## Setup/Compilation

To compile the code, first navigate to the MapleJuice folder. The task executables for maple/juice need to be compiled first like so:

```
javac -d maple_exe/ -sourcepath maple_exe maple_exe/*.java

jar -cfm maple_exe.jar maple_exe/Manifest.txt -C maple_exe WordCountMaple.class

javac -d juice_exe/ -sourcepath juice_exe juice_exe/*.java

jar -cfm juice_exe.jar juice_exe/Manifest.txt -C juice_exe WordCountJuice.class
```

Now the MapleJuice program can be compiled:

```
javac -d bin/ src/mp2/*.java src/mp3/*.java src/mp4/*.java src/mp5/*.java
```

**Starting Nodes and client:**

Navigate to the MapleJuice folder:

To start the Master and Slave nodes use the following command:

For the master node:

*java -cp bin mp4.SdfsNode host port masterHost masterPort 1*

For slave  nodes:

*java -cp bin mp4.SdfsNode host port masterHost masterPort 0*

To start the client:

*java -cp bin mp4.SdfsClient masterHost masterPort*

**Sending client commands:**

supported client commands include

get, put, delete, maple and juice.

Get command : get sdfsfilename

Put command: put localfilename sdfsfilename delete command: delete sdfsfilename

Using our maple_exe.jar and juice_exe.jar program, you can evaluate our application after starting the cluster:

maple command:  maple maple_exe.jar prefix sdfsfile1 sdfsfile2 ...

juice command: juice juice_exe.jar num_juices prefix destinationsdfsfilename

## Design/Functionality

This project builds heavily upon prior MPs. Components including the distributed file system, membership list , and logging mechanism are utilized heavily; their functionality is described in the previous reports. The core of the new functionality is split into two stages - maple and juice - with each part being started and supervised by tracker components.

The first stage, Maple, takes in an executable task file, along with a file prefix and one or more input files which the executable operates upon, and outputs files in the format *prefix_key*. These files are stored in the SDFS system from the previous MP.

The Juice stage takes the following parameters: juice task executable, number of juice instances, file prefix, and output file. Each juice instance is responsible for a variable number of files depending on how many keys maple produced and the number of juices specified by the user. In any case, the task  is performed on each key file, and the cumulative results are stored into a single destination file.

For our tasks, we chose to implement a word count feature. Each word in the input file is considered a key, and the maple function outputs a line of text in the format of a tuple (*word*, 1) for each word seen. Since the word name is the key, these tuples are stored in the file *prefix_word*. For example, if the input file was a sentence saying "foo bar", and the prefix given was "test", the maple task would output (foo, 1) to the file test_foo, and (bar, 1) (bar, 1) to the file test_bar.

The juice task file's job is to sum up the number of tuples in each intermediate file output by maple and store a new tuple in the format (*key, total)* in the given destination file. Continuing from the previous example, the juice task would output (bar, 2)(foo, 1).

Supervision of the maple and juice jobs is handled by a job tracker component. This component is responsible starting tasks, assigning tasks to specific nodes, and noticing failures should they occur. The delegation of tasks to nodes is determined by which nodes hold the SDFS files to be operated upon. So a maple task is performed by the node which holds the input file specified. Similarly, juice tasks are performed by the nodes holding the intermediate files output by maple.

There is also a task tracker module which monitors the status of individual tasks. Responsibilities of this component include calling the maple and juice executables directly, pulling files from SDFS to be operated upon locally, and initializing a heartbeat system which is used to detect failures and indicate that the task should be restarted should a failure occur.

## Failure Tolerance

The project has been update from MP4 to provide tolerance to failure.

In particular, upon a master election, the master browse the membership and selects the first available node as the secondary. Upon connection with the client, the front-end is provided both master and secondary info and default to the secondary node, should an issue occur with the frontend.

When the master receives a new job, it sends the job information to the secondary node for safekeeping. We keep the design simple by limiting the amount of bookeeping to be done.

Indentifying master failure: instead of having the secondary master polling the primary for status, we instead rely on the front-end request for job status in order to trigger a backup if necessary. More precisely, when the master has crashed, the front-end polling for job request is automatically sent to the secondary node. The secondary node proceeds then to restore the saved job and execute it while an election for a new master is already ongoing.

## Measurements

Empty file: While the front-end takes about 3 seconds to realize the job is already done, a maple task take a little more than 2 second, even with an empty file.

For regular file size, it takes more than 25 second for less than 1k file size on a three node cluster.

## Logging

Logging information as well as maple and juice command results are saved in respective log files

logs can be found in the folder **/tmp/ayivigu2_kjustic3**