Jonathan Chu
CSCI-402
February 22, 2013

Part E: Performance

Workload Description

The test files that I created first started with building a balanced tree. By doing this, I ensure that the number of nodes in the two subtrees of the head are about the same. After this, I created a test file to do reading (queries) exclusively on the right subtree, and writing (adds and deletes) only on the left subtree. I do a large number of operations for each test file (about 300,000 commands) in order to put more stress on the system. Furthermore, the "encourage" more switching between reading and writing, I use another test file full of reads (on the proper side). I ran multiple instances of these tests concurrently.

Performance under Different Strategies

I expect this workload to have the intended results of having Coarse be slower than RW which in turn is slower than Fine. Since coarse locks the entire tree, each operation for each thread will lock the tree, and therefore be really slow.

Coarse essentially turns the system into a single-threaded system since multiple threads will be blocked while a single one carries out its operation.

For RW, I only expect a slight, if any performance increase. Since my tests consistently switch between reading and writing, there is not much of a performance gain because threads that are reading will be blocked while there is a write, and vice-versa. There are no long runs of only reads or writes in the tests, so the benefit of having multiple readers for multi-threading will be very apparent. Furthermore, there is a lot of overhead in my code when it comes to checking for mutual exclusion of readers and writers.

For fine-grain, I expect there to be a much larger performance gain This is because of the separation of the reading and writing on the two subtrees. Because of this, reads will not be blocked by writing and vice-versa.

Test Results

The following are the service times for the 4 concurrent threads running under each of the different strategies.

- Coarse (Average Service Time: 7252 ms
  -Thread 0 Terminated and Joined! Service Time: 7328 milliseconds
  -Thread 1 Terminated and Joined! Service Time: 7326 milliseconds
  -Thread 2 Terminated and Joined! Service Time: 7185 milliseconds
  -Thread 3 Terminated and Joined! Service Time: 7169 milliseconds

- RW (Average Service Time: 7180 ms)
  -Thread 0 Terminated and Joined! Service Time: 7251 milliseconds
  -Thread 1 Terminated and Joined! Service Time: 7252 milliseconds
  -Thread 2 Terminated and Joined! Service Time: 7128 milliseconds
  -Thread 3 Terminated and Joined! Service Time: 7128 milliseconds

- Fine (Average Service Time: 4449 ms
  -Thread 0 Terminated and Joined! Service Time: 4532 milliseconds
  -Thread 1 Terminated and Joined! Service Time: 4533 milliseconds
  -Thread 2 Terminated and Joined! Service Time: 4367 milliseconds
  -Thread 3 Terminated and Joined! Service Time: 4365 milliseconds

As you can see, the difference in performance clearly shows that Fine-grain locking is much faster at running this workload. Furthermore, it also shows that RW-locking did not have as much of a performance gain over Coarse-grain locking.

How to Run Workload

The test files I created for this part are names, “test3” and “test4”. To run the same workload, enter the following server commands:

```
>> s
>> E
>> test3
>> <output file name #1>
>> E
>> test3
>> <output file name #2>
>> E
>> test4
>> <output file name #3>
>> E
>> test4
>> <output file name #4>
>> g
>> w
```

This will create 4 threads running the test files that wait for g to go. When w is pressed, the server will wait for all the threads to return, and when they done they will print out their service times.