CS131 Homework3 Report

1. AcmeSafeState Implementation

The AcmeSafeState uses AtomicLongArray in java.util.concurrent.atomic module to implement the class. Its class method uses functions that access and update the AtomicLongArray in volatile mode. For example, the swap method uses getAndIncrement()/getAndDecrement(), which has same memory effect as getAndAdd(), which, according to Lea's paper, is a consensus-2 operation that guarantees that no other write occurs between the read and write of the operation.

Since the class methods access and update the AtomicLongArray in volatile mode, these operations are totally ordered, hence no two accesses can happen at the same time. As a result, the AcmeSafeState is DRF (Data-Race Free).

2. Problem of measurement to overcome

The measurement process is straightforward. I simply run the different combination of threads and array size on lnxsrv 09 and 10 and record the results.

3. Measurements of Different Classes

For all the below test cases, I used 100 million swaps for each.

Here are what the columns represent:

Col1: (Thread Number, Array Size)

Col2: Total Time (Real)

Col3: Total Time (CPU)

Col4: Average Swap time (Real)

Col5: Average Swap time (CPU)

2.1 Tests on lnxsrv 09

CPU spec:

Model Name: Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz

Cache size: 20480 KB

Cpu cores: 8

Number of Threads: 16

Memory spec:

MemTotal: 65755720 kB

MemFree: 36906112 kB

MemAvailable: 63019296 kB

2.1.1. SynchronizedState

Note: For all combination of (Thread Number, Array size), SynchronizedState passes the output test.

(1,5)	1.92s	1.92s	19.18ns	19.17ns
(1,50)	1.96s	1.96s	19.63ns	19.62ns
(1,100)	1.92s	1.92s	19.23ns	19.22ns
(8,5)	31.9s	109s	2549ns	1093ns
(8,50)	27.8s	88.1s	2224ns	881.2ns
(8,100)	28.2s	88.4s	2255ns	884.1ns
(20,5)	32.1s	111s	6425ns	1107ns
(20,50)	26.4s	83.4s	5280ns	833.7ns
(20,100)	28.6s	90.1s	5724ns	901.4ns
(40,5)	32.7s	114s	13068ns	1136ns
(40,50)	27.0s	84.4s	10808ns	844.5ns
(40,100)	28.8s	90.4s	11519ns	904.0ns

Table 1: SynchronizedState on lnxsrv09

2.1.2. UnsynchronizedState

Note: For Unsynchronized State, one additional column is added: Col6 represents the output sum

(1,5)	1.37s	1.37s	13.74ns	13.72ns	0
(1,50)	1.41s	1.41s	14.07ns	14.06ns	0
(1,100)	1.44s	1.44s	14.43ns	14.42ns	0
(8,5)	2.29s	17.0s	182.9ns	169.8ns	-43701
(8,50)	5.20s	40.8s	415.8ns	408.3ns	-37305
(8,100)	4.42s	35.0s	354.0ns	349.5ns	-18170
(20,5)	4.21s	80.5s	842.4ns	804.7ns	-11836
(20,50)	4.11s	80.7s	821.7ns	807.4ns	-19101
(20,100)	3.30s	64.8s	660.5ns	648.2ns	-31488
(40,5)	2.49s	75.7s	997.0ns	756.8ns	107426
(40,50)	3.49s	104s	1395ns	1045ns	-18920
(40,100)	3.05s	91.7s	1220ns	916.8ns	-37441

Table 2: UnsynchronizedState on lnxsrv09

2.1.3. AcmeSafeState

Note: For all combination of (Thread Number, Array size), AcmeSafeState passes the output test.

(1,5)	2.53s	2.53s	25.32ns	25.31ns
(1,50)	2.56s	2.56s	25.59ns	25.58ns
(1,100)	2.47s	2.47s	24.68ns	24.67ns
(8,5)	14.3s	112s	1143ns	1120ns
(8,50)	14.1s	112s	1126ns	1120ns
(8,100)	7.80s	62.0s	623.7ns	620.3ns
(20,5)	11.2s	216s	2232ns	2159ns

(20,50)	8.15s	160s	1630ns	1602ns
(20,100)	2.79s	52.4s	557.7ns	524.4ns
(40,5)	8.77s	267s	3508ns	2673ns
(40,50)	3.47s	106s	1387ns	1058ns
(40,100)	4.23s	125s	1690ns	1254ns

Table 3: AcmeSafeState on lnxsrv09

2.2 Tests on lnxsrv 10

CPU spec:

Model Name: Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz

Cache size: 16896 KB

Cpu cores: 4

Number of Threads: 4

Memory spec:

MemTotal: 65799628 kB

MemFree: 10547932 kB

MemAvailable: 14056080 kB

2.2.1. SynchronizedState

Note: For all combination of (Thread Number, Array size), SynchronizedState passes the output test.

