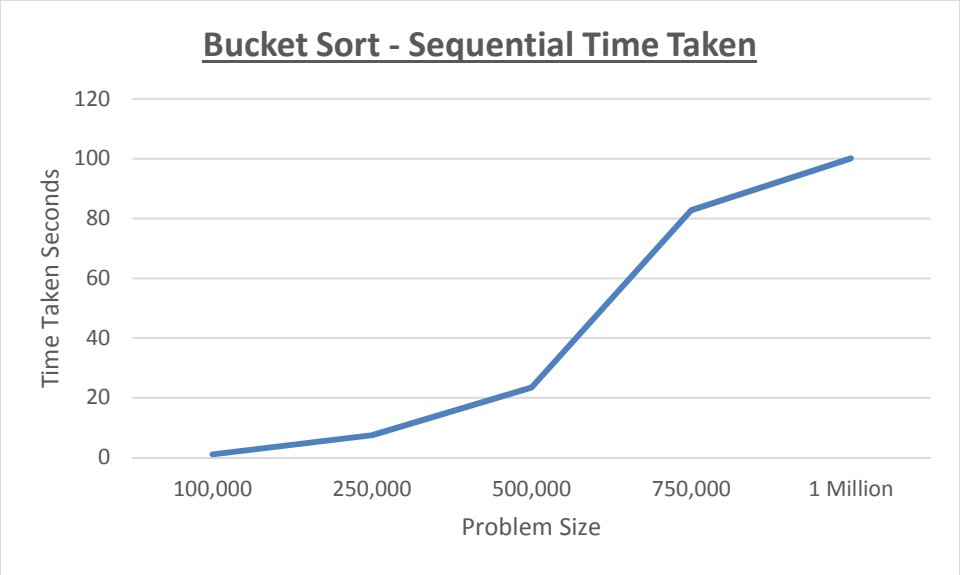
Bucket Sort: In bucket sort the idea is to split the input data into evenly loaded buckets which are later sorted and results appended together. This task of sorting a bucket can be done in parallel. In this exercise I have used Hadoop. I use the following to get the task done:

- Create buckets based on Z values of a standard normal distribution – to create almost even sized buckets
- Spawn as many reducers as there are nodes
- Custom partitioner to send one key to a reducer – in the case of 16 nodes 8 reducers get 2 keys there by making the other 8 idle. To overcome this I user custom partitioner
- Insertion sort to sort the values in each reducer
- Send parameters to mappers using configuration
- Find the time taken by each reducer using TaskCompletion events to find the time taken by each mapper and reducer tasks.

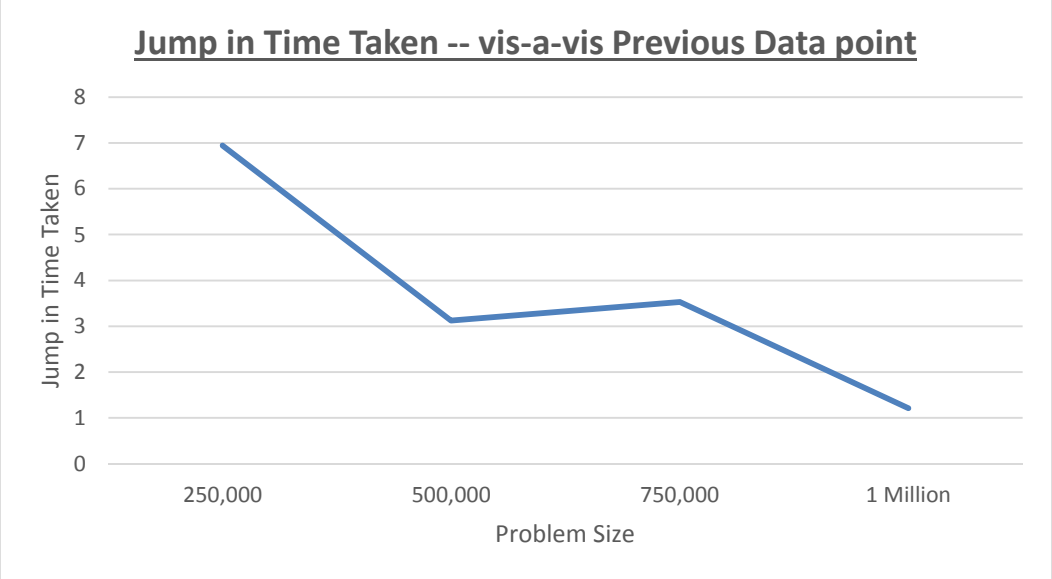
Sequential Code

Bucket Sort:

	Time Taken	
	Seconds	**Minutes**
100,000	1.081097	0.018018
250,000	7.50711	0.125119
500,000	23.45483	0.390914
750,000	82.738692	1.378978
1 Million	100.092278	1.668205



Prob Size	Delta Increase in Time
100,000	
250,000	6.9439745
500,000	3.124348784
750,000	3.527575855
1 Million	1.209739671



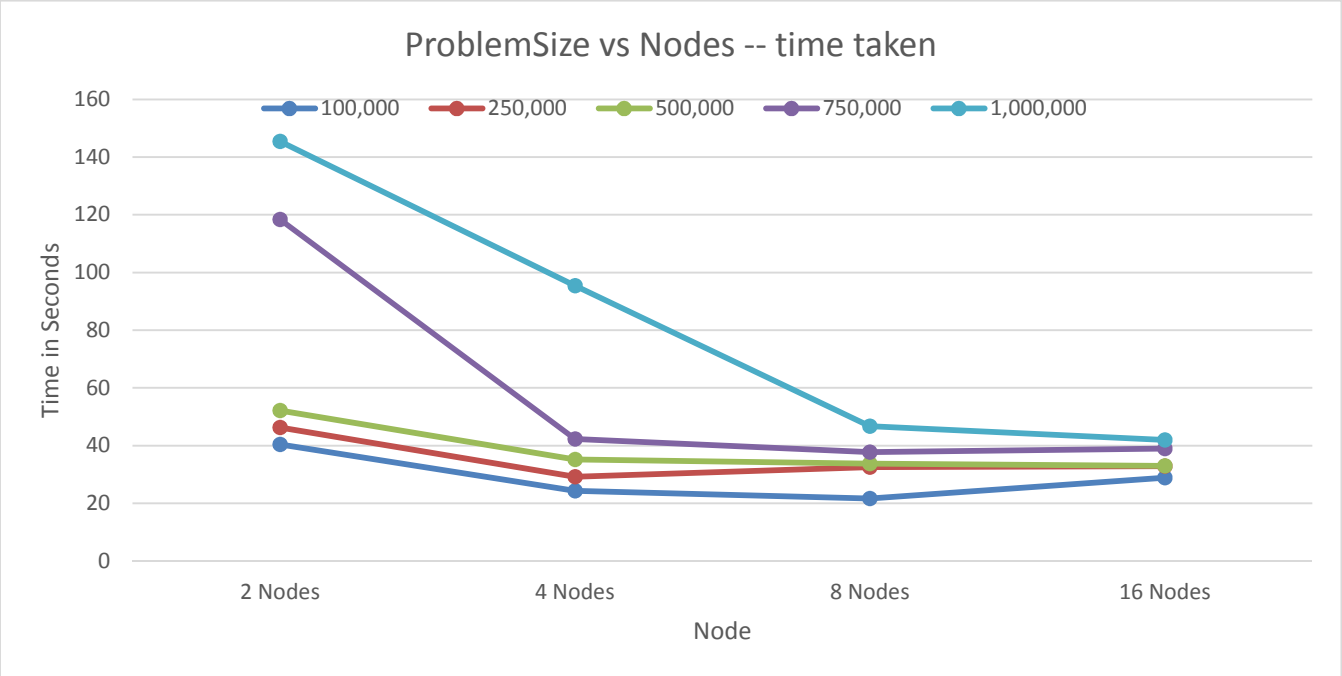
As can be noticed we see that there is an increase in time as the problem size increases.

