Step2:

Case 1

n = 1,000 and m = 10,000, m_{Member} = 0.99, m_{Insert} = 0.005, m_{Delete} = 0.005

Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial	24.33ms	1.155ms				
One mutex for entire list	28.82ms	1.513ms	51.44ms	6.102ms	79.95ms	9.057ms
Read-Write lock	28.32ms	1.675ms	15.8ms	2.678ms	18.93ms	3.207ms

Case 2

n = 1,000 and m = 10,000, m_{Member} = 0.90, m_{Insert} = 0.05, m_{Delete} = 0.05

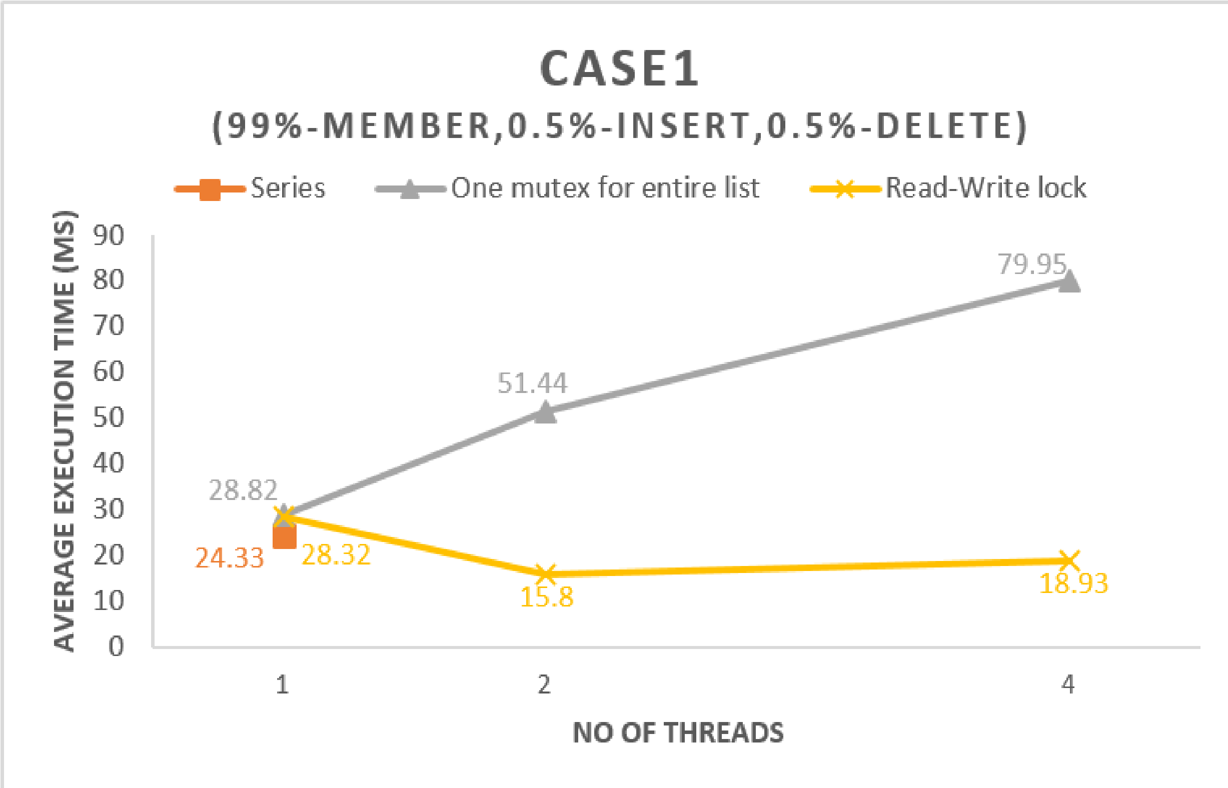
Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial	40.94ms	1.619ms				
One mutex for entire list	44.91ms	1.577ms	78.67ms	10.469ms	106.77ms	10.823ms
Read-Write lock	45.31ms	2.219ms	31.93ms	5.465ms	34.47ms	4.237ms

