DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UNIVERSITY OF MORATUWA

In24-S3-CS5225 - Parallel and Concurrent Programming

Take Home Lab 1

M.K.T. Sampath
248272L

This is submitted as a partial fulfillment for the module
CS5429: Distributed Systems

October 6, 2024

1 Step 3: Experiment

1.1 Computer Specification

- Model Name: MacBook Pro
- Chip: Apple M1 Pro
- Total Number of Cores: 8 (6 performance and 2 efficiency)
- Memory: 16 GB
- OS: macOS 14.4.1 (23E224)

1.2 GCC Specifications

- Apple clang version 15.0.0 (clang-1500.0.40.1)
- Target: arm64-apple-darwin23.4.0
- Thread model: posix

1.3 Calculating Suitable Sample Count

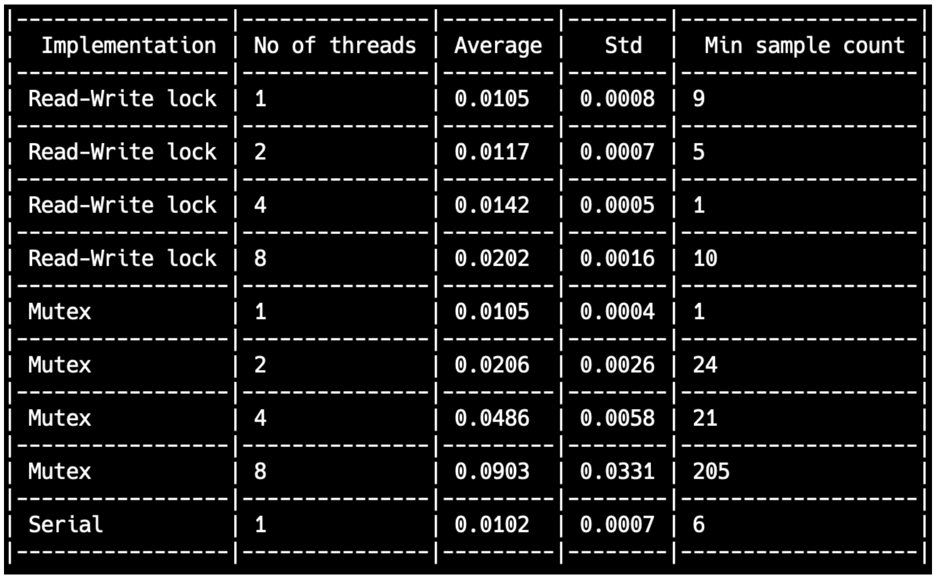
According to the method outlined in (`http://www.cse.wustl.edu/~jain/cse567-08/ftp/k_13cs.pdf`), the minimum number of samples (n) required for a given accuracy and confidence level can be calculated using the following formula:

$$n = \left(\frac{100 \cdot Z \cdot s}{r \cdot x}\right)^2 \tag{1}$$

In this equation:

- n is the required sample size
- Z is the Z-value for the specified confidence level
- s is the standard deviation of the sample
- r is the relative error
- x is the mean of the sample.

To compute n, both the sample mean (x) and sample standard deviation (s) are required. To gather these statistics, I designed the experiment to first run a given number of samples, allowing the user to input an initial sample size. Using this input, the mean and standard deviation are calculated, followed by determining the minimum required sample size for the given accuracy and confidence specifications.



```
|-----------------|---------------|---------|--------|--------------------|
|  Implementation | No of threads | Average |   Std  |  Min sample count  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 1             | 0.0105  | 0.0008 | 9                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 2             | 0.0117  | 0.0007 | 5                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 4             | 0.0142  | 0.0005 | 1                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 8             | 0.0202  | 0.0016 | 10                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 1             | 0.0105  | 0.0004 | 1                  |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 2             | 0.0206  | 0.0026 | 24                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 4             | 0.0486  | 0.0058 | 21                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 8             | 0.0903  | 0.0331 | 205                |
|-----------------|---------------|---------|--------|--------------------|
| Serial          | 1             | 0.0102  | 0.0007 | 6                  |
|-----------------|---------------|---------|--------|--------------------|
```