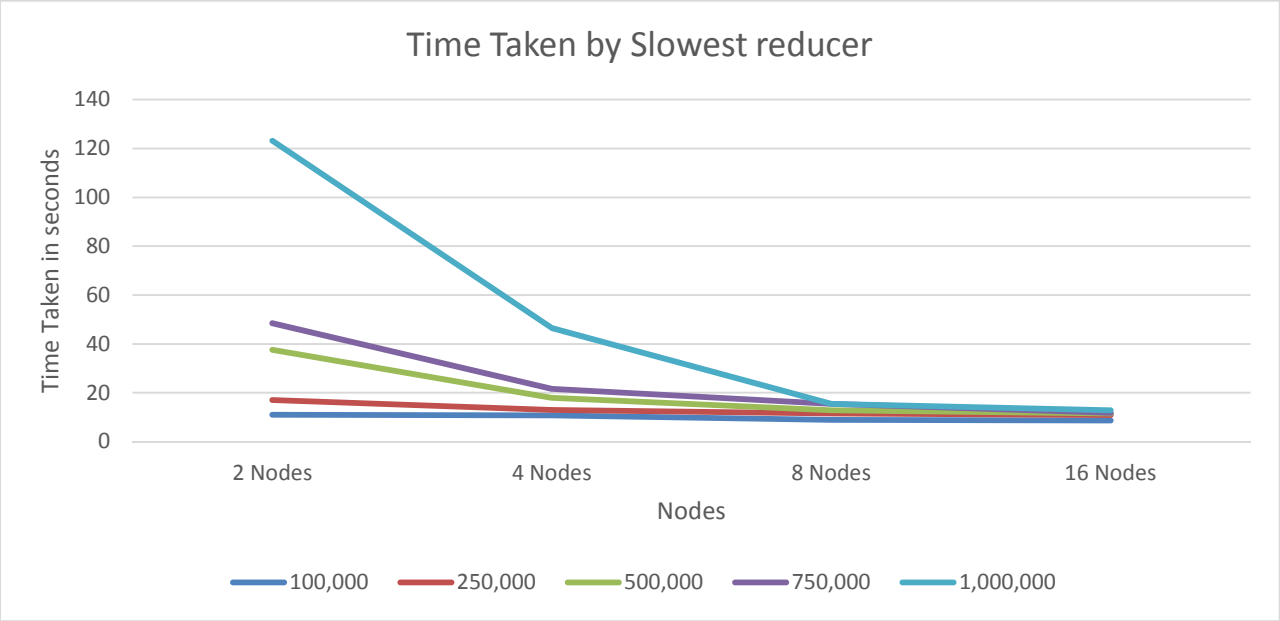
HADOOP

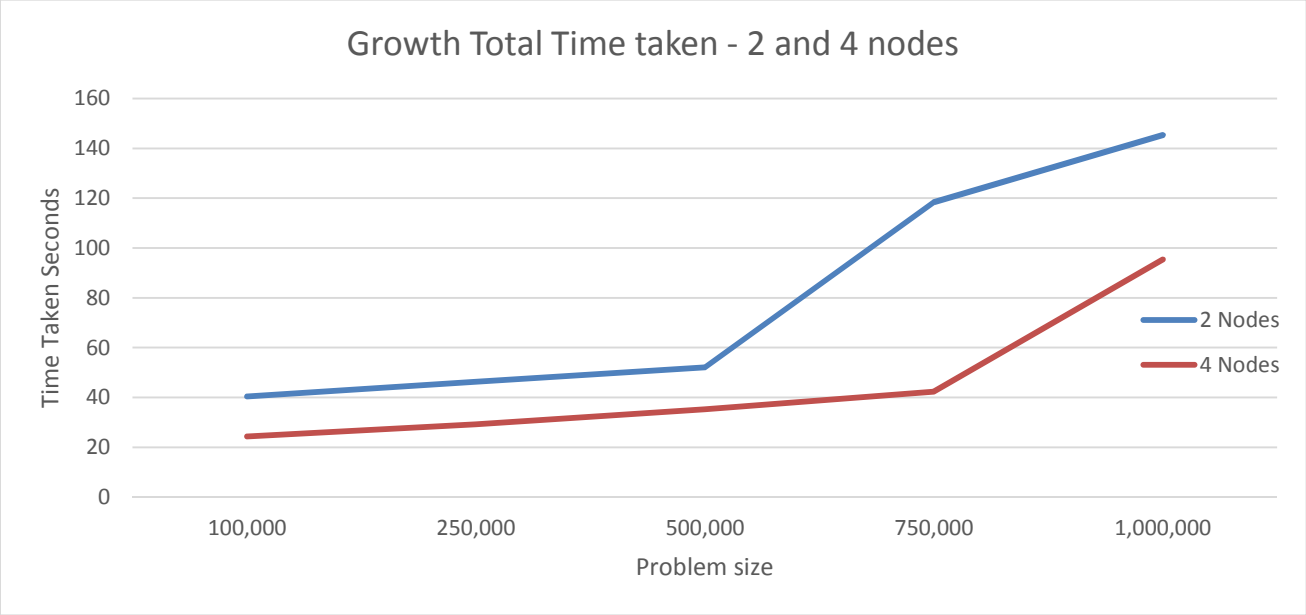
Bucket Sort results: Since Hadoop requires some time to be setup including that time could skew the results in favor of other frameworks. Therefore I have made an attempt to capture the reducer times for each configuration and recorded the slowest reducer. This shall give us some idea about what is happening and Hadoop a fair chance to compete with other programming models.