(1,5)	1.64s	1.64s	16.43ns	16.42ns
(1,50)	1.70s	1.70s	17.02ns	16.98ns
(1,100)	1.69s	1.69s	16.94ns	16.93ns
(8,5)	4.78s	5.65s	382.1ns	56.49ns
(8,50)	4.66s	5.53s	372.5ns	55.30ns
(8,100)	4.67s	5.51s	373.3ns	55.12ns
(20,5)	5.18s	6.51s	1035ns	65.06ns
(20,50)	4.77s	5.66s	954.4ns	56.57ns
(20,100)	4.79s	5.59s	957.2ns	55.90ns
(40,5)	4.92s	5.90s	1966ns	59.00ns
(40,50)	5.05s	6.18s	2018ns	61.84ns
(40,100)	4.85s	5.67s	1942ns	56.68ns

Table 4: SynchronizedState on lnxsrv10

2.2.2. UnsynchronizedState

Note: For Unsynchronized State, one additional column is added: Col6 represents the output sum

(1,5)	1.22s	1.22s	12.20ns	12.18ns	0
(1,50)	1.22s	1.21s	12.15ns	12.14ns	0
(1,100)	1.22s	1.21s	12.16ns	12.14ns	0
(8,5)	3.65s	14.3s	291.7ns	143.4ns	-6106
(8,50)	3.92s	15.6s	313.6ns	155.6ns	-6262
(8,100)	3.52s	13.9s	281.5ns	139.2ns	-24172
(20,5)	3.59s	14.3s	718.8ns	142.8ns	-4216
(20,50)	3.88s	15.4s	775.7ns	154.0ns	-1224
(20,100)	3.86s	15.4s	771.5ns	153.5ns	-18248
(40,5)	1.92s	6.89s	769.9ns	68.90ns	46257
(40,50)	3.82s	14.1s	1526ns	141.0ns	-940
(40,100)	3.53s	12.5s	1411ns	124.9ns	-4121

Table 5: UnsynchronizedState on lnxsrv10

2.2.3. AcmeSafeState

Note: For all combination of (Thread Number, Array size), AcmeSafeState passes the output test.

(1,5)	2.57s	2.56s	25.67ns	25.61ns
(1,50)	2.55s	2.55s	25.47ns	25.45ns
(1,100)	2.49s	2.49s	24.92ns	24.86ns
(8,5)	13.28s	43.17s	1062ns	431.7ns
(8,50)	8.31s	27.0s	665.0ns	269.6ns
(8,100)	5.87s	19.6s	469.8ns	195.9ns
(20,5)	12.6s	45.19s	2526ns	451.9ns
(20,50)	6.10s	20.8s	1220ns	208.5ns
(20,100)	7.18s	18.0s	1437ns	180.2ns
(40,5)	13.6s	50.9s	5451ns	509.0ns
(40,50)	12.0s	44.7s	4801ns	446.7ns
(40,100)	8.01s	27.4s	3204ns	273.5ns

Table 6: AcmeSafeState on lnxsrv10

4. Analysis of Measurements

4.1. SynchronizedState

Performance:

1. Multithread performance is much better on lnxsrv10 than on lnxsrv09(~5 secs vs. ~30 secs), this is unexpected since lnxsrv09 has more cores and more threads, which means that more threads can run on lnxsrv09 at the same time. The unexpected result might be resulted from different loads of the two servers, or how many threads are actually allocated to the program.
2. When the number of threads is larger than the number of threads, the class tends to run poorly since there is more conflicting updates on the same element of the long array. In this circumstance, increasing the size of the array would reduce the average swap time and hence decrease the total running time, since it reduces the change of conflict.
3. Running with a single thread is the fastest in both servers, this is probably because the cost of context switching outweighs the benefit of running in parallel.

Reliability:

This class has 100% reliability with arbitrary number of threads and array sizes, since it uses the synchronized keyword to make sure that two threads does not update the array at the same time.

4.2. UnsynchronizedState

Performance:

1. This class has similar performance in two servers in terms of total real time. However, CPU time in lnxsrv09 is much higher, which indicates that lnxsrv09 uses a higher number of threads to run the program than lnxsrv10.
2. The relative size of thread number and array size does not affect the total real time much in this class.
3. Again, running with a single thread is the fastest in both servers, which indicates that the cost of context switch outweighs the benefit of parallelism in this case.

Reliability:

1. When using only one thread this class has 100% reliability since only a single thread could update the long array at any time.
2. When using multiple threads, this class has close to 0% reliability when the number of swaps is large. This is because multiple threads could update a single element in the long array at the same time, which causes data race and possibly results in incorrect results.
3. When the number of threads is larger than the number the array size, the output sum tends to deviate much more from 0 than in other cases. This is because it is more probable that multiple threads could update the same element in the array at the same time, which causes data race.

4.3. AcmeSafeState

Performance:

1. This class has similar performance in two servers in terms of total real time, with lnxsrv09 performs a little bit better than lnxsrv10 on large array sizes. This could be due to fluctuation in load of two servers. However, the average CPU time (total or average swap) is much higher in lnxsrv09, which might be because lnxsrv09 assigns a higher number of threads to the program than lnxsrv10.
2. We see a decrease in running time when we increase the array size, which is probably because there are less conflicting updates when the array size is large.
3. Again, in both servers the class performs the best in a single thread, which indicates the cost of context switch is higher than the benefit of parallelism.

Reliability:

This class has 100% reliability with arbitrary number of threads and array sizes, since it uses AtomicLongArray and use volatile mode to prevent data races so that it always obtain the correct result.

4.4. Comparison across classes

Among the three classes, UnsynchronizedState is the fastest in terms of total running time, since it does not implement any measure to prevent data races. Hence the result of this class is not reliable except using a single thread. For the other two classes, the relative performance depends on the server, in lnxsrv09, AcmeSafeState has better performance than SynchronizedState in terms of total real time, but it is the opposite in lnxsrv10. These two classes are both 100% reliable since they use different measures to prevent data races. These three classes all perform the best using a single thread. The worst case for these three classes are not clear from the data, since except the single thread case, different combinations of threads and array sizes actually performs similarly, but generally more threads would result in an increase in average swap time.