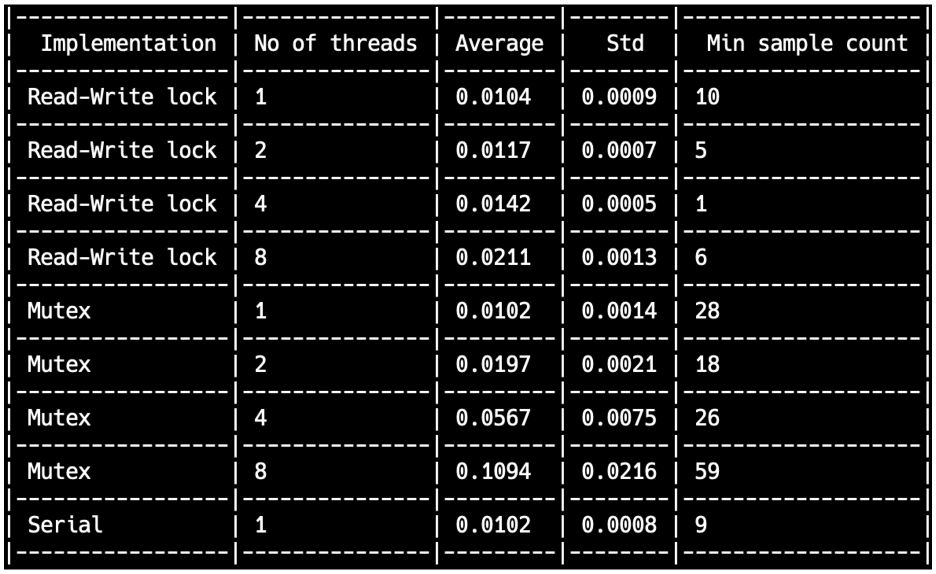
Figure 1: Sample size = 500

With an initial sample size of 500, the calculated minimum required sample size for Case 1 was found to be less than 20. This confirms that a sample size of 500 meets the desired accuracy and confidence level for the experiment. The same procedure was applied to the other cases, and it was consistently determined that 500 samples were sufficient across all scenarios. Therefore, to ensure consistency throughout the experiment, a sample size of 500 was used for all tests.

1.4 Results

1.4.1 Case 1

n = 1,000 and m = 10,000, mMember = 0.99, mIndert = 0.005, mDelete = 0.005



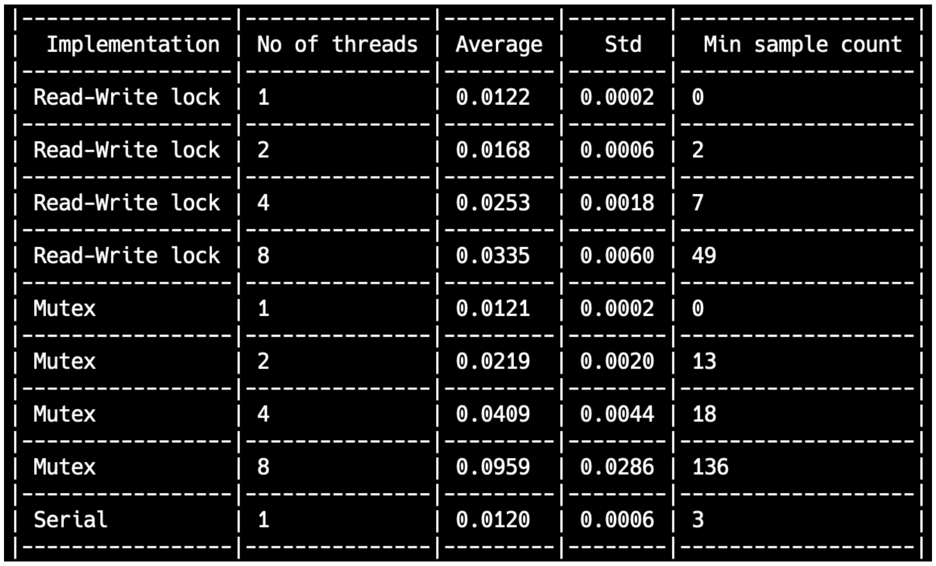
```
|-----------------|---------------|---------|--------|--------------------|
|  Implementation | No of threads | Average |   Std  |  Min sample count  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 1             | 0.0104  | 0.0009 | 10                 |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 2             | 0.0117  | 0.0007 | 5                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 4             | 0.0142  | 0.0005 | 1                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 8             | 0.0211  | 0.0013 | 6                  |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 1             | 0.0102  | 0.0014 | 28                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 2             | 0.0197  | 0.0021 | 18                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 4             | 0.0567  | 0.0075 | 26                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 8             | 0.1094  | 0.0216 | 59                 |
|-----------------|---------------|---------|--------|--------------------|
| Serial          | 1             | 0.0102  | 0.0008 | 9                  |
|-----------------|---------------|---------|--------|--------------------|
```

