DIT065 Assignment 4

Group 20: Patrick Andersson, David Fagrell, Tim Zetterquist

Problem 1

Running k-means with MRJob means that the master node reads the data file and the centroid file. The centroid coordinates are saved as attributes using mapper_init so that each mapper doesn't have to read the centroid file. The master node splits up the data file between the nodes so that each mapper only has access to a data chunk. The splitting depends on the file size and we have little control over this, so the bigger the file is, the greater the advantage of using multiprocessing since there is communication that needs to be done between the master node, the mappers, and the reducers.

Each mapper calculates the closest centroid for each data point, one by one, and yields the centroid id as well as the coordinates for that point. The mapper outputs are shuffled so that all data points which belong to some cluster id get passed into a reducer.

The number of reducers is based on the number of keys, so in our code, there are three reducers that have access to only the data points which belong to its cluster. Once the new coordinates for every centroid are calculated, this information is passed back to the master node which writes them to a file. If we were to run more iterations, the master node would read and write over the original centroid file, which would then be passed into the program for every iteration.

Some advantage of using MRJob is the flexibility if the dataset would increase in size since this is done by the framework and needs little modification from the user. This can naturally be seen as a disadvantage too since we have little control over what happens in the program. Another possible limiting factor is that there is a lot of communication between nodes which might cause overhead complexity if the data isn't big enough. Also, writing and reading to files for every iteration can decrease efficiency, but this might be mitigated if we were to save the updated centroids into self.centroids for every iteration, but with the current solution, this isn't the most time-efficient way.

Problem 2a

Mean	std	min	max
5.1417	1.1547	3.1416	7.1416

Problem 2b

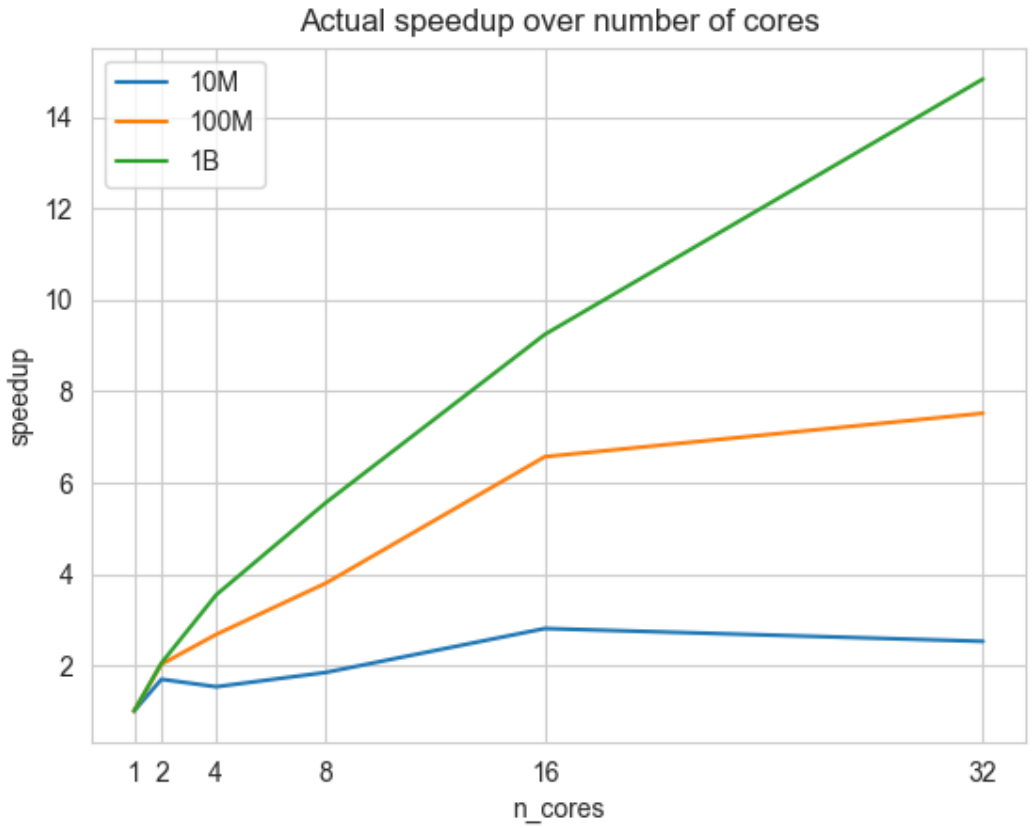

Table with total running time in seconds:

Cores / File-size	**10M**	**100M**	**1B**
1	48.24	472.04	4620.26
2	28.45	233.02	2250.70
4	31.49	176.28	1303.18
8	26.14	124.22	830.60
16	17.18	71.85	499.60
32	19.09	62.76	311.32

Problem 2c

Bin index and count: [(0, 9999637), (1, 9996417), (2, 9996174), (3, 9998175), (4, 10004386), (5, 10003418), (6, 9999031), (7, 10004021), (8, 9997405), (9, 10001336)]
Bin edges: [3.141593, 3.5415929999999998, 3.941593, 4.341593, 4.741593, 5.141593, 5.541593000000001, 5.941593, 6.341593, 6.741593, 7.141593]

Problem 2d

Median: 5.141801368779552