CS4532 Concurrent Programming

Take-Home Lab 1

Fernando T.H.L (210167E)
Gamage M.S (210176G)

September 4, 2025

System Information

CPU

- Model: Intel(R) Xeon(R) Processor @ 2.30GHz
- Vendor/Arch: GenuineIntel / x86-64
- Physical cores: 4
- Threads per core: 1
- Caches:
  - L1i: 128 KiB (4 instances)
  - L1d: 128 KiB (4 instances)
  - L2: 1 MiB (4 instances)
  - L3: 45 MiB (1 instance)

Memory / NUMA

- Total (kB): 8150140
- NUMA nodes: 1
- THP: madvise
- Swap total (kB): 0

Operating System

- Distro: Ubuntu 24.04.2 LTS
- Kernel: 6.8.0
- Logical CPUs: 4

Toolchain

- Compiler: gcc (Ubuntu 13.3.0-6ubuntu2 24.04) 13.3.0
- make: GNU Make 4.3
- glibc: glibc 2.39
- libpthread (NPTL): NPTL 2.39
- Python: 3.12.11
- pandas: 2.3.2
- matplotlib: 3.10.6

Approach

We implemented a singly linked list supporting:

- `Member`
- `Insert` (unique keys only)
- `Delete`

Three variants were tested:

- Serial (no locks)
- Pthreads + single mutex
- Pthreads + single read–write lock

Initialization: $n = 1000$ unique keys in $[0, 2^{16} - 1]$. Workloads: $m = 10000$ operations with given fractions, distributed across $T \in \{1, 2, 4, 8\}$ threads. Timing measures only the m-operations region, not initialization.

Experiment Report (Overview Tables)

Case 1: n=1000, m=10000, m_member=0.99, m_insert=0.005, m_delete=0.005

Threads	Serial (µs)	Mutex (µs)	RW-lock (µs)
1	9252.00 ± 127.00	10112.00 ± 459.00	10930.00 ± 243.00
2	9242.00 ± 75.00	32093.00 ± 5094.00	7674.00 ± 250.00
4	8955.00 ± 288.00	30534.00 ± 2223.00	5229.00 ± 446.00
8	8832.00 ± 212.00	29664.00 ± 1443.00	6179.00 ± 1148.00

Table 1: Summary of results for Case 1.

Case 2: n=1000, m=10000, m_member=0.90, m_insert=0.05, m_delete=0.05

Threads	Serial (µs)	Mutex (µs)	RW-lock (µs)
1	17615.00 ± 825.00	18042.00 ± 462.00	29770.00 ± 533.00
2	17107.00 ± 182.00	39649.00 ± 5609.00	30232.00 ± 908.00
4	17372.00 ± 94.00	42669.00 ± 2007.00	22977.00 ± 823.00
8	17664.00 ± 173.00	48538.00 ± 469.00	22568.00 ± 5037.00

Table 2: Summary of results for Case 2.

Case 3: n=1000, m=10000, m_member=0.50, m_insert=0.25, m_delete=0.25

Threads	Serial (µs)	Mutex (µs)	RW-lock (µs)
1	59140.00 ± 540.00	61343.00 ± 512.00	68963.00 ± 2772.00
2	58727.00 ± 661.00	85704.00 ± 6087.00	107904.00 ± 4267.00
4	58024.00 ± 773.00	103935.00 ± 6010.00	120342.00 ± 4358.00
8	57798.00 ± 1239.00	115380.00 ± 2375.00	121688.00 ± 5765.00

Table 3: Summary of results for Case 3.