Figure 2: Case 1

Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.0102	0.0008						
One mutex for entire list	0.0102	0.0014	0.0197	0.0021	0.0567	0.0075	0.1094	0.0216
Read-Write lock	0.0104	0.0009	0.0117	0.0007	0.0142	0.0005	0.0211	0.0013

1.4.2 Case 2

n = 1,000 and m = 10,000, mMember = 0.9, mIndert = 0.05, mDelete = 0.05



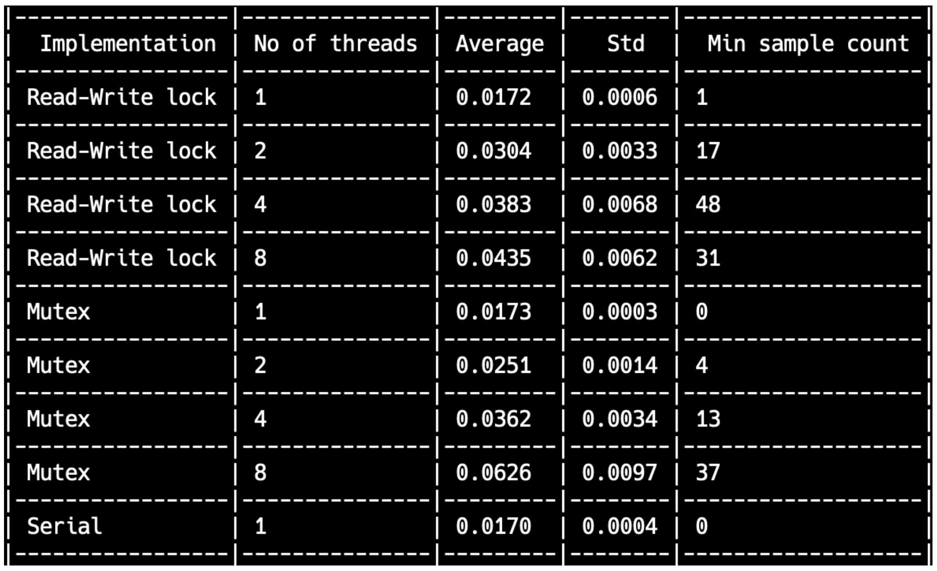
```
|-----------------|---------------|---------|--------|--------------------|
|  Implementation | No of threads | Average |   Std  |  Min sample count  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 1             | 0.0122  | 0.0002 | 0                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 2             | 0.0168  | 0.0006 | 2                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 4             | 0.0253  | 0.0018 | 7                  |
|-----------------|---------------|---------|--------|--------------------|
| Read-Write lock | 8             | 0.0335  | 0.0060 | 49                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 1             | 0.0121  | 0.0002 | 0                  |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 2             | 0.0219  | 0.0020 | 13                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 4             | 0.0409  | 0.0044 | 18                 |
|-----------------|---------------|---------|--------|--------------------|
| Mutex           | 8             | 0.0959  | 0.0286 | 136                |
|-----------------|---------------|---------|--------|--------------------|
| Serial          | 1             | 0.0120  | 0.0006 | 3                  |
|-----------------|---------------|---------|--------|--------------------|
```

Figure 3: Case 2

Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.0120	0.0006						
One mutex for entire list	0.0121	0.0002	0.0219	0.0020	0.0409	0.0044	0.0959	0.0286
Read-Write lock	0.0122	0.0002	0.0168	0.0006	0.0253	0.0018	0.0335	0.0060

