**Design and Implementation of a MapReduce System**

In my system Master node is connected to two Worker nodes. The Master node has a Job Control module, which manages the overall MapReduce job, and a Map Result Queue, which stores the output of each Map task.

The Worker nodes each have a Map Task Queue and a Reduce Task Queue, which store the tasks assigned to them by the Master node. Additionally, each Worker node has a Reduce Input module, which stores the Key-Value pairs output by the Map tasks assigned to that node.

Data Partitioning:

The input data is partitioned into smaller chunks of equal size by the master node and each chunk is assigned to a map worker.

The number of worker threads for the map and reduce tasks is defined in the mr.py file and can be adjusted as needed.

Fault-Tolerance:

The system is designed to be fault-tolerant by implementing a dynamic membership protocol and by monitoring the health of each worker node.

When a worker node fails, the master node detects the failure through a heartbeat mechanism and reassigns the failed worker node's tasks to other available nodes.

In addition, the system uses logging to record important events for debugging and evaluation purposes.

Dynamic Membership:

The system implements a dynamic membership protocol by allowing new worker nodes to join the system and by detecting when a worker node leaves the system.

When a new worker node joins the system, it sends a registration message to the master node, which adds the new node to its list of available nodes.

When a worker node leaves the system, it sends a deregistration message to the master node, which removes the node from its list of available nodes.

Implementation Assumptions:

The system assumes that each worker node has access to the same input data.

The system assumes that the map and reduce functions are pure functions, meaning that they do not have side effects and always return the same output given the same input.

The system assumes that the input data can fit into memory on each worker node.

The system assumes that the network connections between the master and worker nodes are reliable and that messages are delivered in order.

Communication between the Master and Workers:

The master node communicates with the worker nodes through a request-response mechanism using a custom protocol.

When a worker node joins the system, it sends a registration request to the master node, which responds with a registration response containing a unique ID for the new worker node.

When a worker node is ready to receive a task, it sends a task request to the master node, which responds with a task message containing the input data and the map or reduce function to be applied.

When a worker node completes a task, it sends a task result message to the master node, which stores the result in a global data structure.

When the master node detects a worker node failure through a heartbeat mechanism, it reassigns the failed node's tasks to other available nodes and removes the failed node from its list of available nodes.