Sampling/Confidence For Case 1, the worst relative CI was 16.29For Case 2, the worst relative CI was 19.56For Case 3, the worst relative CI was 6.23The target of a 5

Case Analyses with Plots

Case 1: Read-Heavy Workload

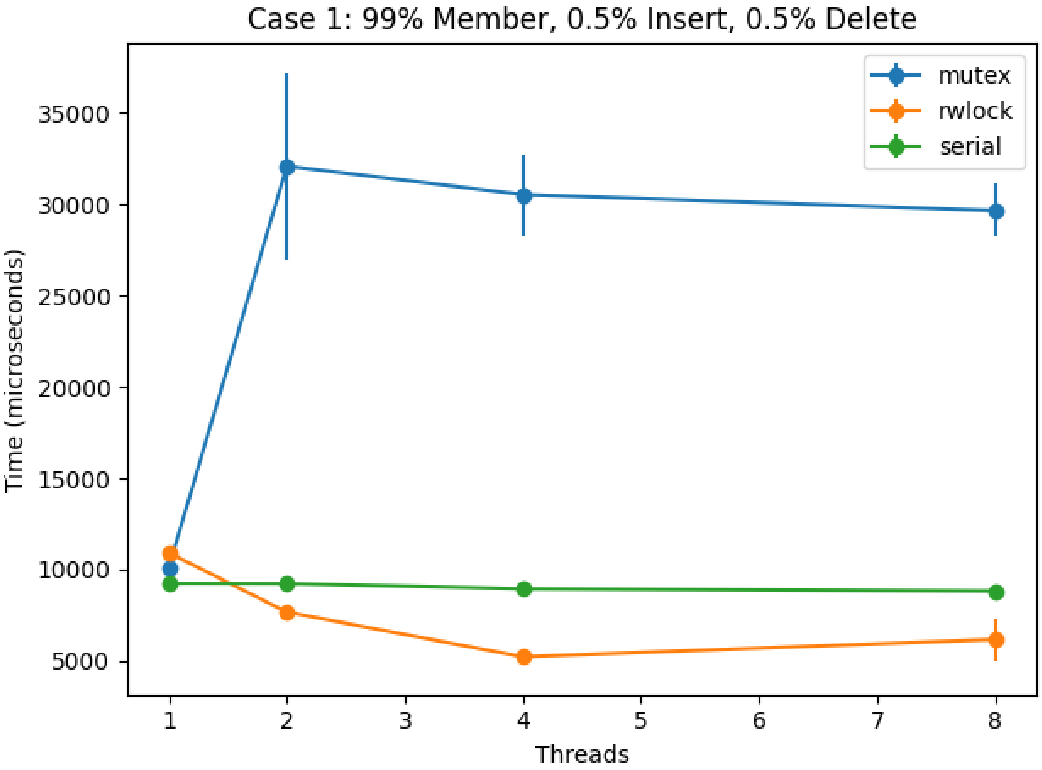

Figure 1: Average time vs. threads for Case 1.

Analysis As shown in Table 1 and Figure 1, at 1 thread, serial is fastest (9252.00µs) vs mutex (10112.00µs) and rw-lock (10930.00µs). From 1 to 8 threads, mutex changes by 193.35At 8 threads, rw-lock is 4.80x faster than mutex. This workload is read-heavy (99