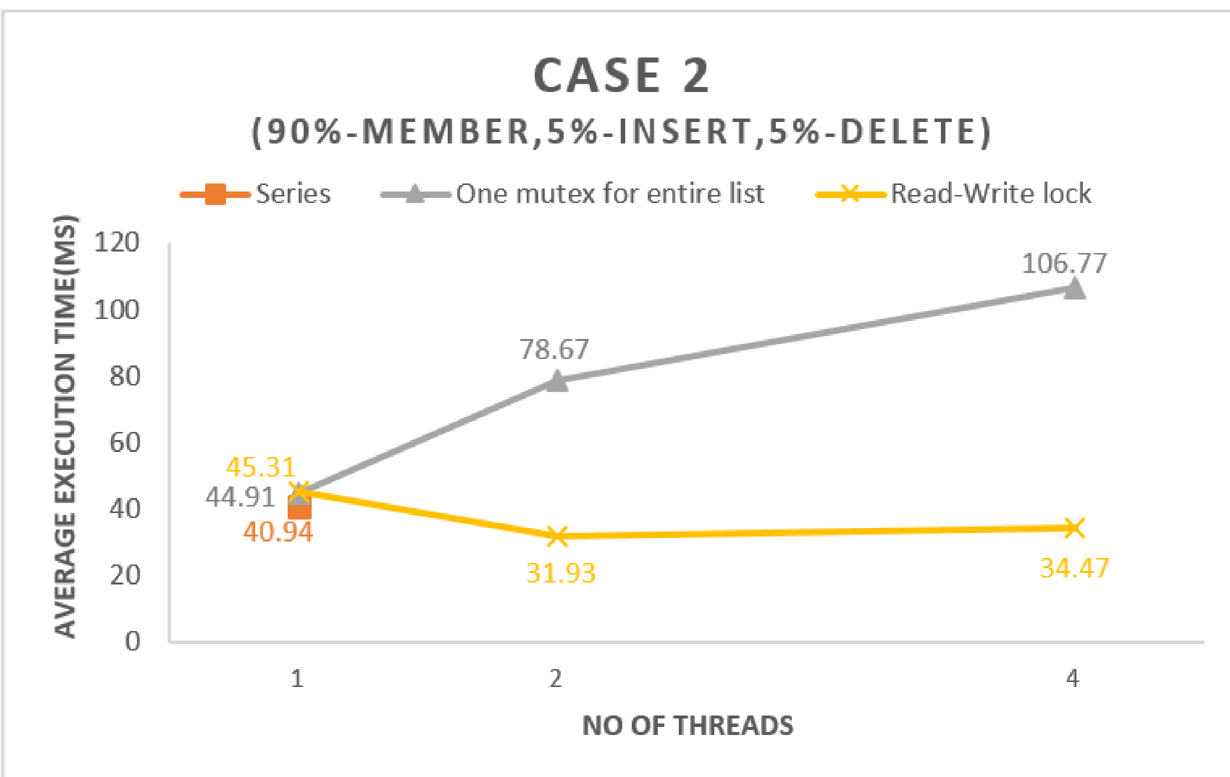
Case 3

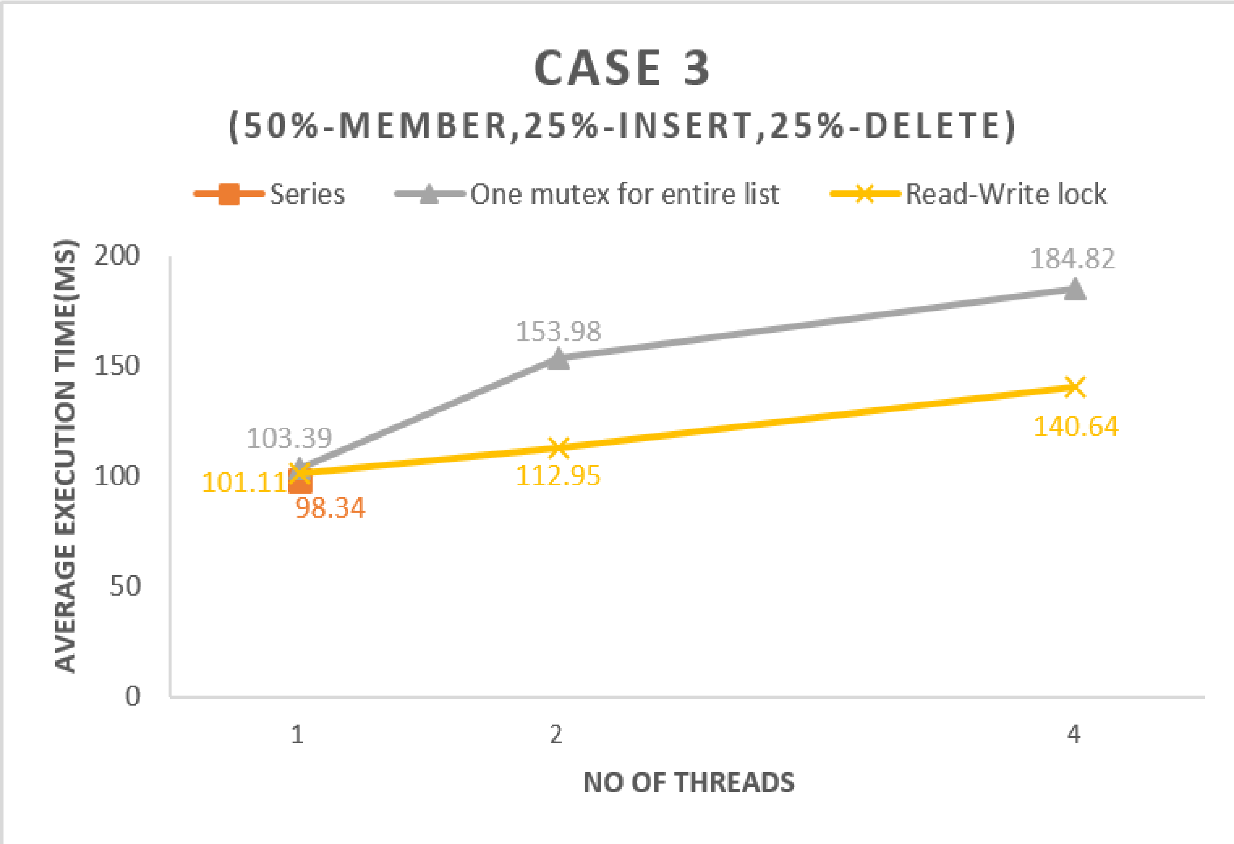
n = 1,000 and m = 10,000, m_{Member} = 0.50, m_{Insert} = 0.25, m_{Delete} = 0.25

