15-440: Lab 3

Spencer Barton (sebarton)
Emma Binns (ebinns)

November 18, 2014

1 Running the Framework

1.1 System Requirements

MapReduce runs on a unix cluster. The CMU unix cluster will work. Place the code in a file on AFS as we rely on that file system to share all of our code files between participants.

1.2 Set-up

The following steps must be followed for a successful set-up.

1. Run `python SETUP.py` in the `src` directory. This will compile all of the code.
2. Start participants: Connect with all machines that you wish to run participants on. Run `java participant`
   `Participant` from the `src` directory. If this does not work you may need to use the non-default port. Try running with the optional port argument `java participant/Participant <port>`. If you see the message `File server ran into trouble` then the file server is unable to run on that machine so try running the participant on anther machine. An important note is that participants must run on different machines.
3. Once you have some participants running, you will need to update the config file to point to these participants. Example config files can be found at `src/examples/wordcount/wordcoun` and `src/examples/wordoccurences/wordoccurences.config`. The key lines to modify, if you plan to run the examples, are `PARTICIPANT` and `NUM_REDUCERS`. The `PARTICIPANT` values is `<hostname>:<port>`. When you started the participants there was a message to tell you which port it is running on.
4. You are now ready to start the master. Pick a machine that does not have a participant running on it. Run `java master/Master` from the `src` directory.
5. The master is now running. You will have a number of possible commands to enter to run jobs. See the next section for details.

1.3 Manage a Job

Now that you have the master and participants running you are ready to start a job. The master provides a command line interface to type three possible commands.

- `start <path_to_config_file>` This will start a process and print results as the process goes. It will also print the process id (PID) for your use with other commands.

- `stop <pid>` Stops the specified process
- `status <pid>` Gets the status of the specified process

2 For the Application Developer

2.1 Writing Your Map-Reduce Functions

We provide two interfaces to which the application developer must adhere. The first is `MapFn` which requires a `map(String input) -> MRKeyVal(String key, int value)` method. This method takes strings which are the lines of the input file and maps them to a key-value pairs where keys are strings and values integers. If `map` returns `null` then the mapper framework will ignore the result.

The second interface is `ReduceFn` which requires a `reduce(String key, List<int> values) -> MRKeyVal(String key, int value)` method. This method takes a key and all of the values that had that key and returns a reduced result on the values in the form of a single key-value pair. If this method returns `null` then the result is ignored.

2.2 Running Provided Examples

To run the provided examples follow the above steps and use the provided config files. Make sure to modify the participants and number of participants as necessary. Take a look at the provided code for examples of how to write the map and reduce functions.

2.2.1 Word Count

Word counts takes a file of one word per line and returns the sorted words with their counts.

Using the above steps run with `src/examples/wordcount/wordcount.config`.

The provided config file takes as input a file with all of the words in *Macbeth* and outputs a word count. The only thing necessary to change in the config file is the participants and number of participants.

2.2.2 Word Occurrences

Word occurrences looks specifically for the word 'macbeth' in the text of the play *Macbeth* and returns a count of the occurrences.

Using the above steps run with `src/examples/wordoccurences/wordoccurences.config`.

The provided config file takes as input a file with all of the text of *Macbeth* and outputs a word count for the word 'macbeth'. The only thing necessary to change in the config file is the participants and number of participants.

2.3 Using Remote Files

The remote file system is abstracted away in this system. Partitions, a name for the intermediate file type used in our framework, extend the `RemoteFile`. Remote files keep track of their original machine so that when the partition reference is passed to a new machine the file itself can be brought over from the original machine. Both participants and master run a file server which serves these remote files from the '/tmp' directory. If the application developer wishes to utilize the remote file object than they can instantiate a `RemoteFile` as long as the base file lives in the '/tmp' directory.

Remote files provide a `load` method to get the file from the remote machine. This must be called before doing anything on the file.

