MapReduce cluster

TABLE OF CONTENTS

In this section, we will cover some of the important high-level concepts of how the system is expected to work. It will be useful to familiarize yourself with the terminology below before diving into the remainder of the spec.

Workflow

Our MapReduce cluster consists of one **coordinator** and several **workers**. The cluster can run several different **applications**, each of which is defined by a `map` function and a `reduce` function.

The general flow for running a job on the MapReduce cluster looks like this:

1. A **client** can make an `SUBMIT_JOB` RPC request to the coordinator that specifies the following parameters:
   - **Input files**: Files containing data to be processed
   - **Output directory**: Directory to write final outputs to
   - **Desired application**: Application to run on the given input files (e.g. word count)
   - **Number of reduce tasks**: Number of sets to divide mapped keys into
2. The coordinator will then assign **map tasks** to each worker. A map task involves running the application's `map` function on the data contained in a single input file. The results of these map tasks **are stored in temporary files on disk** that can be read by other workers, though in practice workers have separate disks and thus would need request data from other workers over the network when needed. There should be one map task per input file, so the number of map tasks is equal to the number of input files. Each map task is identified by a unique **map task number**.

3. Once all map tasks are complete, the coordinator will assign a **reduce task** (i.e. a set of keys to call the application's `reduce` function on) to each worker. To retrieve the results of map tasks from other workers, workers executing a reduce task must send RPCs to request the necessary data. Each reduce task is identified by a unique **reduce task number**.

4. Finally, the reduce workers will write their results to disk, after which the client can do whatever postprocessing it needs to on the MapReduce output files found in the output directory.

Fault tolerance

The MapReduce cluster should also be **fault tolerant**. This means it can tolerate worker crashes and failures, reassigning tasks to alive workers as necessary.
