Yifei Gao(yg578), Yunhao Hu(yh2233)

Introduction

This report introduces the implementation details of a MapReduce project designed to perform word counting tasks on a set of documents. The system architecture comprises a master server and worker nodes operating in a single-machine environment.

System Architecture

The architecture of the MapReduce system is divided into a master server and worker nodes:

- Master: Orchestrates the entire MapReduce process, including task distribution, synchronization, and final result aggregation.
- Workers: Execute map and reduce tasks as delegated by the master. All worker nodes run concurrently on the same machine, sharing the same filesystem.

Each worker is implemented as a separate thread. The master creates a fixed number of worker threads at initialization. These threads remain active and listen for tasks until the master sets a termination flag, after which all workers are joined back before the master exits. These workers will first execute the map tasks, wait for all the map tasks to be completed, and then perform the reduce tasks, with the master ultimately executing the combine task.

Communication and Task Scheduling

The communication between the master node and worker threads is handled using a combination of mutexes, condition variables, and a task queue. This design allows for effective communication and synchronization among threads.

- A shared queue managed by the master stores tasks that need to be executed. This queue is protected by a mutex to prevent simultaneous access by multiple threads.
- The master uses a condition variable to signal worker threads about the availability of new tasks or changes in the task queue's state.
- Workers continuously monitor the task queue. Upon waking and finding a task, a worker dequeues it, processes it, and then signals the completion. This dynamic assignment helps in balancing the load among workers.

Map Task

Each map task processes an input file to produce key-value pairs where the key is a word and the value is its count. These pairs are distributed across multiple intermediate files based on a hash function to ensure an even distribution among reduce tasks. This is handled by opening a file stream per reduce task in the output directory.

Reduce Task

Each reduce tasks reads all intermediate files designated for it, aggregates word counts, and outputs a single file with the total count of each word.

Merge Operation

After all reduce tasks are completed, the master executes a merge operation, combining all outputs from the reduce tasks into a single sorted output file.