**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

**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.