P1A Report

1 Introduction

In this report, we will try to report about the performance of a concurrent Hashtable with 1000 keys and 200 threads under 4 different synchronization options and a benchmark specified below.

1.1 The 4 different Synchronization Options

1. Coarse Grain: Using 1 mutex to lock up the entire table for each access
2. Coarse Grain RW: Using 1 Read-Write Lock on the entire table
3. Fine Grain: Using as many mutexes as there are buckets in the table to lock each bucket for each access on that bucket.
4. Fine Grain RW: Using as many Read-Write Locks as there are buckets in the table, each lock for 1 bucket.

1.2 The Benchmark

This bench mark has 3,007,996 commands including 3,002,000 gets, 4,996 puts, and 1000 removes.

1.3 Other

Besides the bench mark introduced above, we will also report about the average overhead of spawning a thread and joining a thread.

2 Experiments

2.1 Naive Performance Benchmark

Sync. Option	Wall Time (sec)	Throughput (ops/sec)
Coarse	0.522035	45762058.10
Coarse RW	2.13464	1409135.03
Fine	0.0490201	61362502.32
Fine RW	0.0717359	41931529.40

2.2 Edited Performance Benchmark

Because the get performance is so fast, we will need to let each thread sleep for 1 ms whenever it calls get in order to get a closer measure of the performance difference. Here is the result.

Sync. Option	Wall Time (sec)	Throughput (ops/sec)
Coarse	1.73189	1736828.551
Coarse RW	2.79543	1076040.538
Fine	1.30724	2301028.120
Fine RW	1.34542	2235730.107

As predicted, fine grain lock perform much better than the coarse grain lock because it allows more concurrent access than its coarse grain neighbor.
However, something surprising worth noting is that the programs with Read and Write locks actually perform worse than its Mutex counterpart. This is counterintuitive because in theory, a read and write lock implementation should be faster because it allows for as many read threads to finish concurrently as possible compared to mutex where there is a finite number of thread can finish concurrently.
We suspect that the Read and Write Lock actually perform worse in this case because there is an overhead of creating the Read and Write Lock (more locks and condition variables to initialize) and because writers with a read and write lock are actually less efficient than the writers in mutex because they have more instructions to perform and also more wait time.

2.3 Average Spawn Time and Join Time

Sync. Option	Average Spawn Time (sec)	Average Join Time (sec)
Coarse	0.000423615	0.00277990
Coarse RW	0.000355945	0.00837065
Fine	0.000473535	0.00022812
Fine RW	0.000443680	0.00045128

There are two things worth noting here:

1. The Average Spawn Time is somewhat similar regardless of the synchronization option used.

2. The more important thing is that the Average Join Time is faster by an order of 10 magnitude when running on the faster synchronization method. This is due to the fact that the join function will make the main thread waiting on the worker threads to finish. Thus, the longer the worker thread takes to finish, the longer the main thread wait, hence the more time is waste not doing work.