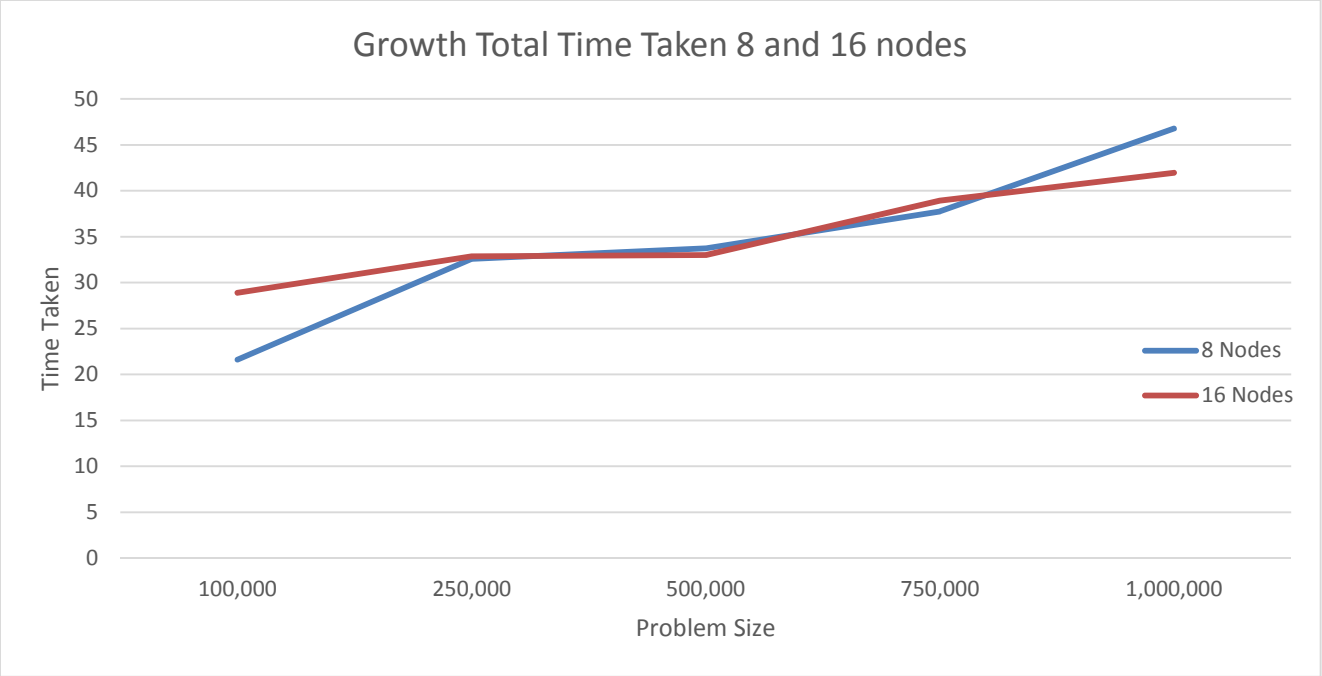
Total Time Takens - As measured from main (Seconds)				
Problem Size\nodes	2 Nodes	4 Nodes	8 Nodes	16 Nodes
100,000	40.4	24.3	21.6	28.88
250,000	46.322	29.2	32.57	32.88
500,000	52.14	35.2	33.75	33.01
750,000	118.4	42.28	37.73	38.91
1,000,000	145.41	95.45	46.759	41.96
5,000,000	0	615.25	312.95	176.83

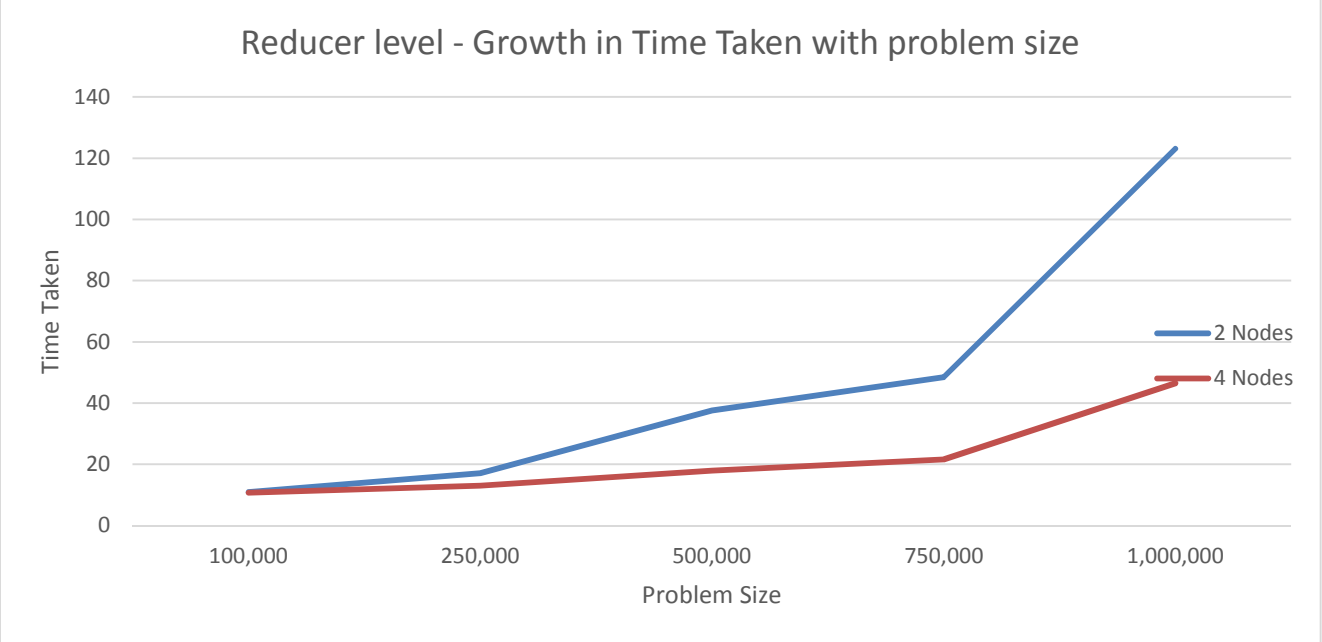
Time Taken - by the slowest reducer (Seconds)				
Problem Size\nodes	2 Nodes	4 Nodes	8 Nodes	16 Nodes
100,000	10.979	10.771	8.894	8.617
250,000	17.085	13.04	11.481	10.87
500,000	37.631	17.97	12.831	11.569
750,000	48.5	21.64	15.41	11.82
1,000,000	123.09	46.48	15.53	12.83
5,000,000	NA	524.42	221.9	69.7

