Transactional System: Memo

8th December 2014

TO: Dr. Otte

FROM: Salvatore, Ryan, Chris, Andrew

SUBJECT: Transactional System Design

Our transactional system design uses a shared array of Cubbys, each of which contains an integer value, and a group of semiautonomous Agents (the number of which is determined by the number of cores on the host machines) which operate on those values. Concurrency is managed with locking, which can be turned on or off. Termination can be set to occur after the main thread has run for a given number of seconds, or after a given number of modifications to the Cubby values has taken place.

Each system has a single instance of the Manager object, which is essentially a container for the associated Cubby and Agent objects. The Manager's constructor takes arguments for:

- The number of cubbies to open
- The maximum value with which a cubby may be initialized
- A boolean indicating whether or not to enable locking
- A string indicating whether to terminate after some time, or after a certain number of transactions
- An integer specifying the termination limit (in seconds or number of transactions)

The Manager constructs an array of Cubbys with limited random values and an array of Agents with length equal to the number of CPU cores on the host's machine. Then it creates a multiprocessing.Process object for each Agent, and initializes each of the Processes.

An Agent takes, in its constructor, a boolean indicating whether locking is enabled. Its self.run() takes as arguments a reference to the Manager's array of cubbyholes. It begins to select Cubbys from the array at random, and proceeds to lock them and apply a value change to each. When the Agent modifies the values of two Cubbys, it selects a delta value for the first Cubby and uses the negative of that value to modify the second. In this way, the sum of the Cubbys' values always remains constant.

Cubbys simply hold an integer, and an associated Boolean indicating whether that value is currently locked by some Agent. They have a get*lock() method which returns the lock's status, and set*lock() and release_lock() methods to match.