DIT065 - Computational Techniques for Large-scale Data
Assignment 3

Huw Fryer
Cem Mert Dalli
Stefano Ribes

May 3, 2022

1 Problem 1

As already mentioned in the previous assignment, we are now implementing a parallel version of the K-means algorithm.

A pool of workers is first created before starting looping over the refinement iterations. In order to call and synchronize the parallel workers, we relied on the `multiprocessing.starmap` function. The centroid center indexes `c` are instead stored in *shared* integer array which do *not* utilize an access lock on it. As explained in the previous assignment, each worker is launched to process a specific, non-overlapping, section of the shared array. Hence, there are no conflicts and no need for mutually exclusive accesses on the array.

Figure 1 shows the theoretical speedup of the algorithm versus the measured one. The results were collected on a dataset of 10,000 samples with 4 classes and 4 clusters.

As calculated in Assignment 2, the serial part of the algorithm is negligible. Therefore, for the theoretical evaluation, we assumed the fraction of the parallel part of the code to be 0.999.

Despite the increasing number of workers, the measured speedup is lower than the theoretical one. In fact, by looking at the case of $w = 1$, we can notice that the thread synchronization can already account for roughly 10% of the parallel implementation. This might therefore indicate that thread synchronisation takes a considerable amount of execution time.

2 Problem 2

2.1 Problem 2a

The code for problem 2a is included in file `problem2a.py`. Table 1 reports the descriptive statistics of the files with 1, 10, 100 million and 1 billion rows.

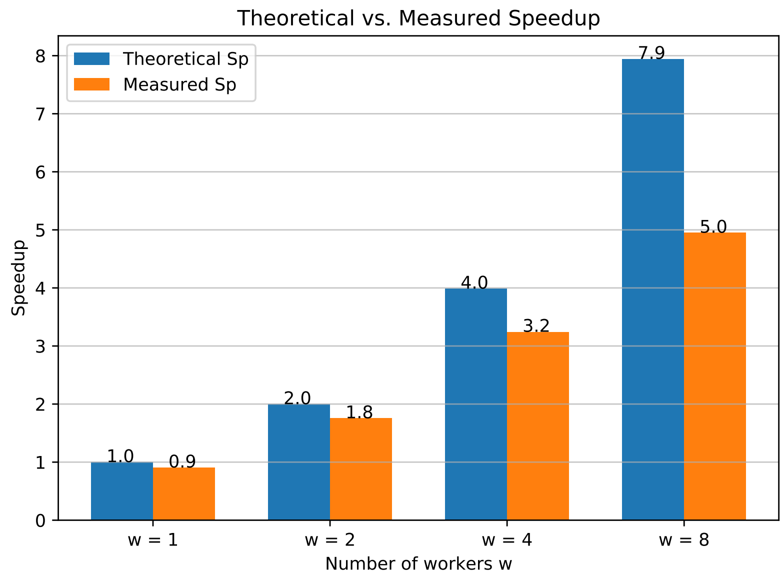

Figure 1: Theoretical versus measured speedup when running K-means with 4 clusters and 10,000 samples.

	1 million rows	**10 million rows**	**100 million rows**	**1 billion rows**
Min	3.141593	3.141593	3.141593	3.141593
Max	7.141593	7.141593	7.141593	7.141593
Mean	5.141801	5.141536	5.141704	5.141608
Std dev.	1.155144	1.154485	1.154656	1.154704
Bin 0 count	99919	999549	9999637	99996208
Bin 1 count	100115	999724	9996417	99997931
Bin 2 count	99842	999833	9996144	100009579
Bin 3 count	99955	999720	9998205	99988301
Bin 4 count	99959	1001143	10004386	100010791
Bin 5 count	99939	1001608	10003418	99998690
Bin 6 count	100055	999454	9999031	99985568
Bin 7 count	99822	999483	10004000	100005794
Bin 8 count	100041	1000161	9997402	100000609
Bin 9 count	100353	999324	10001360	100006529

Table 1: Descriptive statistics for one and 10 million rows files.

The following are the boundaries of the 10 bins for the 1M rows file and do not significantly differ from the 10M, 100M and 1B files:

3.14, 3.54, 3.94, 4.34, 4.74, 5.14, 5.54, 5.94, 6.34, 6.74, 7.14.

2.2 Problem 2b

Figure 2 shows the relative speedup of generating the statistics when varying the number of cores. The x-axis shows the log_2 of the running number of cores.

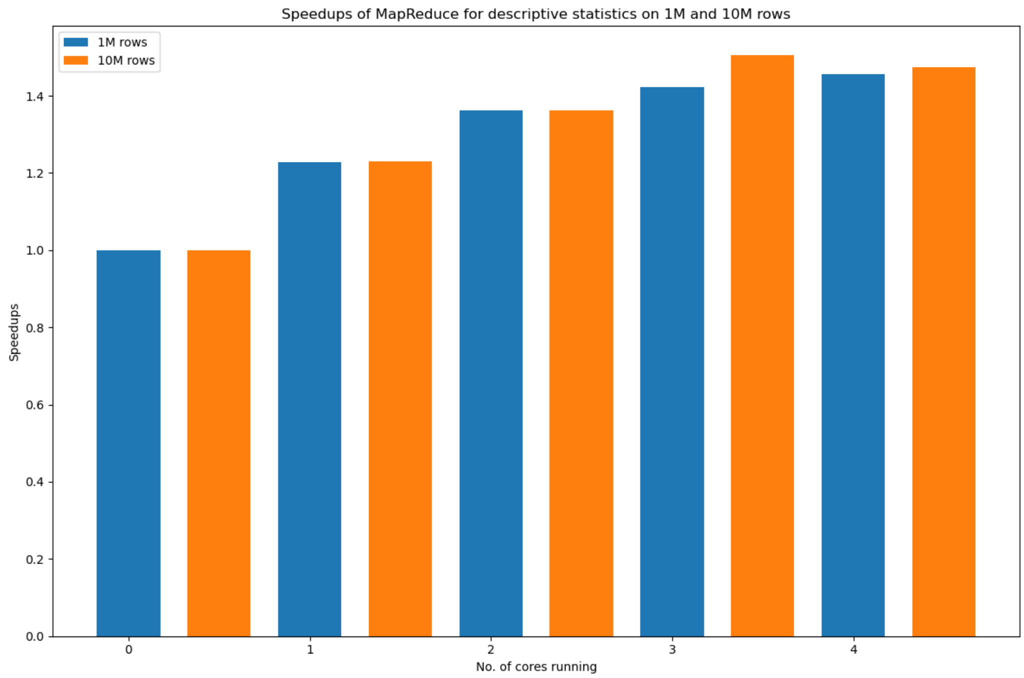

Figure 2: The speedups of the algorithm running on 1M and 10M rows data. The x-axis shows the log_2 of the running number of cores (*e.g.* label 3 corresponds to 8 cores).

2.3 Problem 2c

Median values in data files having 1, 10, 100 million and 1 billion rows are presented in Table 2. The table also reports the execution time of the aforementioned calculation, which was performed on Bayes exploiting 32 cores.

Data size	**Median**	**Time [s]**
1 million	5.1424655	5.88
10 million	5.141607	58.35
100 million	5.141801	544.39
1 billion	5.141581	5471.51

Table 2: Median values and execution time for the provided data files.