Implementation	No of threads					
	1		2		4	
	Average	Std	Average	Std	Average	Std
Serial	98.34ms	4.404ms				
One mutex for entire list	103.39ms	5.464ms	153.98ms	11.145ms	184.82ms	17.148ms
Read-Write lock	101.11ms	2.382ms	112.95ms	10.693ms	140.64ms	19.054ms

Step 3:


CASE1
(99%-MEMBER,0.5%-INSERT,0.5%-DELETE)
Series
One mutex for entire list
Read-Write lock
AVERAGE EXECUTION TIME (MS)
90
80
70
60
50
40
30
20
10
0
79.95
51.44
28.82
24.33
28.32
15.8
18.93
1
2
4
NO OF THREADS


CASE 2
(90%-MEMBER,5%-INSERT,5%-DELETE)
Series
One mutex for entire list
Read-Write lock
AVERAGE EXECUTION TIME(MS)
120
100
80
60
40
20
0
106.77
78.67
45.31
44.91
40.94
31.93
34.47
1
2
4
NO OF THREADS



Step 4:

- The above results that are plotted in the graphs are based on our simulation. This is a discrete event because possible values for the thread counts are positive integers. Only for those values, we can get the execution time. So in the graph those points are denoted with different kind of pointers according to their data category. The connecting line is only for reveals the trend. Since serial operation is only applicable with 1 thread there is only one data point in the graph and no tread line for that.
- According to the equation for sample size determination $n = \left(\frac{100zs}{r\bar{x}}\right)^2$ we need an initial standard deviation and average of the distribution inorder to find the minimum number of samples. Initial average and standard deviations were calculated using 100 samples and required samples with an accuracy of ±5% and 95% confidence level are shown in the following table.

Case 1:

Implementation	Number of Threads		
	1	**2**	**4**
	Number of samples		
Serial	4		
One mutex	5	22	20
RW Lock	6	45	45

Case 2:

Implementation	Number of Threads		
	1	**2**	**4**
	Number of samples		
Serial	3		
One mutex	2	28	16
RW Lock	4	46	24

Case 3:

Implementation	Number of Threads		
	1	**2**	**4**
	Number of samples		
Serial	4		
One mutex	5	9	14
RW Lock	1	14	29

Since the required number of samples for all the test cases are below 100, our results are valid with an accuracy of ±5% and 95% confidence level.

- The machine specification that used to run our simulations
  - Ubuntu 14.04 x64 bit
  - Memory: 6 GB, 1067 MHz
  - CPU: Intel(R) Core(TM) i5 CPU M 460 @ 2.53GHz - 4 cores
  - L1d cache:32K
  - L1i cache:32K
  - L2 cache:256K
  - L3 cache:3072K

Observations

- In all three cases with single thread, serial code gives the better performance. Because if there is only one thread to access the link list, then there is no need for locks. But those parallel codes still use the locks there. Due to those lock acquire and release overhead those parallel codes give poor performance than serial code in single thread scenario.
- In all three cases Read-write lock gives the better performance than one mutex for entire list. This is because in the one mutex scenario if one thread is doing any operation with the

link list, no other threads are allowed to do any operations with that link list. But in Read-Write locks when one thread is doing read operation with the link list, another thread is allowed for only read operation. Due to this improvement Read-Write lock gives better performance. Case 1 to Case 3 the read operation percentage decrease, this resulted as the deviation between mutex and read-write lock also converged from Case1 to Case 3 (in the graph those two trend line come closer from case 1 to case 3).

- In Case 1 and Case 2, Read-write lock could be able to give better performance than the serial when thread count increases, Even though it fails with thread count 1. But in Case 3 it fails even we increase the thread count. This is due to, in case 3 only 50% operations are read other 50% are insert and delete. In the Read-write lock, when a thread is doing insert operation/delete operation, no other threads can access the link list. But when a thread is doing read operation, another read operation is allowed. Because of this, when read operation percentage falls its performance become poor.
- In all three cases, one mutex for entire list fails to give better performance than serial code even we increase the thread up to 4. Conversely, it gives poor performance when we increase the thread count. This is due to the overhead of accruing and releasing the lock. Since there is only one mutex for entire list, when the thread count increase the threads starvation time increase so it gives bad performance. Since no other operations are allowed in the entire list when an operation is performed with the list, it gives poor performance than serial code.