CSE 687 Object Oriented Design Project

MapReduce

Phase #3

# Background

# In Stage 3, you will be extending your Project 2 solution:

# Create multiple mappers and multiple reducer processes.

# Create a partition function that splits keys into R buckets (where R is the number of reducer processes). Each mapper will generate R output files corresponding to the number of reducers using the partition function.

# The Reducers can be created once all of the mappers are complete.

# The directories and bucket/file naming convention can be configurable.

# Your command line executable (controller) will:

# Assign the files in a reasonable way amongst the map processes.

# Wait for all map processes to complete successfully.

# Create reducer processes.

# Wait for all reducer processes to complete successfully.

# Write the SUCCESS file indicating a successful run.

# Notice that the Sorting and grouping should happen as part of the reducer process.

# The mapper process and reducer process are separate entities from the DLLs. Although clearly, the mapper process will need to run the mapper function from the user supplied DLL, same goes for the reducer.

# You may either have a single executable that can do either mapper or reducer OR you may choose to split these into separate executable projects. Either is fine.

# No network communication is required for this project. The controller can get the exit codes from the mappers & reducers to measure success.

# Methodology Requirements

1. You must work in a team of 2. If there is an odd number of students, there will be a single team of 3. No one will be allowed to work independently.
2. You must use C++ in conjunction with MS Visual Studio and GitHub (student account).
3. You may use anything in the std library and additionally may use anything in the BOOST (https://www.boost.org/) library.
4. All changes must go through a code review by your partner(s).
5. All partners must submit a reasonably equal number of check-ins for each project, as evidenced by git submissions.
6. All partners must treat each other with respect.

# Technical Requirements

1. Code must have unit tests above 95% line coverage. You may use, Boost unit testing, Google unit tests, or your own unit testing.
2. Code formatting & comments should follow Google style guide: <https://google.github.io/styleguide/cppguide.html>
3. Errors, warnings, information, fatal errors, etc., will be logged. Boost Logging can be used (<https://www.boost.org/doc/libs/1_63_0/libs/log/doc/html/index.html>).

# Rubric

1. Project should not crash in error scenarios: 10%

Errors should be appropriately handled and made visible.

1. Code behavioral correctness: 50%
2. Code Unit testing: 10%

Code must have code coverage above 95% by line count.

1. Code Commenting & Style: 10%

Code must meet style & comment guidelines.

1. Code Organization and Structure: 10%

Code must adhere to SOLID principles.

1. Procedural Correctness: 10%

Code must have gone through code reviews. Code reviews should be sufficiently thorough. Members that do not give their team quality code-reviews will be penalized.

Additionally, each member of the team must contribute a reasonably equal amount to their project. Each member will submit a brief report detailing what percentage of work was completed by each member. I encourage teams to keep detailed meeting notes.

# Bonus

You may only attempt the bonus if the code is behaviorally correct, unit tested, correctly formatted, and reasonably structured.

10%: Each process should support a configurable level of multithreading.