William Sease

The problem: Multi-Master Map Reduce

One of the issues with map reduce is
that the master forms a single point of failure
- if it crashes, the operation fails

I will seek to solve this problem by implementing
multiple masters and duplication of files, ideally
without losing the performance benefits
that are the whole point of map reduce

Proposed Solution:
RAFT ?

At the most basic level, we need a backup master to take over if the original master fails

How does it know if the master fails? Heartbeats?
If we’re sending heartbeats, we might as well use them to keep up to date so the new master can pick up rather than starting over from scratch

Implementation Details

I intend to make use of the Unity engine to build the simulator itself, to avoid spending excessive time on the construction of a useable UI

The code for the various nodes and packets will be written in C#. With the simulator handling packet delivery, no additional libraries will be needed and the C# code can hopefully be written from scratch

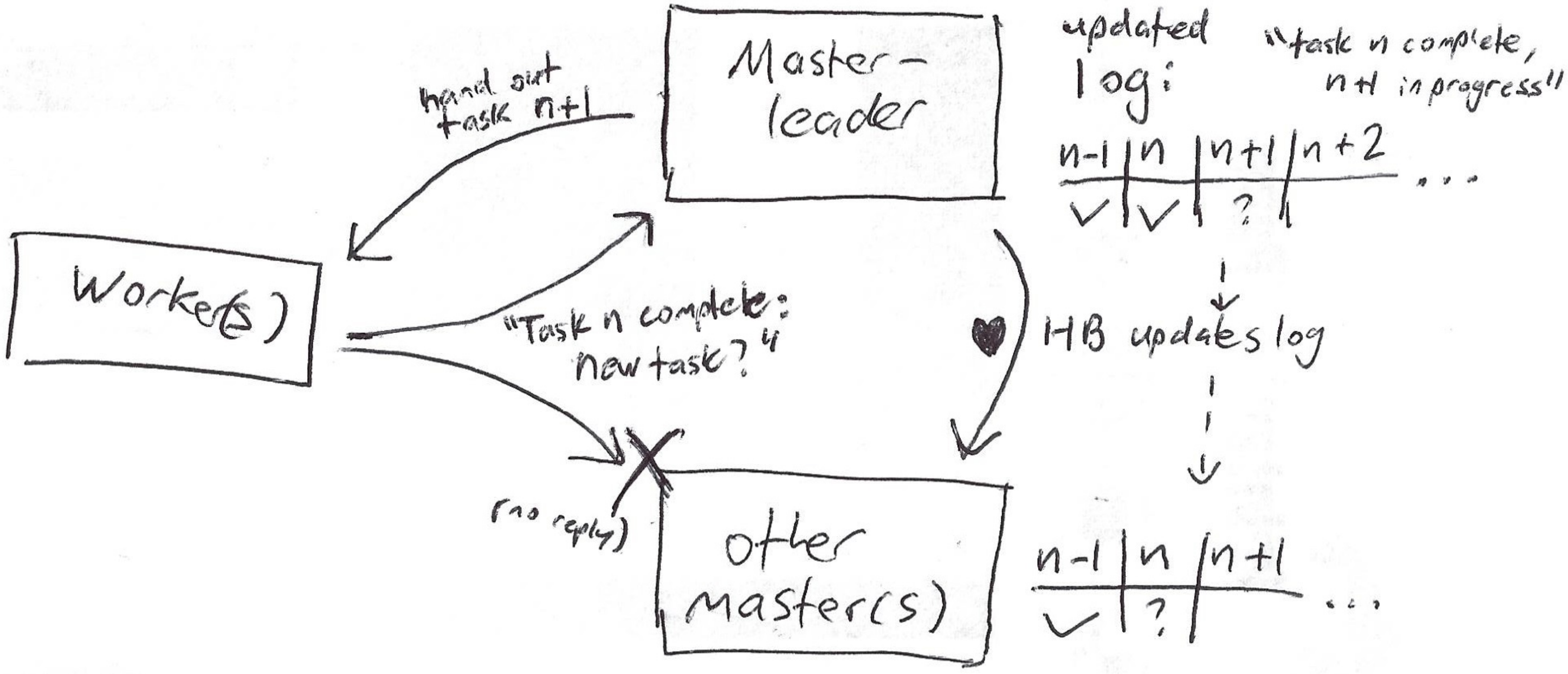
The Algorithm

The Map-Reduce part of the algorithm will be largely as is: The primary master will keep a list of which jobs are completed, in progress, or not begun yet

The primary master will regularly back this list up to the other masters (using the RAFT heartbeats)

For convenience, workers will direct their queries to ALL masters, but only the master-leader will respond

