Question 2 Report

Kshitij Nigam
2023202031

1. Broad Overview of the System

This system implements a distributed MapReduce framework using gRPC for communication between clients and workers. It supports two key MapReduce jobs: **Word Count** and **Inverted Index**, both of which are common applications of the MapReduce model. The system consists of a master server that coordinates the distribution of tasks to worker nodes, which process the tasks in parallel. The tasks are divided into two stages: Map and Reduce. The system is designed to efficiently process large datasets by distributing the workload across multiple worker nodes.

The primary components of the system include:

- **Master Node**: It assigns map and reduce tasks to workers and manages the execution flow of these tasks.
- **Worker Nodes**: They execute the map and reduce tasks. For map tasks, they read input files, process them, and generate intermediate results. For reduce tasks, they aggregate intermediate results and write the final output.
- **gRPC**: The system uses gRPC to enable communication between the master and workers. gRPC handles the remote procedure calls (RPCs) between the client (master) and server (worker).
- **Task Types**: The system supports two types of tasks—map and reduce—each performing specific actions based on the job type, which could either be word count or inverted index generation.

2. Design Choices for Each Requirement

1. **Task Distribution**:
    - The system uses a master-worker model for task distribution. The master node assigns tasks to workers by choosing the one with the least load (based on a priority queue). This ensures a balanced distribution of tasks and prevents overloading any worker.
    - **gRPC** is used for communication, ensuring efficient, strongly-typed communication between the components. Each worker exposes an RPC service with methods for executing tasks and shutting down.
2. **Map Task**:
    - The map task is responsible for reading the input file, processing the contents (extracting words), and writing the intermediate results (either word counts or inverted index entries) to multiple output files.
    - The number of output files is determined by the number of reduce tasks (nReducer), and the map output is partitioned based on a hash of the key (word). This ensures that related data (words) are sent to the same reducer.

- The system handles different job types (word count or inverted index) by writing the corresponding output format (e.g., (word, count) or (word, filename)).

3. **Reduce Task**:
    - The reduce task is responsible for reading the intermediate output files from the map tasks, aggregating the results (word counts or filenames), and writing the final output to a new file.
    - For word count, the reducer aggregates counts for the same word. For inverted index, it aggregates the filenames where the word appears.
4. **Concurrency and Parallelism**:
    - The system leverages multiple worker threads for parallelism, ensuring that map and reduce tasks can be executed concurrently.
    - The workers are managed by a worker manager, which uses a priority queue to allocate tasks to the least loaded workers, ensuring fair distribution and load balancing.
5. **Worker Failure**:
    - The system assumes that worker failures are not handled, as specified in the requirements. The design of the worker manager, however, can be easily extended to handle failures by retrying failed tasks or redistributing the tasks.
6. **gRPC Protocol**:
    - The system uses Protocol Buffers (protobuf) to define the service interface, including the task requests and responses. The TaskRequest message defines the necessary parameters for map and reduce tasks, and the TaskResponse message indicates the success or failure of a task.

3. Assumptions and Future Work

- **Assumptions**:
    1. **Worker Failures**: The system assumes no worker failures as per the problem statement. However, it can be extended to handle retries or reassign tasks if a worker fails.
    2. **Input Format**: The input files are assumed to be well-formed text files, and the system expects only alphanumeric words in these files. Malformed lines are handled with warnings, but they don't halt execution.
    3. **Job Types**: The two job types (word count and inverted index) are the primary jobs supported. Other types of MapReduce jobs can be integrated in the future.
- **Future Work**:
    1. **Worker Failure Handling**: Adding fault tolerance mechanisms, such as reassigning tasks if workers fail, would make the system more resilient.
    2. **Dynamic Scaling**: The system could be enhanced to dynamically add or remove worker nodes during execution based on workload demands.
    3. **Optimization**: Improving the partitioning logic for map outputs to optimize the distribution of tasks across reducers could reduce the time taken for large datasets.

4. Additional Points

- The system provides scalability, as tasks are distributed across multiple workers, allowing for efficient processing of large datasets.
- The use of gRPC ensures strong communication guarantees, such as bidirectional streaming and request/response handling.
- The modular nature of the system makes it easy to add additional functionalities, such as supporting more job types or extending the system to handle worker failures.