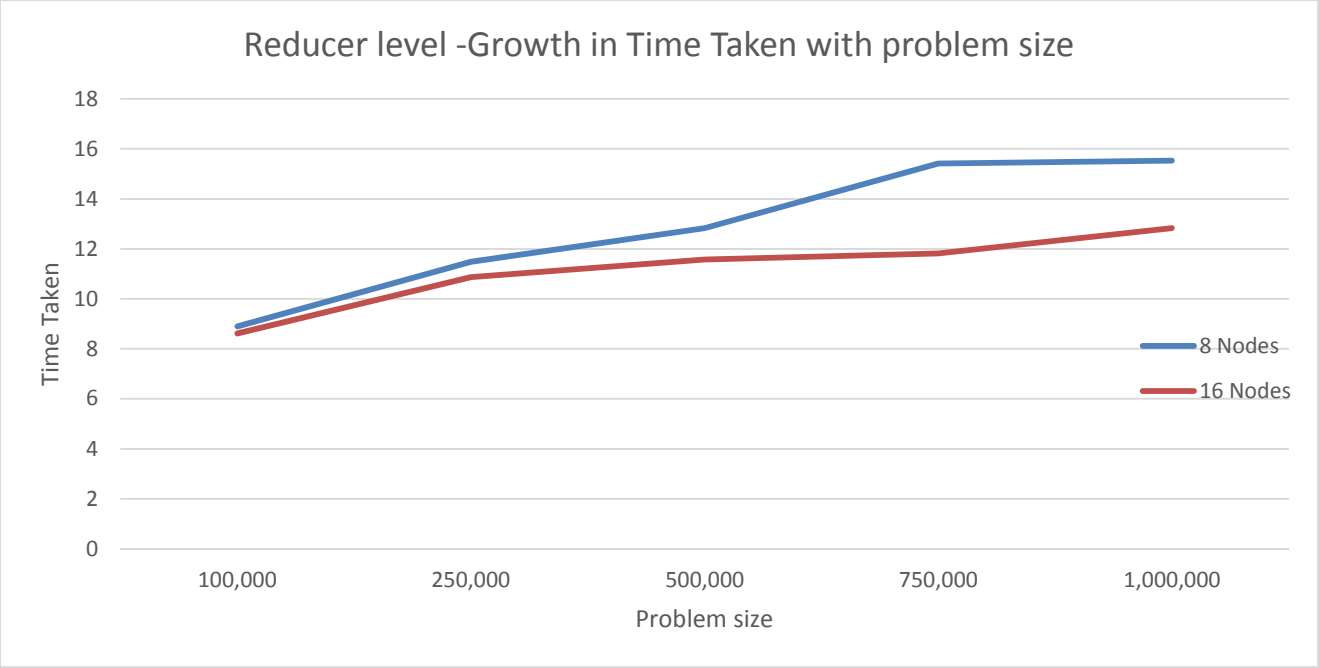



Time Taken by Slowest reducer
Time Taken in seconds
140
120
100
80
60
40
20
0
2 Nodes
4 Nodes
8 Nodes
16 Nodes
Nodes
100,000
250,000
500,000
750,000
1,000,000


Growth Total Time taken - 2 and 4 nodes
Time Taken Seconds
160
140
120
100
80
60
40
20
0
100,000
250,000
500,000
750,000
1,000,000
Problem size
2 Nodes
4 Nodes


Growth Total Time Taken 8 and 16 nodes
Time Taken
50
45
40
35
30
25
20
15
10
5
0
100,000
250,000
500,000
750,000
1,000,000
Problem Size
8 Nodes
16 Nodes


Reducer level - Growth in Time Taken with problem size
Time Taken
140
120
100
80
60
40
20
0
100,000
250,000
500,000
750,000
1,000,000
Problem Size
2 Nodes
4 Nodes



Time taken by Hadoop to preform bucket sort is huge. But one can note the following:

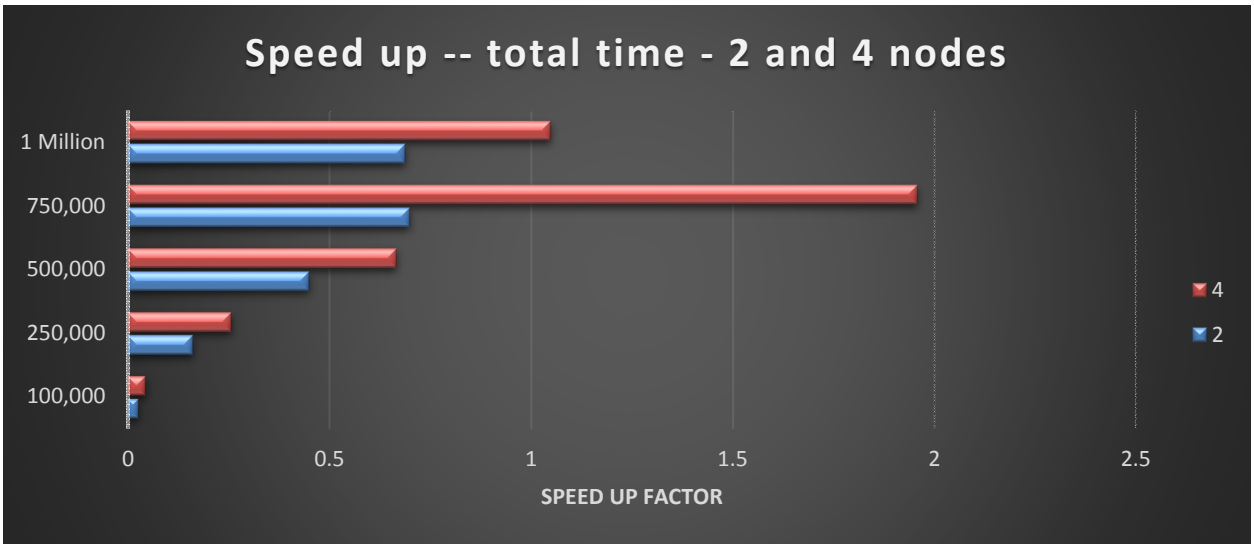
- With increase in number of nodes the performance for a given problem size becomes better
- For each configuration of nodes the growth in time as the problem size grows is an interesting graph; for 2 and 4 nodes there seems to be a steady growth in time with increase in problem size, however for 16 and 8 nodes the time does not grow that fast
- With 8 and 16 nodes we get better performance at larger problem size levels

