Assignments for week 7

Anders Degn Lapiki, Jacob Kjærulff Furberg, Jonas Ishøj Nielsen

01/10/2020

Contents

1 7.1

1.1 7.1.1 - green

We chose to make the ArrayList concurrent by implementing the ReentrantReadWriteLock on both the items and the size of the arraylist. Since the ReentrantReadWriteLock uses the methods lock and unlock, we didn't have to use the synchronized keyword for locking.

We considered using a volatile long/int for the size, but we chose to "play it safe" and avoid using it.

We considered using striping to split locks all over the item array, similar to how the ConcurrentHashMap split its hashmap into smaller parts, where a single lock is responsible for each part. To test it we made an additional version using striping.

We also considered using LongAdder objects, in stead of Longs. This would allow several threads to access each long at the same time. We didn't do it since we return values and not references so an example where a user writes: long x = lst.get(i); and then x=2; will not affect lst.get(i).

Using mark6 the scaling performance can be seen in `out71.txt`. The last results are:

```
msg                                     mean         sdev      count
Better add:                       144493.3 ns     2327.52       4096
Stripping add:                    201844.0 ns    15726.56       2048
Synchronized add:                  55787.2 ns     4630.80       8192

Better get:                           32.6 ns        1.48    8388608
Stripping get:                        53.1 ns        1.98    8388608
Synchronized get:                      7.5 ns        0.43   33554432

Better set:                           30.9 ns        1.42    8388608
Stripping set:                        52.9 ns        2.61    8388608
Synchronized set:                      7.6 ns        0.21   67108864
```

In general it would be easier to make the class efficiently synchronized, if we had more information about it. Mainly it would be good to know how it will be used. Fores it have many writes? Or is it mainly reads? Also it would be good to know a performance requirement for the class. Such that we knew when to stop modifying the class.

2 7.2

2.1 7.2.1 - Green - Get

Done

2.2 7.2.2 - Green - Size

If the strip `s` is not locked, visibility is not guaranteed. In StripedWriteMap size is an AtomicIntegerArray meaning visibility is guaranteed and thus locking is not required.

2.3 7.2.3 - Green - putIfAbsent

Done

2.4 7.2.4 - Yellow - putIfAbsent call reallocateBuckets

Done

2.5 7.2.5 - Yellow - remove

Done

2.6 7.2.6 - Yellow - forEach

We implemented version (2) as it only locks on 1 out of 16 stripes at a time. Had there been a requirement that they must be iterated over in a specific order then we could not have used this version.

2.7 7.2.7 - Yellow

Using small number reduces memory usage and operations like size where all locks are used become constant time instead of linear time.

2.8 7.2.8 - Yellow

Increasing lock count to 32 increases lock granularity. This can reduce lock contention thus giving it better concurrent performance.

2.9 7.2.9 - Red

Result can be seen in `out72.txt`. All four classes give the same result. This is summarized to:

```
class SynchronizedMap, class StripedMap, class StripedWriteMap, class WrapConcurrentHashMap
        17 maps to B
       117 maps to C
        34 maps to F
       217 maps to E
        17 maps to B
        17 maps to B
       217 maps to E
        34 maps to F
       217 maps to E
        17 maps to B
        34 maps to F
```

2.10 7.2.10 - Red

It makes sense that the SynchronizedMap is the slowest and that StripedWriteMap is faster than StripedMap.
We expected WrapConcHashMap to be faster as it uses java's ConcurrentHashMap implementation, but it was the second slowest

Todo: ask what the ??? column is because it isn't mentioned in code, appears to be a secret print statement called from somewhere.

```
# OS:   Mac OS X; 10.15.4; x86_64
# JVM:  Oracle Corporation; 14.0.2
# CPU:  null; 8 "cores"
# Date: 2020-10-17T11:37:00+0200
             threadCount                mean       sdev      count
SynchronizedMap       16        649864,7 us   15359,39          2
    99992.0        ???
StripedMap            16        152497,4 us   57897,03          2
    99992.0        ???
StripedWriteMap       16         93043,8 us   20696,53          4
    99992.0667     ???
WrapConcHashMap       16        250045,7 us   47728,03          2
    99992.0        ???
```

2.11 7.2.11 - Red

index of k1= 5% 3 = 2, index of k2= 8 % 3 = 2.
index 0 = 0%2= stripe0, index 1 = 1%2= stripe1, index 2 = 2%2= stripe0.

It hurts performance that they don't all guard same amount of buckets, but we don't see any reason why thread safety should suffer.