BIG DATA COMPUTING 2019-20 – HOMEWORK 3

(USE ONLY 1 PAGE)

1. **Required tests.** Do the tests with the parameters indicated in the first 4 columns of the following table. Fill the table reporting, for each test, the following values: **Init** = *time to read the input and create the RDD*; **Ti** = *time of Round i of runMapReduce* (i=1,2), **AvgDist** = *average distance among the solution points*. **Times should be in ms.** If you notice anomalies in the values of a test try to repeat the test 3 times and take the median values.

   Some anomalies are to be expected!

Table to be filled by Java users

Dataset	K	L	num-executors	Init	T1	T2	AvgDist
Glove2M300d	100	16	16	52654	36032	19751	29.040733847821866
Glove2M300d	100	16	8	56913	73994	19196	29.040733847821866
Glove2M300d	100	16	4	83418	122290	20062	29.040733847821866
Glove2M300d	100	4	4	78319	128517	1078	29.066280389112293

2. **Observations:** *write here a brief comparison among the results of the experiments reported in the table (and perhaps of other experiments that you did on your own), and try to justify the differences that you may have observed.*

We can notice that the first 3 configurations result in the exact same AvgDist value. This is because the same number of partitions (**L** = 16) and extracted points (**K** = 100) are used, so the coreset contains **K*****L** = 1600 items. In the latest configuration, *coreset* is a smaller sample containing only **K*****L** = 400 items. Since the number of samples is ¼ of the samples available in the first 3 configurations, it's reasonable to expect a slightly different AvgDist result. The initialization time tends to decrease as **num-executors** increases. It is also slightly affected by the number of partitions.

In Round 1, each worker collects approximately (**N** / **L**) samples and returns **K** selected centers. On average, every entry in the Glove dataset is ~2.445 MB. With the 4th configuration, in Round 1 we gather about 549.004 samples (~1.25GB) and we return 100 selected centers (~244.5 MB). Since executors run on a core with 2 GB of RAM, we believe our implementation of *kCenterMPD* is more space-efficient than expected. In fact, we aren't able to observe an *OutOfMemoryError* unless **L** <= 2. Another optimization we did is avoiding useless further repartitions setting the *preservesPartitioning* flag to true.

If **num-executors** < **L**, some executors will run Round 1 more than once, each time with a different partition. In fact, we notice that T1 with the 1st configuration is the minimum amongst the other T1 times, and it is about double the time required by the 2nd configuration (where num-executors is exactly halved). Similar reasoning can be applied to the 3rd configuration. Of course, the relationship between Round 1's time and num-executor isn't exactly linear, as can be expected in a distributed system.

Round 2 is executed on a single worker, no matter the value of **num-executors**. If **K** and **L** are fixed, collecting **K*****L** points and running *runSequential* on them takes about the same time. The 4th configuration contains ¼ elements w.r.t. the other 3 configurations. That's why the 4th configuration's T2 time is the fastest.