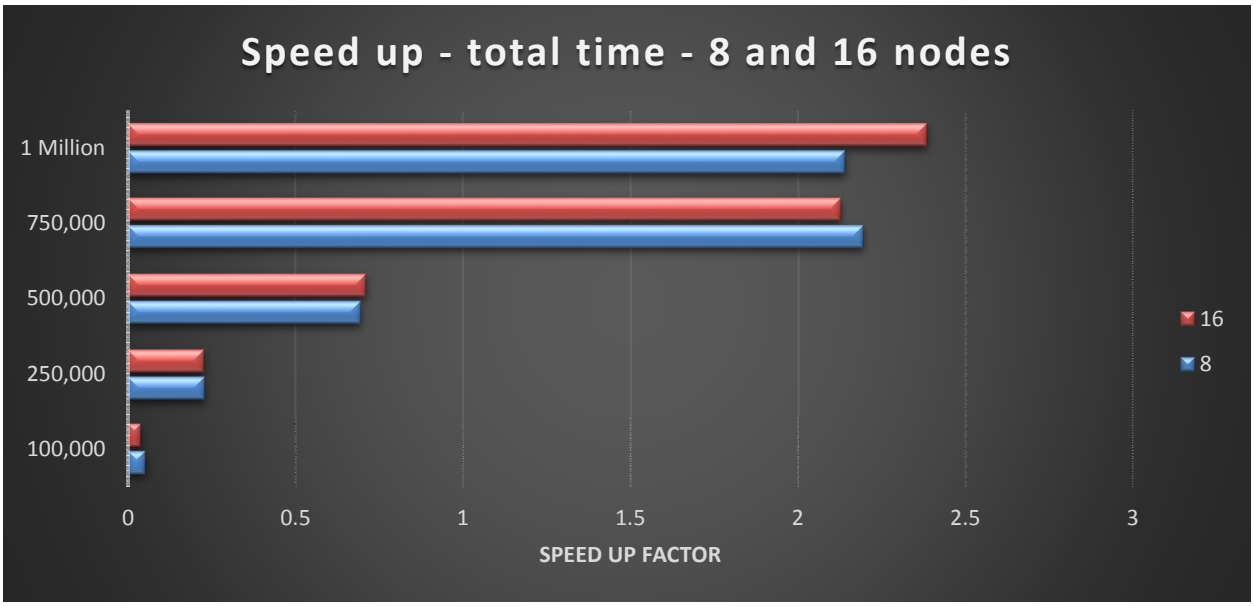
Speed-up:

Hadoop does a bad job of speeding up a process when:

1. The number of nodes is too small &
2. Problem Size is not large enough

Speed - up Calculated using total time				
	2	4	8	16
100,000	0.02676	0.04449	0.050051	0.037434
250,000	0.162064	0.257093	0.230492	0.228318
500,000	0.449843	0.66633	0.694958	0.710537
750,000	0.698807	1.956923	2.192915	2.126412
1 Million	0.688345	1.048636	2.140599	2.385421

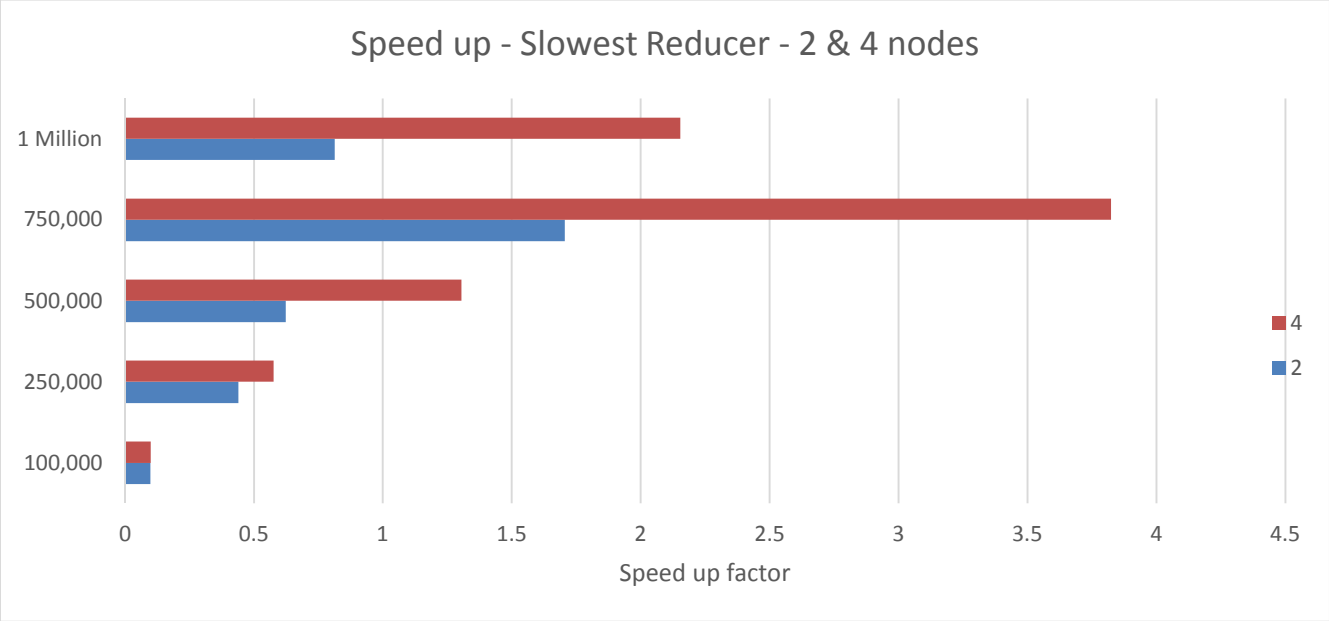


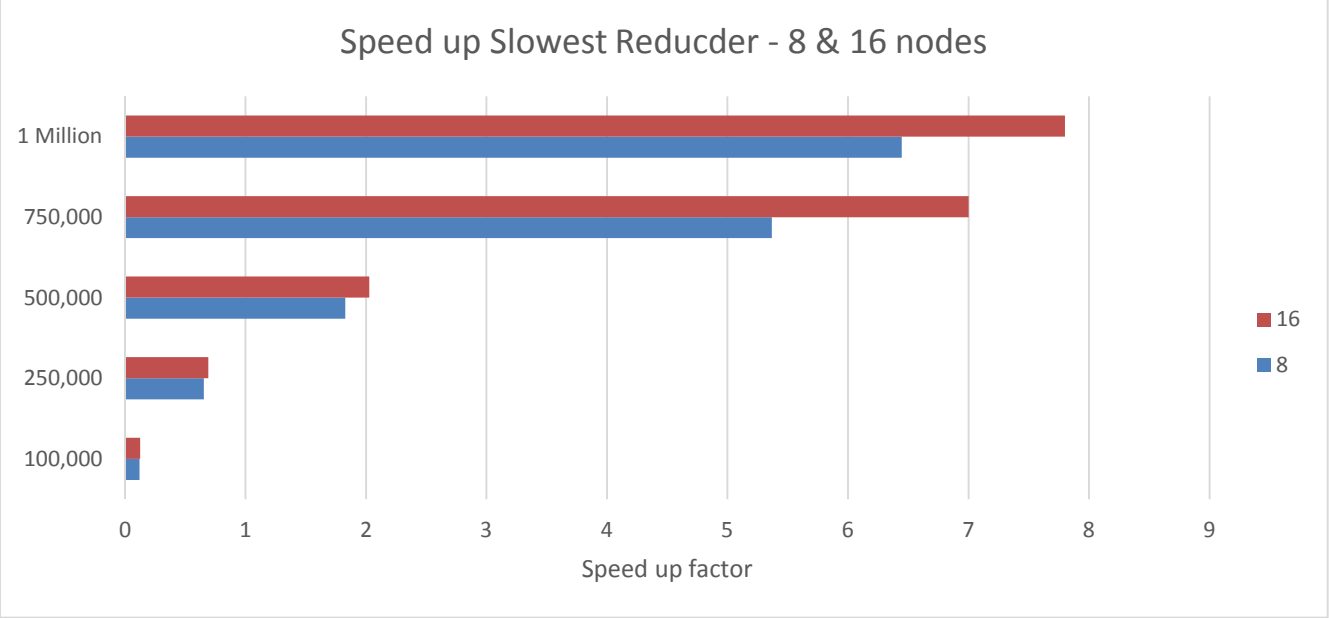


Speed up Notes:

- Hadoop is truly made for Big data and big data only. For relatively smaller data sizes we see that no matter how many cores are thrown at the problem is the speed is non-existent in-fact the code preforms worse than sequential code
- For large problem sizes we do see a speed-up however that speed up is not significant compared to other programming frameworks. This could possibly be due to the fact that sorting is done at the reducer and one is not using the implied sorting performed by the framework
- As expected better speed up results for higher number of cores

Speed -up calculated using slowest reducer				
	2	4	8	16
100,000	0.098469533	0.100371089	0.121554	0.125461
250,000	0.439397717	0.57569862	0.653872	0.690626
500,000	0.623284792	1.30522148	1.827981	2.027386
750,000	1.705952412	3.823414603	5.369156	6.999889
1 Million	0.81316336	2.153448322	6.445092	7.801425





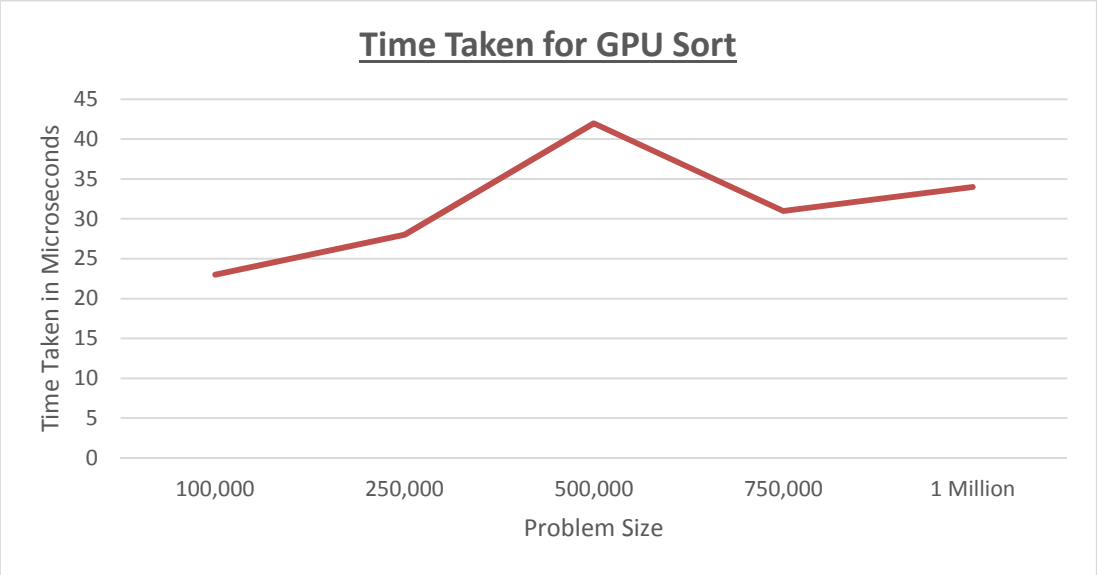
The slowest reducer results mirror those of the total time taken and show a correlation. However if one chooses to ignore the start and shutdown time of Hadoop then one can see that there is speed all along

the places where total time speeds up. At the reducer level we see close to 6 and 7 speed up factors but nothing can be read much into it.

GPU – CUDA

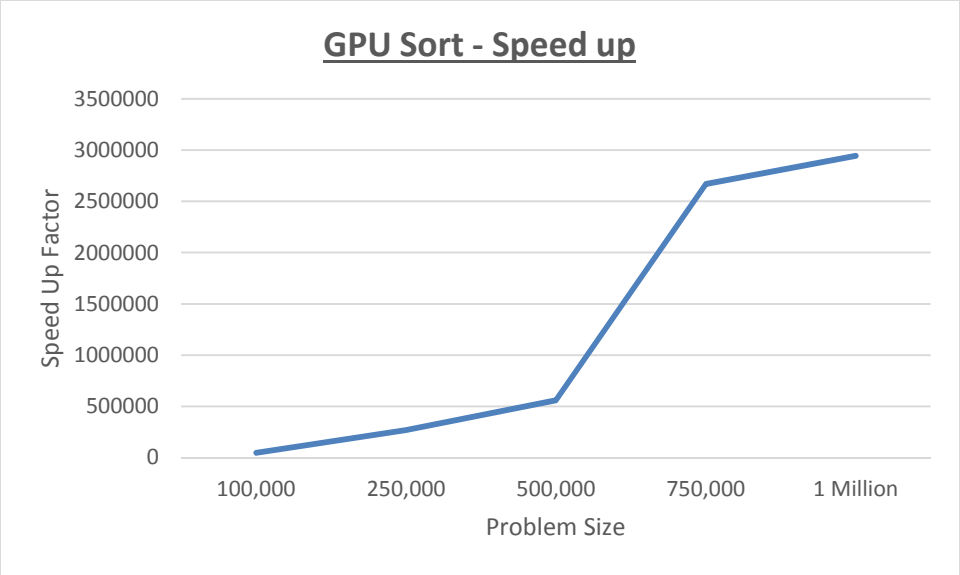
Bucket Sort:

	Time Taken	
	Seconds	**Micro Seconds**
100,000	0.000023	23
250,000	0.000028	28
500,000	0.000042	42
750,000	0.000031	31
1 Million	0.000034	34



Problem Size	Delta Increase in Time
250,000	1.217391304
500,000	1.5
750,000	0.738095238
1 Million	1.096774194

As can be noticed that we see a trend in the case above. It does take more time with increase in the problem size.



We again see massive scaling from the GPU. This is primarily due to the ability to launch many threads to do massive parallelization.