1.4.3 Case 3

n = 1,000 and m = 10,000, mMember = 0.5, mIndert = 0.25, mDelete = 0.25



```
|------------------|---------------|----------|---------|-------------------|
|  Implementation  | No of threads | Average  |   Std   |  Min sample count |
|------------------|---------------|----------|---------|-------------------|
| Read-Write lock  | 1             | 0.0172   | 0.0006  | 1                 |
|------------------|---------------|----------|---------|-------------------|
| Read-Write lock  | 2             | 0.0304   | 0.0033  | 17                |
|------------------|---------------|----------|---------|-------------------|
| Read-Write lock  | 4             | 0.0383   | 0.0068  | 48                |
|------------------|---------------|----------|---------|-------------------|
| Read-Write lock  | 8             | 0.0435   | 0.0062  | 31                |
|------------------|---------------|----------|---------|-------------------|
| Mutex            | 1             | 0.0173   | 0.0003  | 0                 |
|------------------|---------------|----------|---------|-------------------|
| Mutex            | 2             | 0.0251   | 0.0014  | 4                 |
|------------------|---------------|----------|---------|-------------------|
| Mutex            | 4             | 0.0362   | 0.0034  | 13                |
|------------------|---------------|----------|---------|-------------------|
| Mutex            | 8             | 0.0626   | 0.0097  | 37                |
|------------------|---------------|----------|---------|-------------------|
| Serial           | 1             | 0.0170   | 0.0004  | 0                 |
|------------------|---------------|----------|---------|-------------------|
```

Figure 4: Case 3

Implementation	1		2		4		8	
	Average	Std	Average	Std	Average	Std	Average	Std
Serial	0.0170	0.0004						
One mutex for entire list	0.0173	0.0003	0.0251	0.0014	0.0362	0.0034	0.0626	0.0097
Read-Write lock	0.0172	0.0006	0.0304	0.0033	0.0383	0.0068	0.0435	0.0062

