CSE 687 Object Oriented Design Project MapReduce Phase #1

Team: Joseph Laible, Pedro Ortiz, Vimal ramnarain

Abstract

This paper details a standalone MapReduce system designed by our team for efficient analysis of textual data, specifically applying it to the works of Shakespeare as provided for our project. This system operates within a single-process architecture and utilizes a command line interface to manage the flow of data from input through processing to output.

Introduction

The architecture of this MapReduce system is designed to effectively process extensive text datasets. Utilizing a single-threaded process, the system integrates various components, each assigned specific functions within the data handling and processing lifecycle. This design ensures that the system operates efficiently and effectively, adhering to the constraints of object-oriented design principles. The development of this system incorporates the four fundamental principles of object-oriented programming—encapsulation, abstraction, inheritance, and polymorphism each playing a crucial role in ensuring the system's modularity and scalability.

System Architecture and Component Functionality

At the heart of the system's architecture is the Executive component, which acts as the system's entry point. It handles command-line inputs, validating and parsing the

paths for input files, temporary storage, and output directories. This setup is crucial for establishing the environment in which the MapReduce process will operate.

Following this, the Workflow component orchestrates the overall execution of the MapReduce process. It is responsible for initiating the mapping and reducing phases, ensuring orderly data processing. The Workflow component's execute() method manages the sequence of operations, ensuring smooth and efficient phase transitions.

Supporting the system's operations, the File Management utility abstracts the complexities of file interactions. It provides robust methods for opening (openFile), reading (readNextBlock), writing (writeFile), and closing (closeFile) files. It also manages file deletions and checks for directory existence, which are critical for processing the data files.

Central to the data processing operation is the MapClass, tasked with the initial data processing phase. This component uses methods to start (start), process data (MapFunction), export processed data (ExportFunction), and conclude the mapping phase (end). The MapFunction manages the tokenization and normalization of text, converting it to lowercase, removing punctuation, and pairing each word with an initial count. The ExportFunction then writes these pairs to a temporary file in batches.

Following the MapClass, the Reduce class aggregates the intermediate data to compile the final results. It starts with the start() method, processes the aggregated data through the reduce() method, and concludes with the end() method. This phase aggregates occurrences of each word and writes the final counts to the output directory, marked by the creation of a "SUCCESS" file to indicate successful processing.

Detailed Process Description

The system's functionality is encapsulated within its components' methods. In the MapClass, the MapFunction method processes text by breaking it into words, normalizing these words, and pairing them with initial counts. The ExportFunction efficiently buffers and writes this data to disk. This batch processing method minimizes disk writes, enhancing the system's efficiency.In the Reduce class, the reduce method aggregates the counts of each unique word, compiling these into final results, which are then written to the output directory. The management of file operations by the File Management utility ensures data integrity and system efficiency throughout the process.

Conclusion

The MapReduce system's design provides an efficient method for processing textual data, as demonstrated by the processing of Shakespeare's works. Each component is specifically designed to perform integral roles within the workflow, ensuring effective and efficient data processing. This architectural approach supports the current project requirements and offers a scalable framework for future phases.