MULTI-MASTER MAP REDUCE

William Sease (V00857661)

INTRODUCTION

(This report is intended to follow the more general introduction in the slides – explanation of the single-point-of-failure of traditional MR is not explained here).

This project functions as a combination of RAFT and MapReduce. While MapReduce operations are being performed by the workers, the masters run a RAFT operation to make sure a master-leader with correct information remains active. The RAFT itself is somewhat simplified, since we are tracking completion of a list of tasks rather than managing an order-sensitive FSM.

DATA FLOW

The client supplies the files to be operated on by MapReduce. The node that receives these files has correct information, and must therefore be the leader of RAFT. It assumes leadership and immediately sends a message to the other masters, both to reset their election timers (as a heartbeat) but also to duplicate to their local databases the files it has received and the arrays it created to track the completion of worker jobs.

The workers will request tasks from all masters simultaneously, since they don't track which masters are leader or which may have crashed. Only the leader-master responds, doling out tasks and keeping track of which have been completed. This latter information will be duplicated to all the other masters with each heartbeat, to minimize the amount of duplicated work in the case of a master crash.

It is possible that a worker completion message/new task request will receive no response if it happens to arrive during an election following a leader disconnection, when there is currently no active leader. As such, a worker will have to run a short timer and repeatedly reissue its request until it receives a response. For this reason, the masters will be required to remain online for a short time after the completion of the job and the return of the results to the client, to make sure all currently-looping workers receive an "exit" instruction.

IMPLEMENTATION

I have elected to make a visual simulator to allow hands-on testing of the algorithm by crashing / resuming or partitioning servers while the operation is running. This simulator is being created in Unity to take advantage of the UI tools provided. All code is written in C#.

In Unity, support code is written for Servers, Links, and Messages. A single coordinator keeps track of all of these, allowing an observer to reach in and strategically inflict anomalous behaviour such as crashes or dropped messages. Code for the Algorithm (code for Masters and Workers) is written separately, extending the Server code by making use of methods declared there such as SendMessage.

PROGRESS

At time of writing, the project is behind schedule. Unfortunate time overlap with the final week of a three-week reading/writing intensive course combined with difficulties encountered in Lab 2B severely reduced the hours available to work on this project. At time of writing, the simulator is functional and the master-leader sends a heartbeat to keep the other masters quiescent, but no more. The workers are as yet completely unimplemented, so the mapreduce is not performed.

Lab 2B's issues are time-consuming enough that it is unlikely I will be able to do more work before the submission date on Monday the 6th (that being the day before my lab), so this partial project is being submitted ahead of the deadline.

FILES/WORK

https://github.com/williamtsease/WilliamSease-DistributedComputingProject

The project is structured with the Master and Worker code outside of the Simulator directory, with the aim of allowing the algorithm's logic to be examined separately from the implementation of the simulator that supports it.

I didn't think to review the exact format of the submission while I was making the initial simulator code, and since I am working alone I had no need for a repository. I created one for submission but there will at this time not be a visible history of commits. As I am working alone, there will be no concern for division of work among team members and thus hopefully a history of commits is of less significance.

Since the project is in an unfinished state, it will be impossible for anyone without an installation of Unity to run it, though the code can be freely examined. As mentioned above, the Master and Worker currently have very little code to view.

A short video is also contained in the repository, demonstrating several of the simulator's capabilities:

- leader sending heartbeats
- crashing servers
- dropping / adding latency to messages