Daniel Hunnicutt

Bryan Hickerson

Project 4 README and Description

Team Contribution

Both team members shared the workload, participating in coding and documentation.

Building and running the code

We have provided a make file in our project root directory. To build the library and corresponding files, simply type make in the command line. This will compile our library into a librvm.a file which can be linked easily to other source.

To build and run the library with all of the tests use ‘make test’. The provided tests can then be executed. An OK result is the expected outcome of the tests.

Short Description:

The project is a lightweight implementation of transactional memory using files. This means that altering data in memory does not flush to disk unless an explicit rvm_commit_trans is executed. It also means that altering data in memory followed by an rvm_abort_trans should reset the data to its original value before altering.

Our library uses multiple logs to accomplish transactional memory. There is a main log named store.log (arbitrarily named) which tracks all the current actions on the directory. Along with the main log, each individual segment gets its own file for storing the data and own log. Whenever a rvm_map, rvm_about_to_modify, rvm_commit, or rvm_abort is called on a particular segment, both the main log and the segment log will get updated with the corresponding action. The most important part of transactional memory is the ability to restore data to its previous state if the program errors out or explicitly aborts the transaction. We restore the data during an rvm_abort by locking the log file (which will lock out other processes from altering the data) and restoring the data in memory by reading the actual data from disk. This effectively restores the data to its original state before the transaction. Our transactional memory does not support nested transactions though. So if a rvm_commit occurs and succeeds, that will be flushed to disk and be considered the valid state of the segment.

As far as we know, all transactions work to specification and all tests pass. We know that transactional memory is slower than other ways of managing memory, but the goal was reliable memory not speed. We believe that all requirements were satisfied.