2 Step 4: Graphs

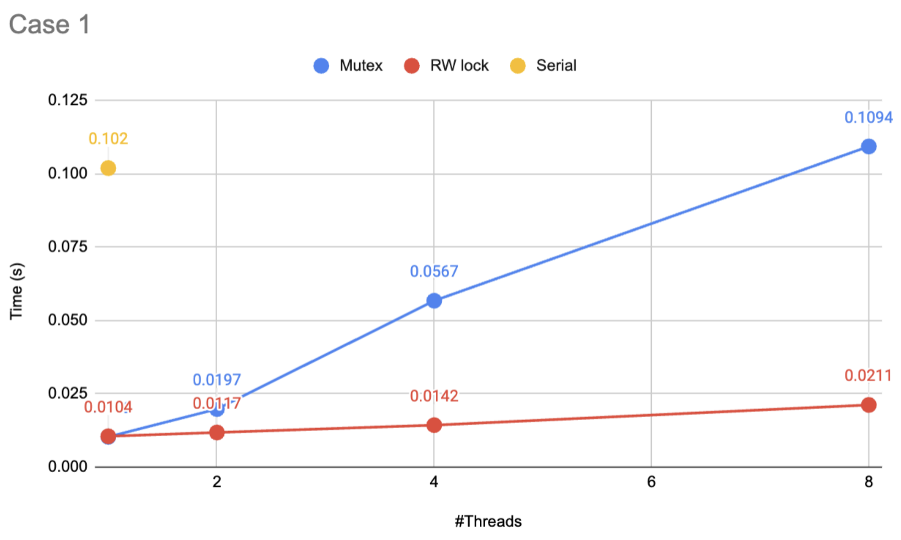


Figure 5: Case 1

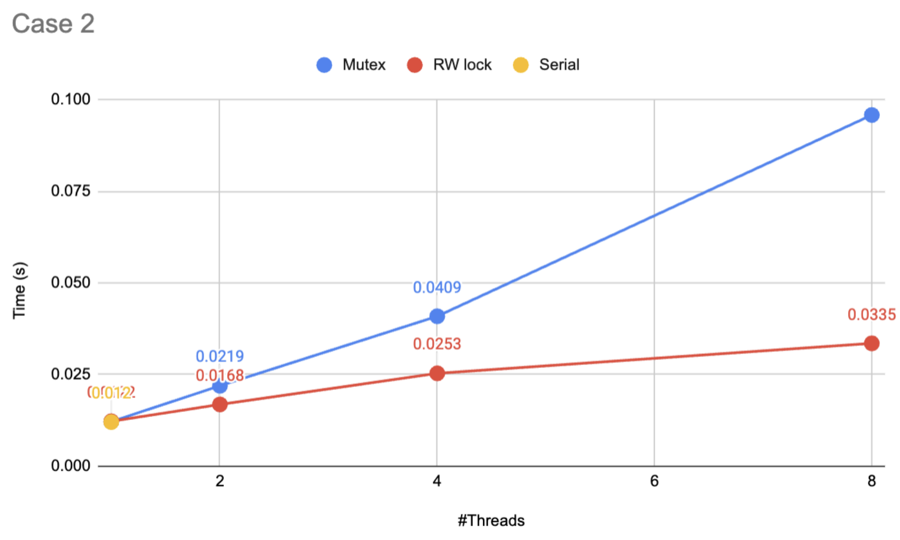


Figure 6: Case 2

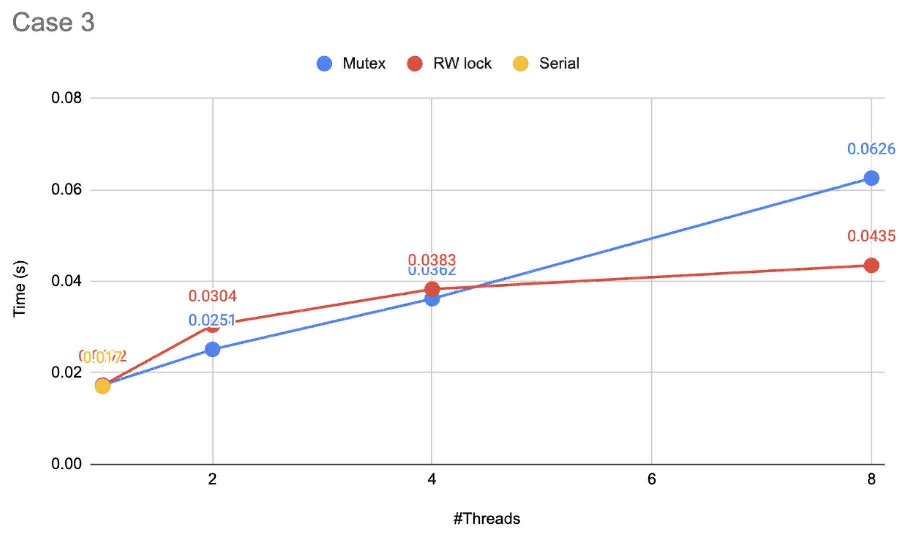


Figure 7: Case 3

3 Step 5: Observations

1. In all cases, the serial implementation provides the best performance when using a single thread. This is likely due to the absence of locking overhead, which impacts the other two implementations.

2. The performance of the mutex and read-write lock implementations with a single thread shows high variability, with no clear pattern emerging. This inconsistency is likely due to the random nature of the input operations, which can introduce unpredictable behavior even in the absence of true concurrency.

3. In all cases, as concurrency increases, overall performance declines (i.e., total execution time increases). This performance degradation occurs due to the overhead introduced by managing concurrent access. Specifically, synchronization mechanisms such as locks, which are required to ensure

data consistency in a multi-threaded environment, introduce contention and blocking. As more threads compete for access to shared resources, the time spent waiting for locks or managing thread coordination increases, leading to a rise in execution time despite the additional threads.

4. In both Case 1 (Figure 5) and Case 2 (Figure 6), the read-write lock implementation outperforms the others when using multiple threads, with performance improving as the thread count increases. This is likely because mutexes lock on every operation, including reads, which is unnecessary in many cases. The read-write lock mitigates this performance loss by allowing multiple threads to acquire the read lock simultaneously, while reserving exclusive locks only for write operations. This reduces contention and improves efficiency, especially under high concurrency.

5. In Case 3 (Figure 7), with lower Member operation ratio relative to Case 1 and Case 2, the mutex implementation shows slightly better performance with lower thread counts (2, 4). However, as the thread count increases, the read-write lock implementation surpasses it, delivering better results. An additional test run with few more thread counts, shown in Figure 8 and 9, confirms this behavior, validating the observation.

   The reason for this behavior can be the nature of the workload. In Case 3, where a relatively significant portion of the operations are writes (inserts and deletes), the mutex implementation initially handles the lower concurrency well, as the contention for write operations is low. However, as the number of threads increases, contention grows, and the mutex, which blocks all operations (including reads), starts to introduce significant overhead.

