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 3 kind of benchmark

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

1. Read only: a total 3,002,000 get commands split evenly among 200 threads.
2. Write only: 4,996 puts and 1000 removes split evenly among 200 threads.
3. Mix of Read and Write: 3,007,996 commands including 3,002,000 gets, 4,996 puts, and 1000 removes, split evenly among 200 threads.

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

2.1.1 Read Only

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	0.408884	7,341,935.61
Coarse RW	0.799159	3,756,448.97
Fine	0.0393131	76,361,314.68
Fine RW	0.101336	29,624,220.42

2.1.2 Write Only

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	0.00861979	695,608.59
Coarse RW	0.0707738	84,720.62
Fine	0.0054729	1,095,580.04
Fine RW	0.00751996	797,344.67

2.1.3 Mix of Read and Write

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	0.522035	45,762,058.10
Coarse RW	2.13464	1,409,135.03
Fine	0.0490201	61,362,502.32
Fine RW	0.0717359	41,931,529.40

2.2 Edited Performance Benchmark

Because the get performance is so fast, we will need to let each thread sleep for 1 microsecond whenever it calls get in order to get a closer measure of the performance difference. Since the sleep command is only applied for the read operations, there will be no different on the second and third benchmarks. So we will report only the first and fourth benchmarks here.

2.2.1 Read Only

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	1.4874	2,018,286.944
Coarse RW	1.75429	1,711,233.604
Fine	1.28838	2,330,057.902
Fine RW	1.30686	2,297,109.101

2.2.2 Mix of Read and Write

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	1.73189	1,736,828.551
Coarse RW	2.79543	1,076,040.538
Fine	1.30724	2,301,028.120
Fine RW	1.34542	2,235,730.107

2.2.3 Comment

As predicted, fine grain lock perform much better than the coarse grain lock because it allows more concurrent access than its coarse grain counterpart.

Also, there was no surprise when the Read and Write lock performs worse than its Mutex counterpart when it comes to write only task. This is because the writers in the read and write lock have more instructions to execute and also have to spend more time waiting.
However, something surprising worth noting is that the programs with Read and Write locks actually perform worse than its Mutex counterpart even on the Read Only benchmark and Mix of Read and Write benchmark. 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 the added overhead within the locking mechanic when using read write block. These overheads come in the form of extra condition checks, extra instructions, and more time spent waiting.
We believe that these added overheads outweigh the gain from unlimited reader access when executing with only 200 threads. We believe if the number of threads was to be increased to a proper amount, gain in performance will outweigh these overhead. Extra experiments can be found in section 3.

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.

3 Extra Experiment with different number of threads

Since Read and Write locks are designed to optimize performance for reader-intensive applications, we will only test on Benchmark 1 and 4. Also, the 1 microsecond delay is still added in these experiments.

3.0.1 Read Only

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	1.4874	2,018,286.944
Coarse RW	1.75429	1,711,233.604
Fine	1.28838	2,330,057.902
Fine RW	1.30686	2,297,109.101

Table 1: 200 threads

Sync. Option	Wall Time (sec)	Operations per Second
Coarse	1.4874	2,018,286.944
Coarse RW	1.75429	1,711,233.604
Fine	1.28838	2,330,057.902
Fine RW	1.30686	2,297,109.101

Table 2: 400 threads

3.0.2 Mix of Read And Write