MapReduce Phase #2 Update

Team: Joseph Laible, Pedro Ortiz, Vimal ramnarain

Introduction

This phase uses a client program written in C++ that calls in the Map and Reduce Dynamic-Link Libraries (DLLs) to carry out the MapReduce functionality. The code includes two DLLs, one that carries out the Map function (map_func) and the other that carries out the Reduce function (reduce), along with a client app that contains the 'fileManagement.h/cpp' and 'workflow.h/cpp'. Similar to Phase 1, the program processes a directory of Shakespeare's works in text files, performs a word count operation, and stores the intermediate and final results in user specified directories.

System Architecture and Component Functionality updates

The heart of the system's architecture is now the workflow component (incorporating the phase 1 executive utility into workflow). Workflow still maintains the system road map of phase 1, maintaining the same phase transitions (file paths, input, map, temp output, reduce, output). Workflow handles command line inputs, DLL integration, validating and parsing the paths of input, temporary storage, output, and orchestrates the overall execution of the MapReduce process. The Workflow component's execute() method manages the sequence of operations and creates the necessary connections to utilize the DLLs, and their included functions. The incorporation of DLLs replaces the standalone Map and Reduce classes of phase 1.

The Map DLL contains the map_func method, the map_func method is used to process the input data and output the processed data to a new temporary file. The map_func method consolidates the methods MapFunction and ExportFunction from phase 1 into one, still handling the tokenization and processing of text, as well as writing

the output to a temporary directory. The map_func method still writes these pairs into the temporary file in batches via the same buffer utilized in phase 1.

The Reduce DLL contains the reduce method. The reduce method, as in phase 1, opens and reads the input file (temporary file exported from map_funct), then aggregates the counts of each unique word, and concludes by writing the updated word count pairs to the output file.

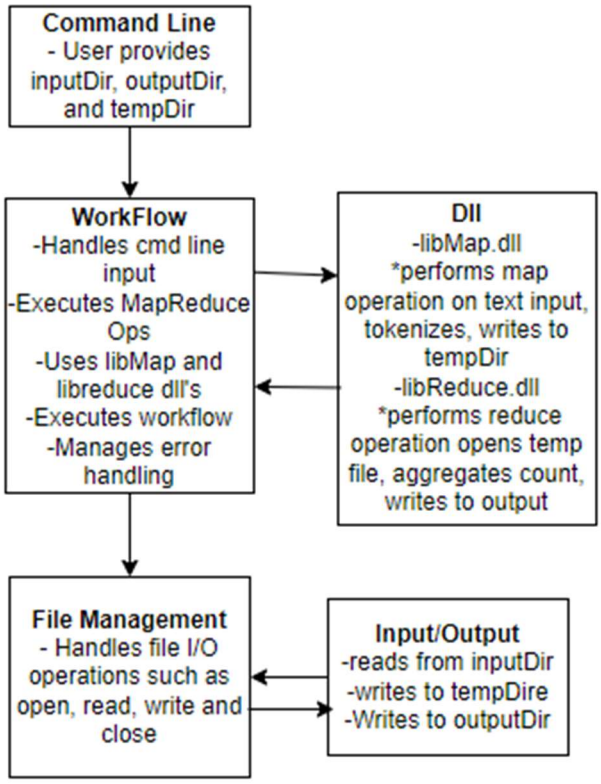

Figure 1: Phase 2 System Diagram for MapReduce Program

Conclusion

The inclusion of DLLs allows for greater re-usability of the code, as well as increasing the efficiency of the system. This update builds on our successes from phase 1, keeping the functionality of the file management class, removing, and incorporating the executive utility into workflow, and replacing the Map and Reduce classes with their respective DLLs. The updated architectural approach maintains the phase 1 vision of a scalable framework to be built upon, incorporating the system advantages of DLLs.