Diagram of MR structure:

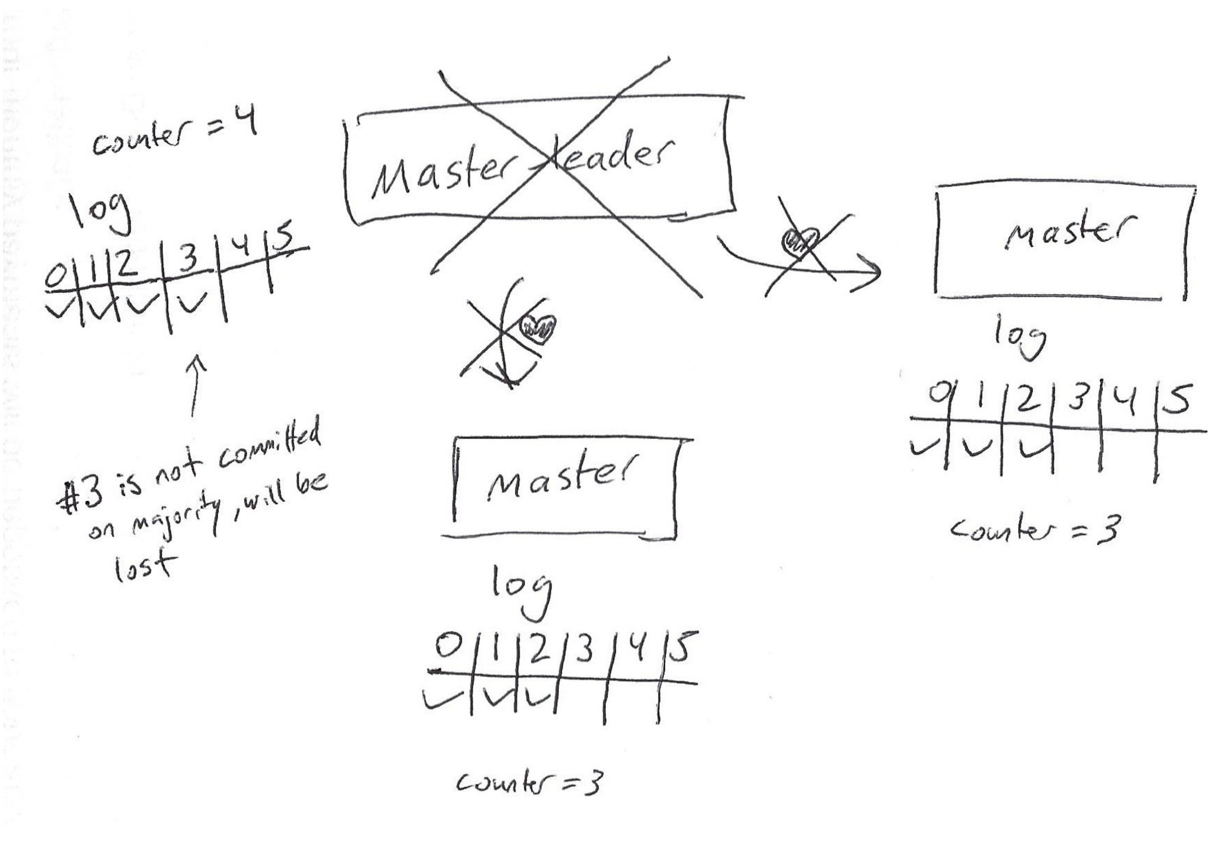


Along with a list, the master-leader will maintain a counter that goes with it to track how up-to-date this list is (counter = number of complete tasks)

This counter will be used to determine the successor in case the master-leader crashes; the backup master with the most up-to-date log will take over

During this election period no master will be answering worker queries, so there will be a short period of system downtime

The only way log data will be lost is if the master-leader crashes after receiving a notification but before sending a heartbeat:



Since the only loss of performance and (possibly) data is in the result of the master-leader becoming unresponsive, a contingency the original MR did not allow for at all, this is deemed acceptable

The evaluation will include a comparison with a single-master map reduce, and in the event of no master-leader crash my algorithm should be no worse

Alternate Possibility: Truly Distributed

The downtime during elections could be avoided if all masters respond to worker calls, and periodically update each other

There would be no leader, and since all that matters in the logs is task completion, any update would result in the logs being merged with each task being completed if it is completed in either log

This would raise the possibility of duplicated work if two workers make concurrent requests which are serviced by different masters, and are given the same task

If the masters wait to coordinate, the performance benefits of asynchronous action are lost

Hopefully this chance could be minimized by having the master hand out a random available task rather than the “next” one, or some other trick