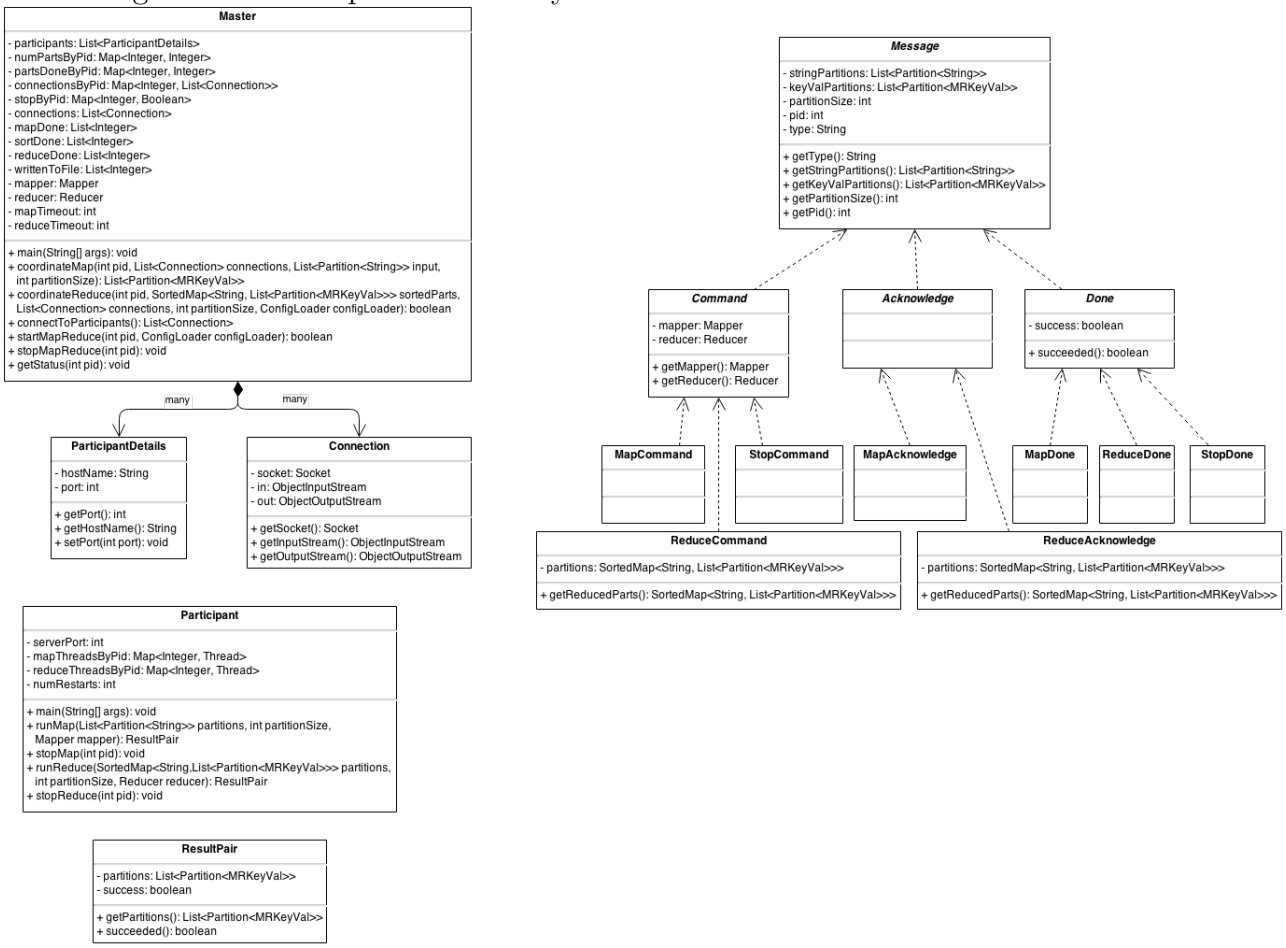
3 Project Requirements

3.1 Requirements Met and Capabilities

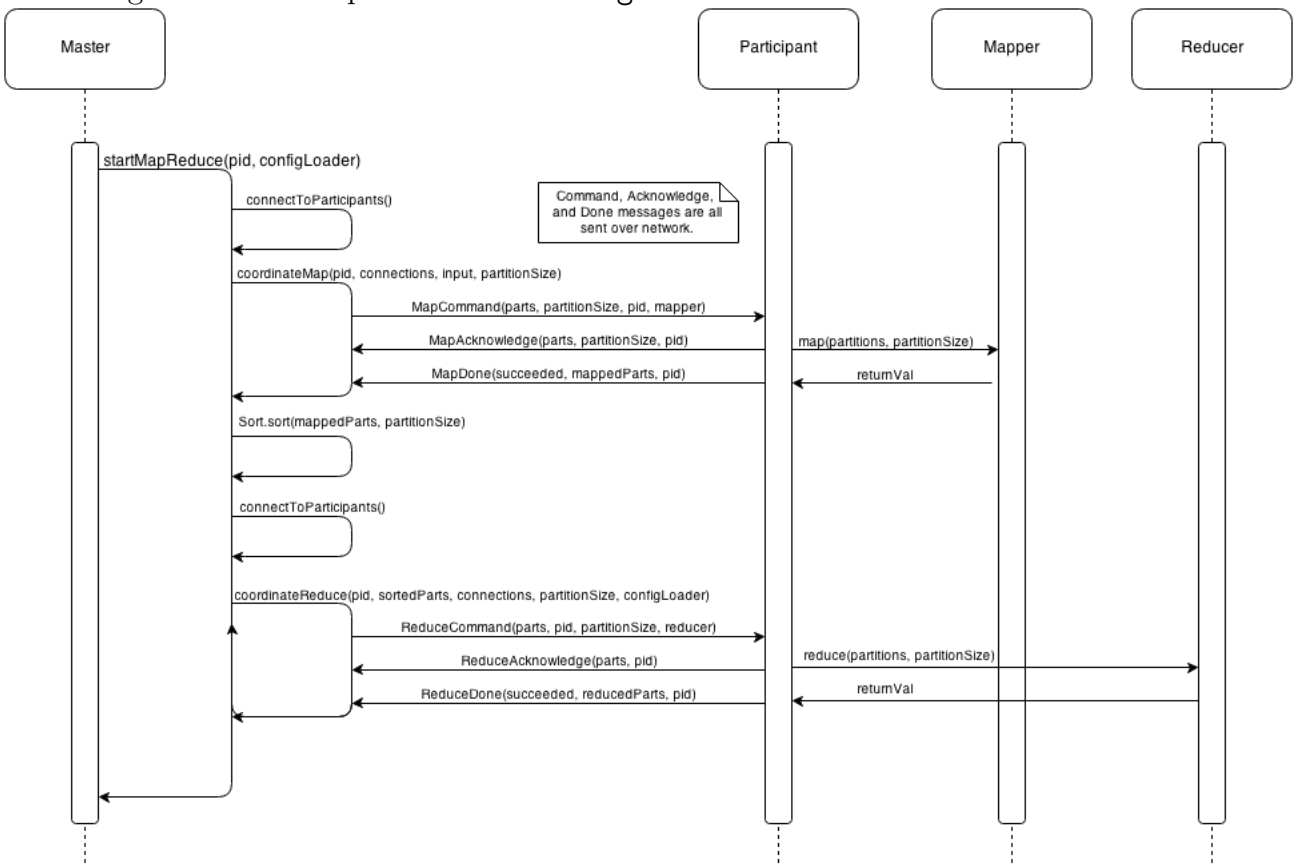
We have met all of the

TODO UML, FLOW DIAGRAMS

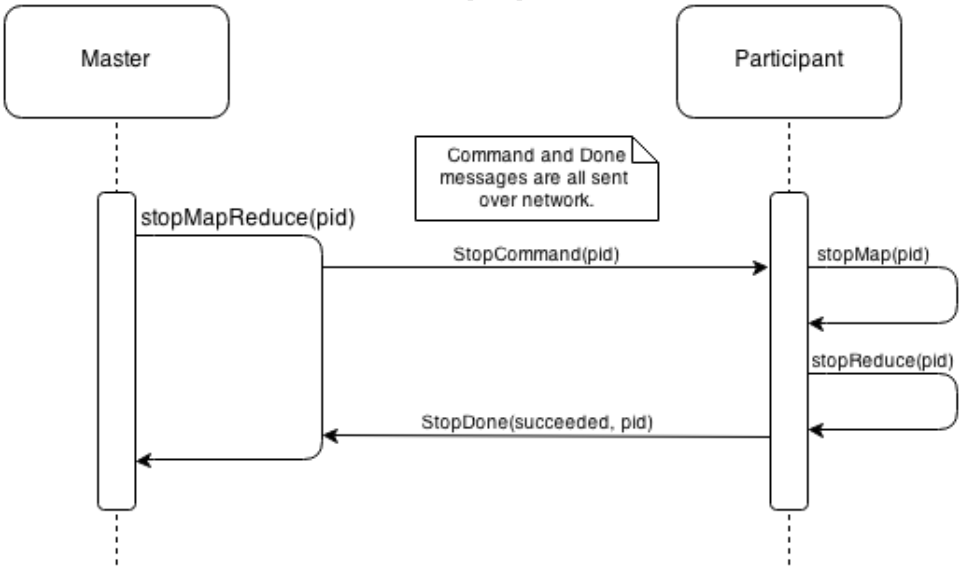
UML Diagram of our MapReduce facility:



Flow Diagram for user input "`start <config file>`":



Flow Diagram for user input "`stop <pid>`":



3.2 Requirements Not Met and Limitations

3.3 Improvements