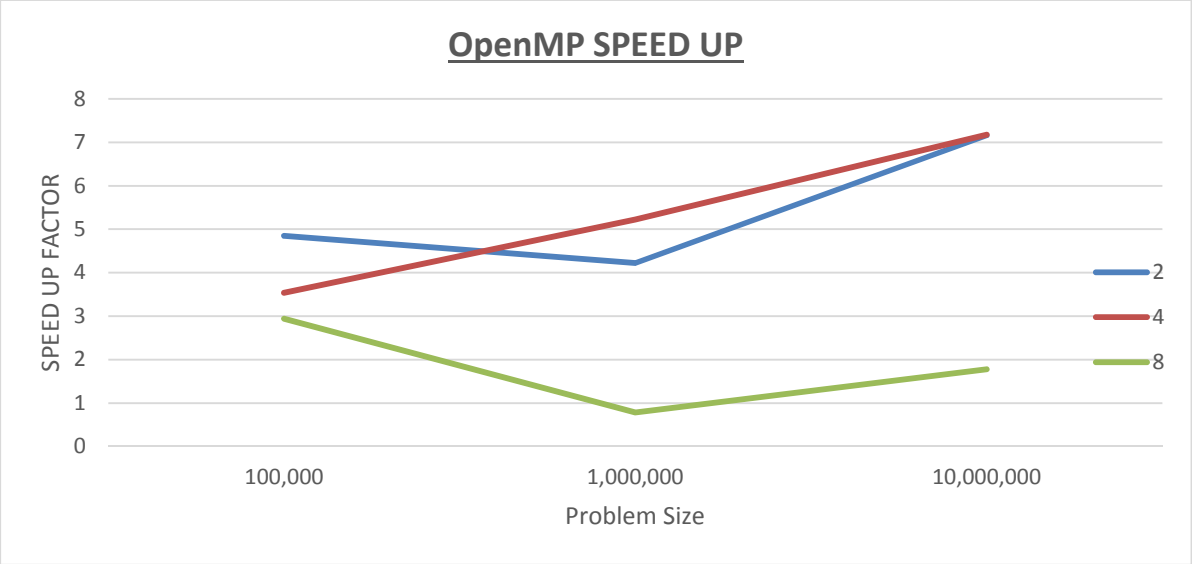
These results did seem too good to be true. I therefore had made some checks, such as for the prime numbers: Listing the largest prime number & also counting the number of primes for a given a limit. And for the sort I went ahead and checked if the output is sorted in the ascending order. The codes that have been shared have these checks and balances as part of them.

OpenMP

Sort: (Result from previous assignment)

$	OPENMP TIME TAKEN SECONDS		
Problem Size\ N cores	**2**	**4**	**8**
100,000	0.008354	0.011452	0.013763
1,000,000	0.120606	0.097489	0.648612
10,000,000	1.444217	1.4416	5.825037

$	SPEED-UP OpenMP		
Problem Size \ N CORES	2	4	8
100,000	4.849413	3.537548	2.943544
1,000,000	4.222501	5.223759	0.785152
10,000,000	7.162718	7.175721	1.775872

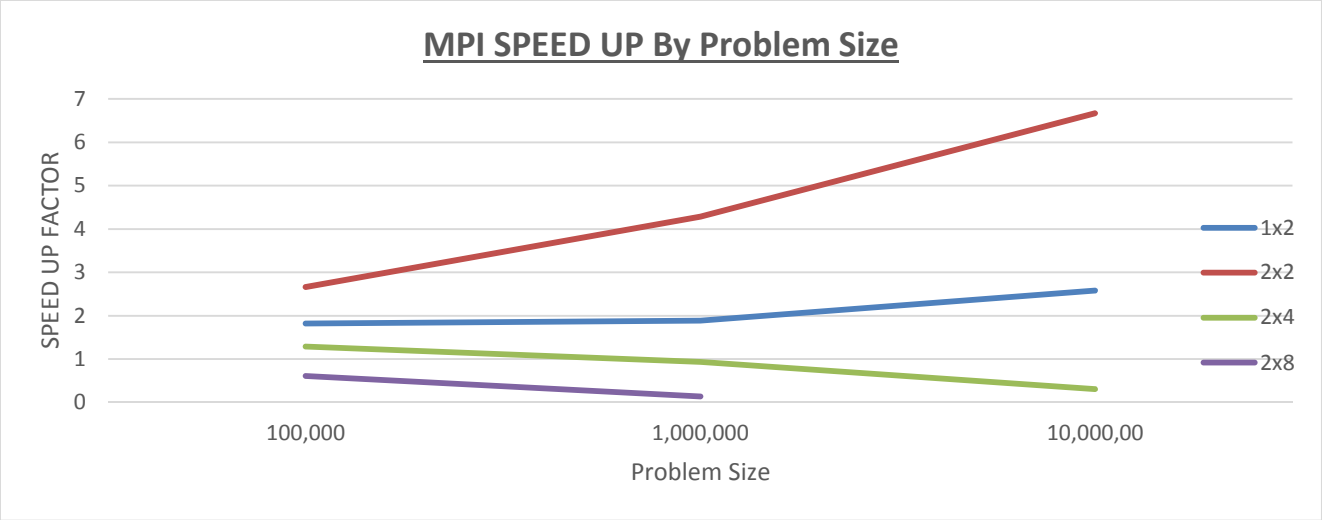


MPI

Sort: (Result from previous assignment)

$	MPI SCALING OUT (Include Time for Data Transfers)			
Problem Size \ N Cores	**1x2**	**2x2**	**2x4**	**2x8**
100,000	0.022308	0.015233	0.031467	0.066356
1,000,000	0.270464	0.118943	0.547569	3.788662
10,000,00	4.006131	1.551939	33.41657	**DATA NOT AVAIL**

$	SPEED - UP - SCALE OUT			
Problem Size \ N Cores	**1x2**	**2x2**	**2x4**	**2x8**
100,000	1.816030124	2.659489267	1.287443989	0.610525047
1,000,000	1.882908631	4.281538216	0.930036215	0.134416583
10,000,00	2.582171926	6.665544844	0.309562561	**DATA NOT AVAIL**



Mahout

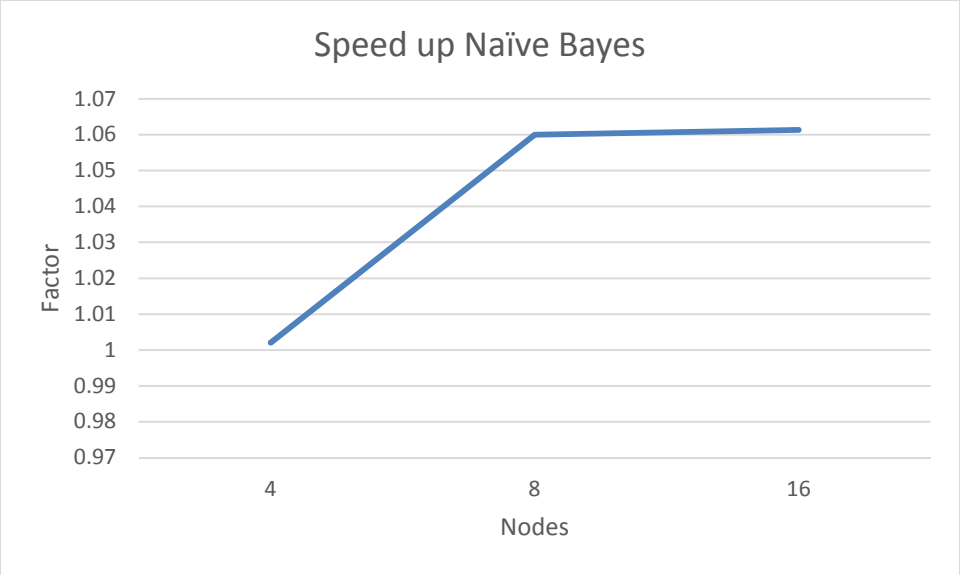
Naïve Bayes:

