# CS 6240: Assignment 2

**Goal**: Explore partitioners and compare combiners to the in-mapper aggregation design pattern.

This homework is to be completed individually (i.e., no teams). You have to create all deliverables yourself from scratch. In particular, it is not allowed to copy someone else's code or text and modify it. (If you use publicly available code/text, you need to cite the source in your report!)

All deliverables for this HW are due as stated on Canvas. For late submissions you will lose one percentage point per hour after the deadline. This HW is worth 100 points and accounts for 10% of your overall homework score. To encourage early work, you will receive a 5-point bonus if you submit your solution on or before the early submission deadline stated on Canvas. (Notice that your total score cannot exceed 100 points, but the extra points would compensate for any deductions.) Please submit your solution through the Canvas assignment submission system and make sure your report is a **PDF file**. Always package all your solution files, including the report, into a single ZIP file. Make sure you are using standard ZIP, i.e., the format also used by Windows archives.

You have to submit the assignment by 19th October which is 1 week from now. Of course, the earlier you work on this, the better.

## Extending Word Count (Complete NoCombiner and SiCombiner in Week1)

We continue working with the same data set from HW 1 and the word count example code from the Hadoop distribution.

**Step 1**: Modify the Word Count program so that it only counts "real" words, i.e., strings that start with a letter of the English alphabet. In fact, we only want to count real words starting with letters "m", "n", "o", "p", and "q" (no matter if they are capitalized or not). For instance, "$100" and "1word" should not be counted because they do not start with a letter. Words "honey" and "User" should not be counted because their first letter is "out of range". On the other hand, "pill" and "NEU4U" should be counted.

**Step 2**: Add a custom partitioner to this Word Count program. It should assign words starting with "M" or "m" to reduce task 0, those starting with "N" or "n" to task 1, and so on.

**Step 3**: Using this modified Word Count program (only counts strings starting with the right letter, uses custom partitioner), compare the performance without combiner, with combiner, with Map-local aggregation, and with the in-mapper aggregation design pattern. To do so, we need four different versions of the modified Word Count program:

1. NoCombiner: Go through the Word Count source code and locate the command that sets the combiner class. Make sure you disable the combiner, i.e., the NoCombiner program should never do any aggregation in the mapper. (Find out how to disable the combiner.)
2. SiCombiner: This should be identical to the original source code that came with the Hadoop distribution. To make sure, verify that the combiner class is set.
3. PerMapTally: Make sure the combiner is disabled, then modify the Map *function* as follows:
    a. Inside the Map function, create a hashmap (or similar) data structure.
    b. For every relevant "real" word the tokenizer finds in the input "value", increase the corresponding word counter in the hashmap.
    c. After the entire "value" is processed, emit all words and their counts from the hashmap.
4. PerTaskTally: This is the design pattern we discussed in class. Make sure the combiner is disabled, then modify the Map *class* and create the functions for initializing the hashmap data structure and for cleaning it up after all Map calls are completed.

Note: Do not modify the input type of the mapper class and Map function.

# Report

Write a brief report about your findings, using the following structure.

## Header

This should provide information like class number, HW number, and your name.

## Source Code (44 points)

For each of the four programs, show the source code, except for the block of import statements. Highlight the lines in the code that differ between the four versions. Make sure your custom partitioner and the functionality for identifying the relevant "real" words are clearly visible and well commented.

Explain what the input "key" and input "value" of the provided Word Count's Map function are and state how you found this answer. Your answer should be something like this: *The key is the name of the file and the value is the actual document in that file. I looked this up in the Hadoop 1.2.1 API where it says "…" about the Text type.* (This is of course NOT necessarily the correct answer!) Hint: You can look in the Hadoop API for information about Hadoop's Text type. You can search in the textbooks or on the Web for more information. Or you can use the debugging perspective to watch the actual data as you are stepping through the execution. We recommend you do the latter anyway. It is easy and fun. And you can verify if whatever information you found in the textbook or on the web is correct.

## Performance Comparison (40 points total)

Run all four programs in Elastic MapReduce (EMR) on the large text data set from HW 1. Use the following parameters:

- Configuration 1: 6 small machines (we want 1 master and 5 workers)
- Configuration 2: 11 small machines (1 master, 10 workers)

Report the total running time of each program execution. (Find out how to get the running time from a log file. It does not have to be down to a tenth of a second.) Repeat the time measurements one more time for each program, each time starting the program from scratch. Report all 4 programs * 2 configurations *2 independent runs = 16 running times you measured. (16 points)

Look at the syslog file. It tells you about the number of records and bytes moved around in the system. Try to find out what these numbers actually mean, focusing on interesting ones such as Map input records, Map output bytes, Combine input records, Reduce input records, Reduce input groups, Combine output records, Map output records. Based on this information, explain as much as you can the following, backing up your answer with facts/numbers from the log files: (4 points each)

- Do you believe the combiner was called at all in program SiCombiner?
- What difference did the use of a combiner make in SiCombiner compared to NoCombiner?
- Was the local aggregation effective in PerMapTally compared to NoCombiner?
- What differences do you see between PerMapTally and PerTaskTally? Try to explain the reasons.
- Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.
- NEW: Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.

## Deliverables

1. The report as discussed above. (1 PDF file)
2. The source code for each version of the Word Count program. (4 java files) (4 points)
3. The syslog files for one successful run for each program for Configuration 1. (4 plain text files) (4 points)
4. All output files produced by that same one successful run for each program (4 sets of several part-r-… files) (8 points)