Case 2: Balanced Workload

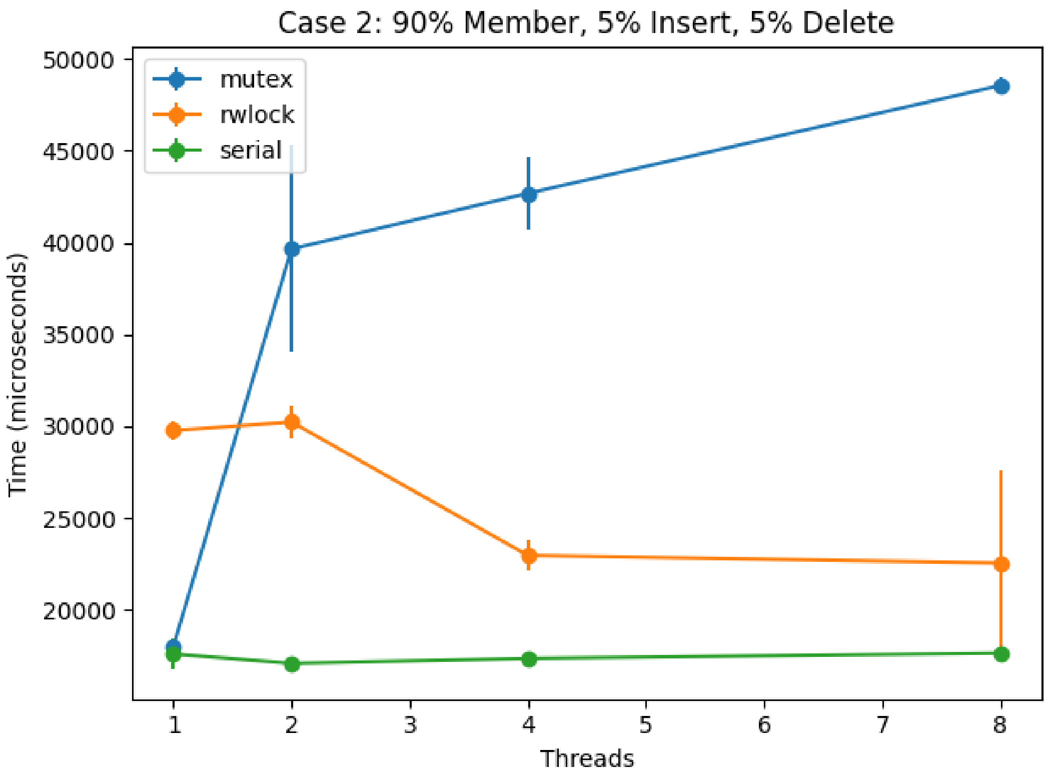

Figure 2: Average time vs. threads for Case 2.

Analysis As shown in Table 2 and Figure 2, at 1 thread, serial is fastest (17615.00µs) vs mutex (18042.00µs) and rw-lock (29770.00µs). From 1 to 8 threads, mutex changes by 169.03At 8 threads, rw-lock is 2.15x faster than mutex. With a higher write fraction (10