Accuracy: In classification of test set vs % of data chosen for training

% Training	Accuracy on Test in %
60	71.8391
75	73.3945
80	75.8621

Best accuracy is for training size of 80% of the data and a test set of 20% of the data. Having set these parameters I ran Naïve Bayes on 4, 8 and 16 nodes and here are the results:

Naïve Bayes in Minutes		
Nodes	**Time**	**Speed up**
2	0.311066667	
4	0.310433333	1.002040159
8	0.29345	1.060032941
16	0.293083333	1.061359113



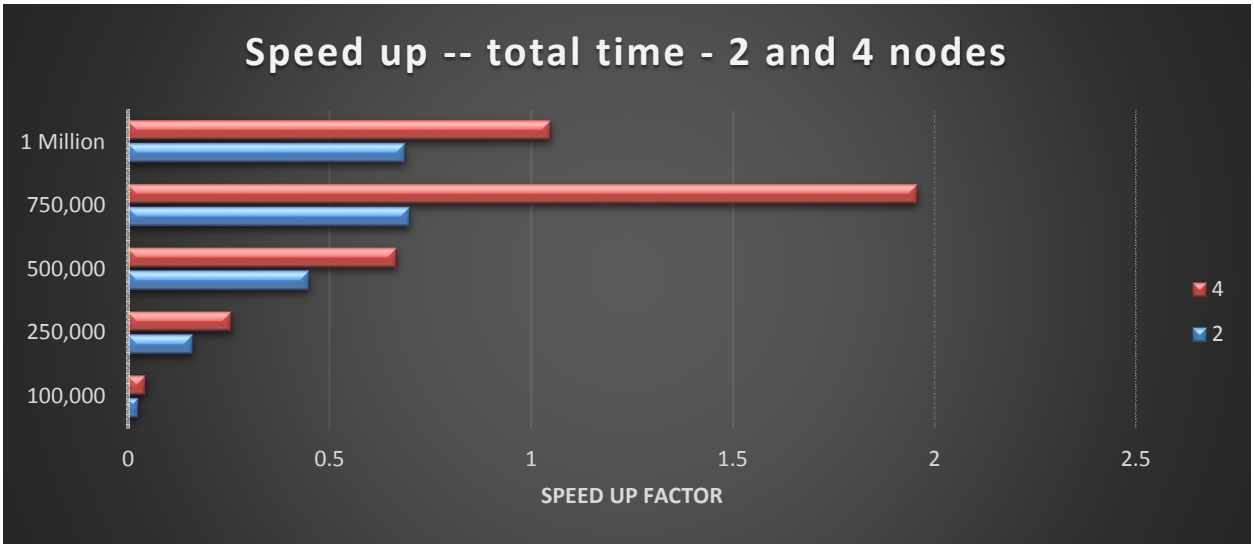
Illustrates the speed up factor between consecutive increases in number of cores. Considerable jump for 8 and then plateauing at 16.

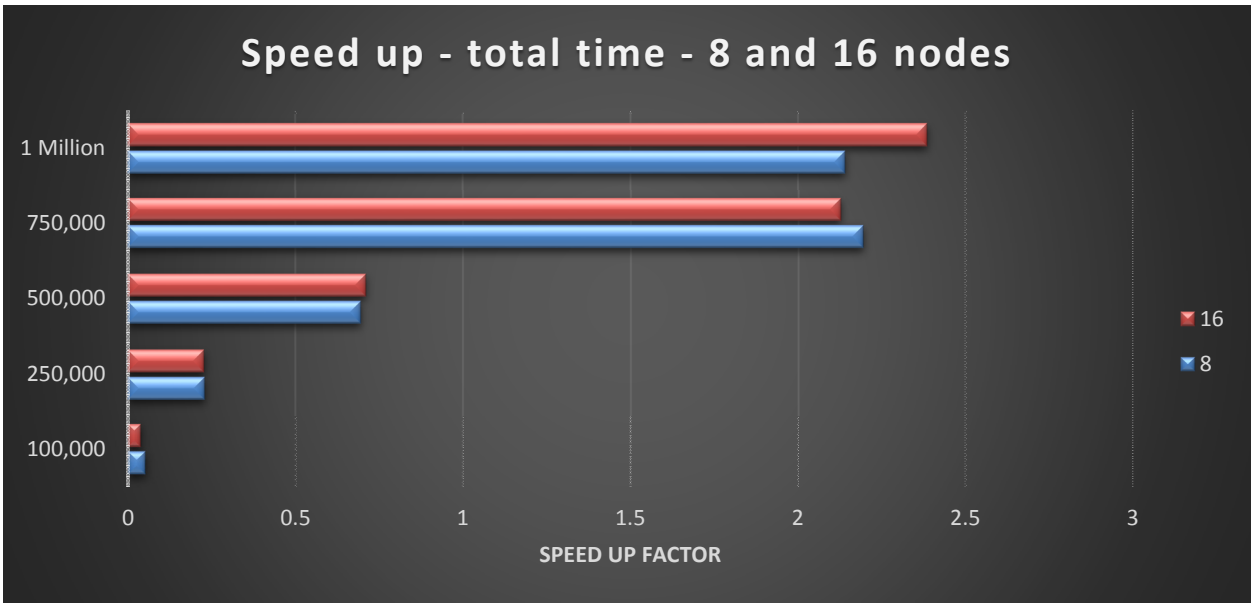
K-means using Mahout

Kmeans is a clustering algorithm. And performing classification using that and finding accuracy would be difficult. I have run K-means and got the result for the same. However there are not accuracy results, just the different clusters. Check my submission folder for the results that I have got.

Discussion of Results

Hadoop must be used for big-date and problems that require data to be taken to the processors. We see as stated in the observations that:





Note:

- Speed up whenever I notice is minimal
- Hadoop with more cores preforms better that less number of cores – as expected
- Speed-up factor greater when problem size increases
- 8 and 16 node runs are better but only at big data levels

By the theory learnt one should expect greater results with more core being thrown at a fixed problem size, but that is not the case because of the inherent nature of Hadoop. Requires time to initialize and process to the problem. Starting mapper and reducers a major problem. And also note that reducers can only start after the mappers have completed their tasks.

Other technologies such as MPI and GPU do a better job at speed up than Hadoop with reference to the problem given to me.