Fault tolerance

TABLE OF CONTENTS

In this part, you will complete your MapReduce system by implementing fault tolerance. Specifically, you will update your coordinator to handle worker crashes and failure.

You should handle worker crashes by detecting when a worker has failed, and reassigning relevant tasks to other workers. In this system, we will consider a worker failed if it hasn't completed its assigned task within `TASK_TIMEOUT_SECS` seconds.

Worker crashes

Your coordinator should be able to determine whether a worker has died. This should be implemented by checking whether an incomplete task was assigned longer than `TASK_TIMEOUT_SECS` seconds ago.

Completed tasks should not be reassigned, since their output is stored on disk.

Since the wait times we define happen to be multiples of seconds, you may use `time.h`. An example is provided below:

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>

int main () {
   time_t start = time(NULL);
   sleep(5);
```

```
    printf("Seconds since start of program = %ld\n", time(NULL) - start);
    return 0;
}
```

MORE ACCURATE WAITING (OPTIONAL)

While we will not be testing the accuracy of your wait times, you may find the code snippet below helpful if you want to support millisecond precision for wait times. **Keep in mind that improving wait time precision is entirely optional.**

```
#include <sys/time.h>

suseconds_t micros_elapsed(struct timeval time) {
  struct timeval current_time;
  gettimeofday(&current_time, NULL);
  return (current_time.tv_sec - time.tv_sec) * 1000000 + current_time.tv_usec - time.tv_usec;
}
```

In order to use this effectively, your coordinator will also need to store when each map or reduce task is assigned. Modify your data structures to keep track of the assigned time of each task as either a `time_t` or a `struct time_val`, and update this field appropriately using the `time` or `get_time_of_day` functions.

Once you have done this, implement the logic for reassigning tasks that are not completed within `TASK_TIMEOUT_SECS` seconds.

Job failures

If an error that cannot be fixed occurs, the job should fail. That is, no more tasks for the job should be assigned, and polling the job's status with the `POLL_JOB` RPC should give `done = true` and `failed = true`.

Examples of errors that should cause a job to fail immediately include:

- Being unable to find or open an input file
- Being unable to write to an output file
- Receiving an error from an application `map` or `reduce` function

The worker side of this has already been implemented, and will set `success = false` in the `FINISH_TASK` RPC to notify the coordinator that an irrepairable error has occured.

You may have already implemented this in the previous part, but if not, modify your `FINISH_TASK` RPC to complete the coordinator's side for this functionality.

Debugging

Fault tolerance is difficult debug due to its dependence on timing. For some tips, take a look at the Testing and debugging section.

Autograder

After completing this, you should be passing all the autograder tests.