Case 3: Write-Heavy Workload

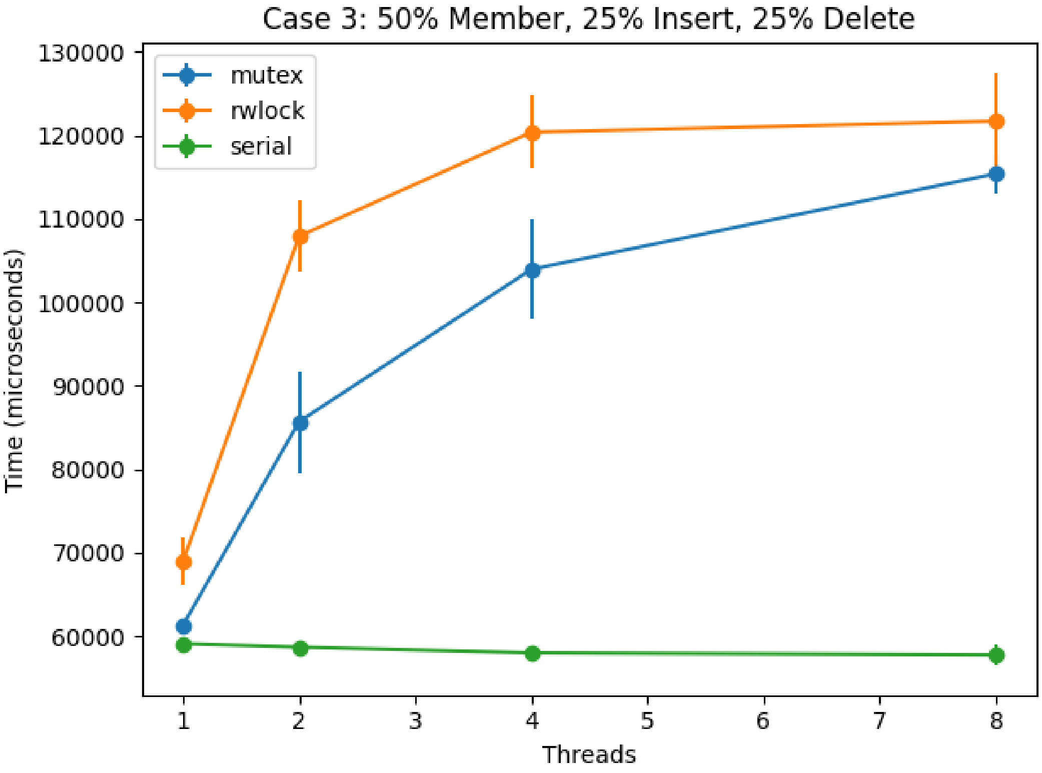

Figure 3: Average time vs. threads for Case 3.

Analysis As shown in Table 3 and Figure 3, at 1 thread, serial is fastest (59140.00µs) vs mutex (61343.00µs) and rw-lock (68963.00µs). From 1 to 8 threads, mutex changes by 88.09At 8 threads, rw-lock is 0.95x faster than mutex. In this write-heavy scenario (50

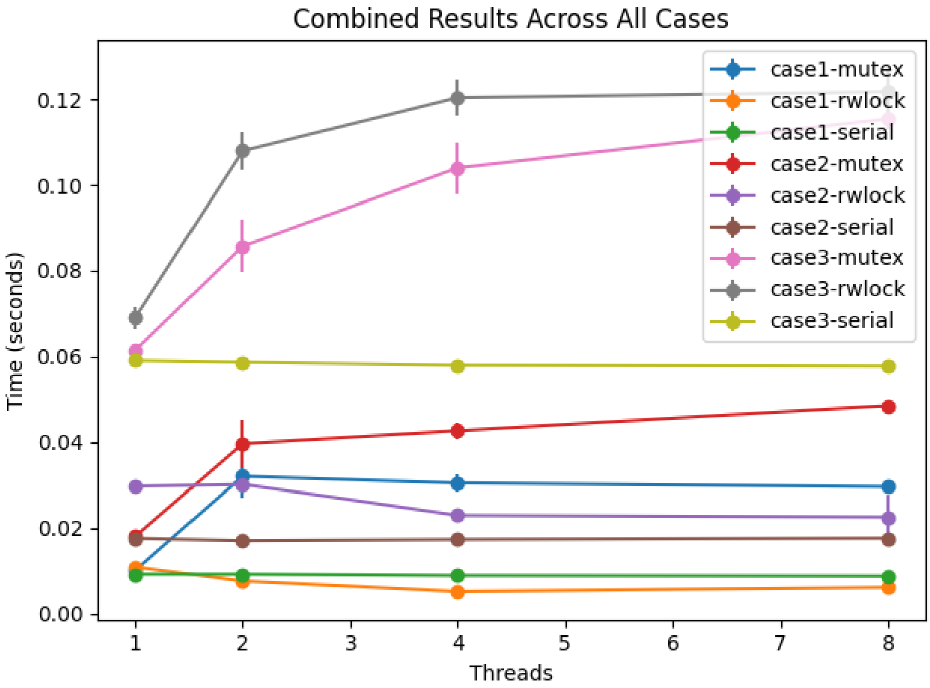

Figure 4: Combined view across all cases and implementations.

Conclusion

Results align with expectations: the serial baseline dominates at T=1 (no lock overhead). Read-heavy workloads: rwlock outperforms mutex via concurrent readers. Write-heavier workloads: rwlock advantage shrinks; both converge due to writer serialization; parallel versions can underperform serial when contention dominates. Scaling saturates near core count due to contention and scheduling overhead. The ±5