1. What was wrong with the original wordcount-5 data set that made repartitioning worth it? Why did the program run faster after?

As there were 8 files in wordcount-5 data, the 8 executors each got assigned with one file to process. However, some of the files were larger than the others, so their corresponding executors took more time to process them. Meanwhile, the other executors that finished the job early were not utilized. By doing repartition, each partition has similar data size, which helps to balance the workload of each executor and increase the parallelism of processing data.

1. The same fix does **not** make this code run faster on the wordcount-3 data set. (It may be slightly slower?) Why? [For once, the answer is not “the data set is too small”.]

Because the data size of wordcount-3 is relatively small compared to wordcount-5, and each file has similar data size.

1. How could you modify the wordcount-5 input so that the word count code can process it and get the same results as fast as possible? (It's possible to get about another minute off the running time.)

I would split wordcount-5 into smaller files.

1. When experimenting with the number of partitions while estimating Euler's constant, you likely didn't see much difference for a range of values, and chose the final value in your code somewhere in that range. What range of partitions numbers was “good” (on the desktop/laptop where you were testing)?

4~400

10: 11.70s user 1.73s system 4% cpu 4:40.08 total

20: 20.26s user 2.99s system 8% cpu 4:34.31 total

50: 19.35s user 2.81s system 8% cpu 4:22.58 total

80: 16.46s user 2.49s system 7% cpu 4:12.58 total

100: 19.35s user 2.85s system 8% cpu 4:19.33 total

120: 21.15s user 3.25s system 9% cpu 4:09.96 total

150: 20.32s user 3.16s system 9% cpu 4:09.25 total

200: 23.32s user 3.70s system 10% cpu 4:09.24 total

1. How much overhead does it seem like Spark adds to a job? How much speedup did PyPy get over the usual Python implementation?

Spark Python with PyPy: 18.00s user 2.51s system 62% cpu 32.743 total

standard CPython implementation:

20.70s user 3.19s system 10% cpu 3:57.06 total

Non-Spark single-threaded PyPy:

49.41s user 0.10s system 99% cpu 49.527 total

Non-Spark single-threaded C:

29.18s user 0.04s system 99% cpu 29.346 total