   In contrast, the read-write lock implementation, which allows multiple threads to read concurrently while restricting write access, becomes more efficient under higher concurrency. With an increased thread count, the read-write lock reduces contention during read operations, while still managing write operations effectively, resulting in better performance than the mutex.

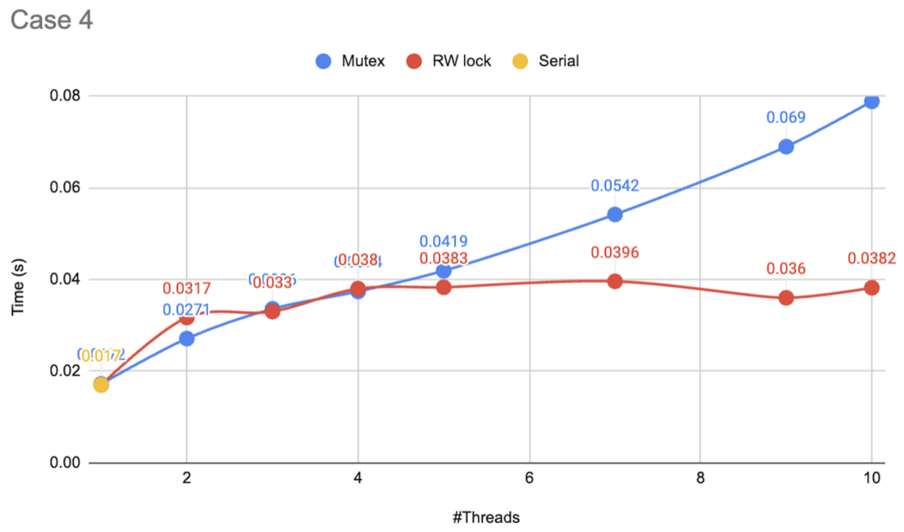


Figure 8: Case 4 : n=1000, m=10000, mMember=0.50, mInsert=0.25, mDelete=0.25, sample$_{count}$ = 500

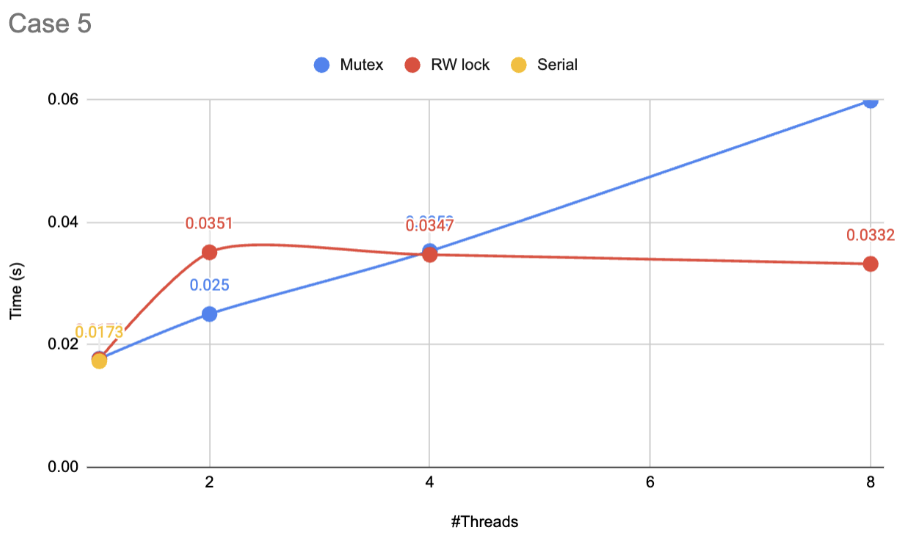


Figure 9: Case 5 : n = 1000, m = 10000, mMember = 0.20, mInsert = 0.30, mDelete = 0.50, sample count = 500

4 Coclution

The experiment demonstrates that the read-write lock implementation outperforms mutex-based locking as thread count increases, especially under higher concurrency. This is because read-write locks allow multiple threads to read simultaneously, reducing contention compared to mutexes, which block all operations, including reads. The performance degradation observed with higher thread counts across all cases is primarily due to the overhead of managing concurrent access.