Homework 3 Report

Kun Zhou
3106942495

lnxsrv07.seas.ucla.edu
Java version: 1.8.0_112
CPU: Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz
Memory 64GB
Four new models are implemented: UnsynchronizedState, GetNSetState, BetterSafeState, BetterSorryState. The main goal is to add data race intendedly in order to speed up and measure whether things will break down. It may help to discover a balance between speed and data-race free.

UnsynchronizedState: It is similar with SynchronizedState class except for removing all *synchronized* keywords. Before some thread sets new value for byte array at index i, another thread may have read the previous byte at index i rather than the new byte which should be read. Thus, byte array is not stable and it's possible that at some state, all bytes in array are *maxval* or 0 which leads to an infinite loop.

GetNSetState: Here *AtomicIntegerArray* is imported and used to storage byte array. When some thread implements *incrementAndGet* method, all other threads are not able to interfere the reading, updating, and writing procedures. However, different threads can interfere with each other between *incrementAndGet* and *decreamentAndGet.* Therefore, this class decrease the possibilities of breaking down and speed up the program, but data-race free is not perfectly solved.

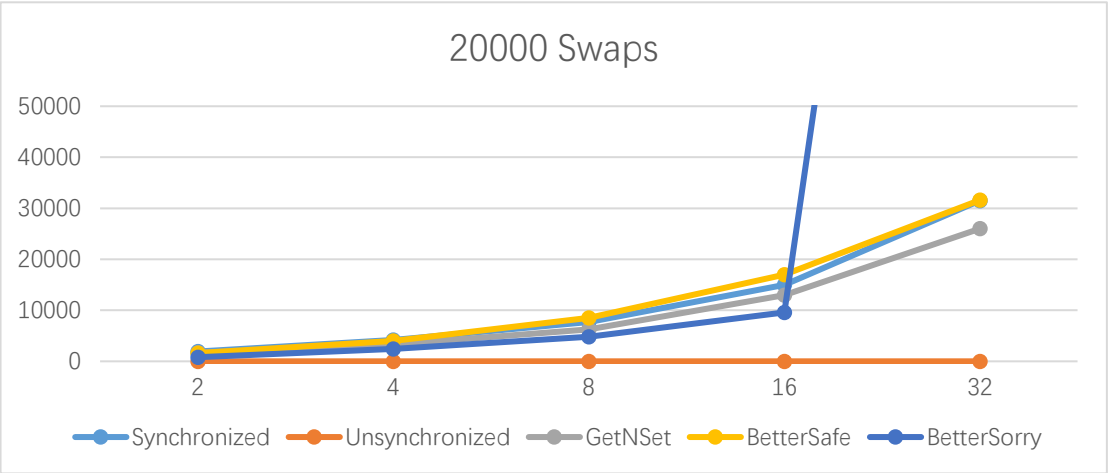
BetterSafeState: This class use *reentrantlock* instead of *synchronized*. The *reentrantlock* use *synchronized* methods and statements, but with extended capabilities. It is not complicated in our cases (using *lock* and *unlock* method) and more efficient. Lock before reading and unlock after writing.

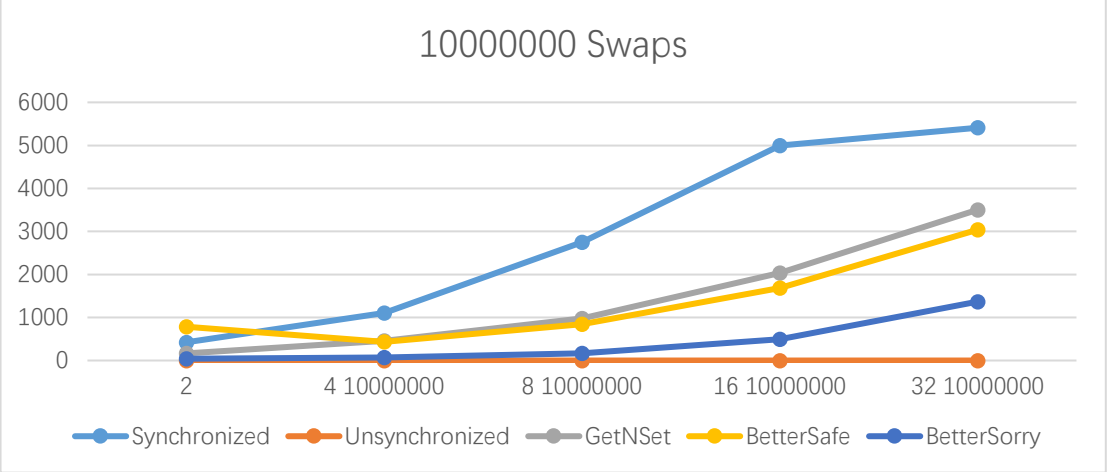
BetterSorryState: Unlike UnsynchronizedState, it will initialize two new variables ai and aj to store the bytes at index i and index j, then implement ++ and -- on these two variables and finally assign the value of ai and aj back to the byte array. Essentially, this class has no difference with UnsynchronizedState. But it speeds up much faster and never met infinite loop when testing. Because local variables are thread safe.

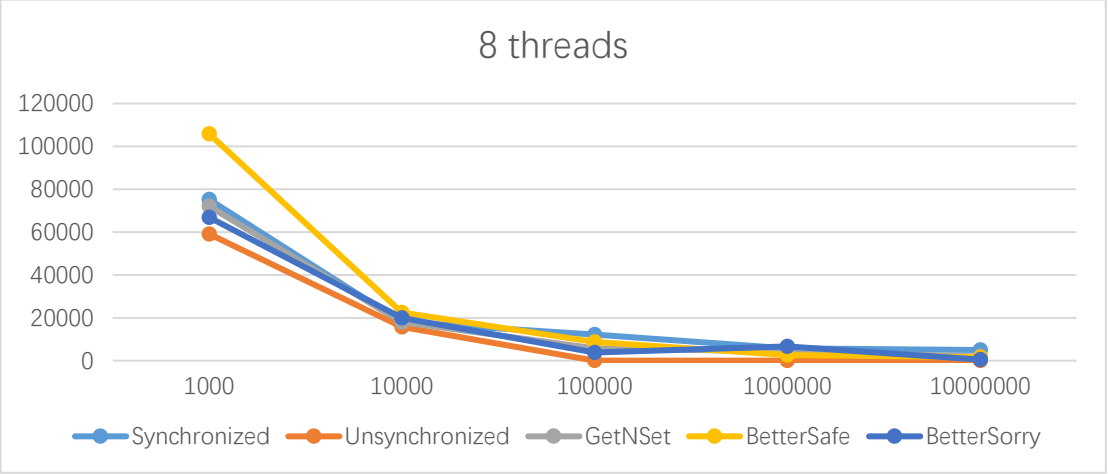
SynchronizeStae, UnsynchronizedState, GetNSetState, BetterSafeState, BetterSorryState are tested with different number of threads and successful swap transitions. The initial array is always [5, 6, 3, 0, 3] and *maxval* is 6. Speed is measured by 'ns/transition'.

In the first and second graph, 2000 swaps and 10000000 swaps are used, respectively. It shows

the relation between speed and number of threads. In the third graph, 8 threads are used. It demonstrates the relation between speed and number of swaps.







BetterSorryState is usually the fastest when number of swaps is large. Though there is no error suggested when implementing BetterSorryState, there is no guarantee my program gets the right result. GetNState gets the second place. BetterSafeState gets the third place and most importantly it is as reliable as SynchronizedSate. UnsychronizedSate performs poorly since it will go into infinite loop with 10000 swaps and over 3 swaps.