Comparison of CLH and MCS Queue Locks
CO21BTECH11004

Implementation: -

- Global variables n, k, lambda1 and lambda2 are created.
- A global array to measure entry time for each thread is created.
- Time is measured in nanoseconds using the Chrono library.
- Global lock object is created.
- Random sleep time is generated using exponential_distribution using random library. Taking lambda1 and lambda2 in millisec.
- For log, a global output file pointer is created, and each thread writes to that output file its log.

CLH Lock: -

- QNode class containing a locked variable is created.
- atomic<QNode*> tail is created.
- Following changes made in book code to implement in C++: -
    - For myNode and myPred, vector<QNode*> is created, and each element in the vector is assigned to one thread for thread-local behavior.
    - threadId is passed in the lock and unlock function to get the corresponding myNode and myPred for the thread.
    - A destructor is also made to free the memory.

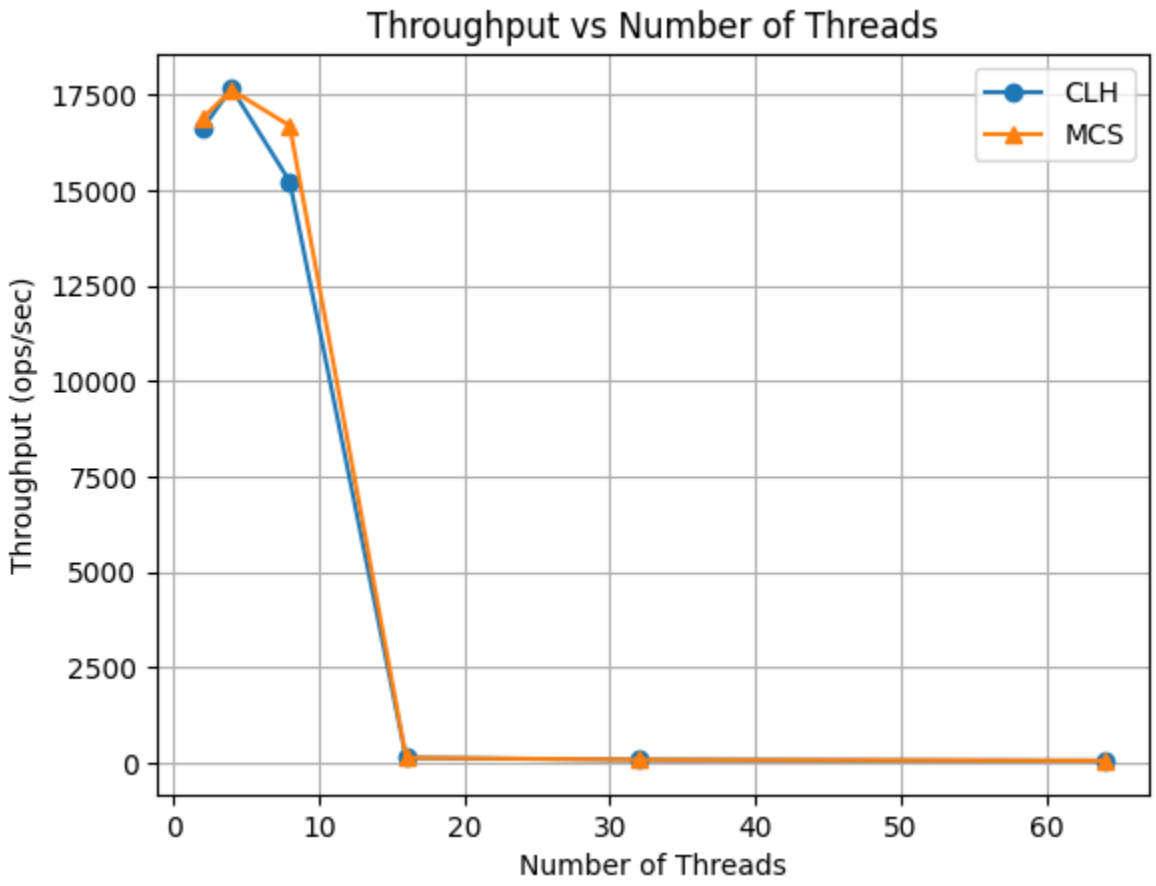
MCS Lock: -

- QNode class containing a locked variable, and a next pointer is created.
- atomic<QNode*> tail is created.
- Following changes made in book code to implement in C++: -
    - For myNode, vector<QNode*> is created, and each element in the vector is assigned to one thread for thread-local behavior.
    - threadId is passed in the lock and unlock function to get the corresponding myNode and myPred for the thread.
    - A destructor is also made to free the memory.

Difficulty faced: -

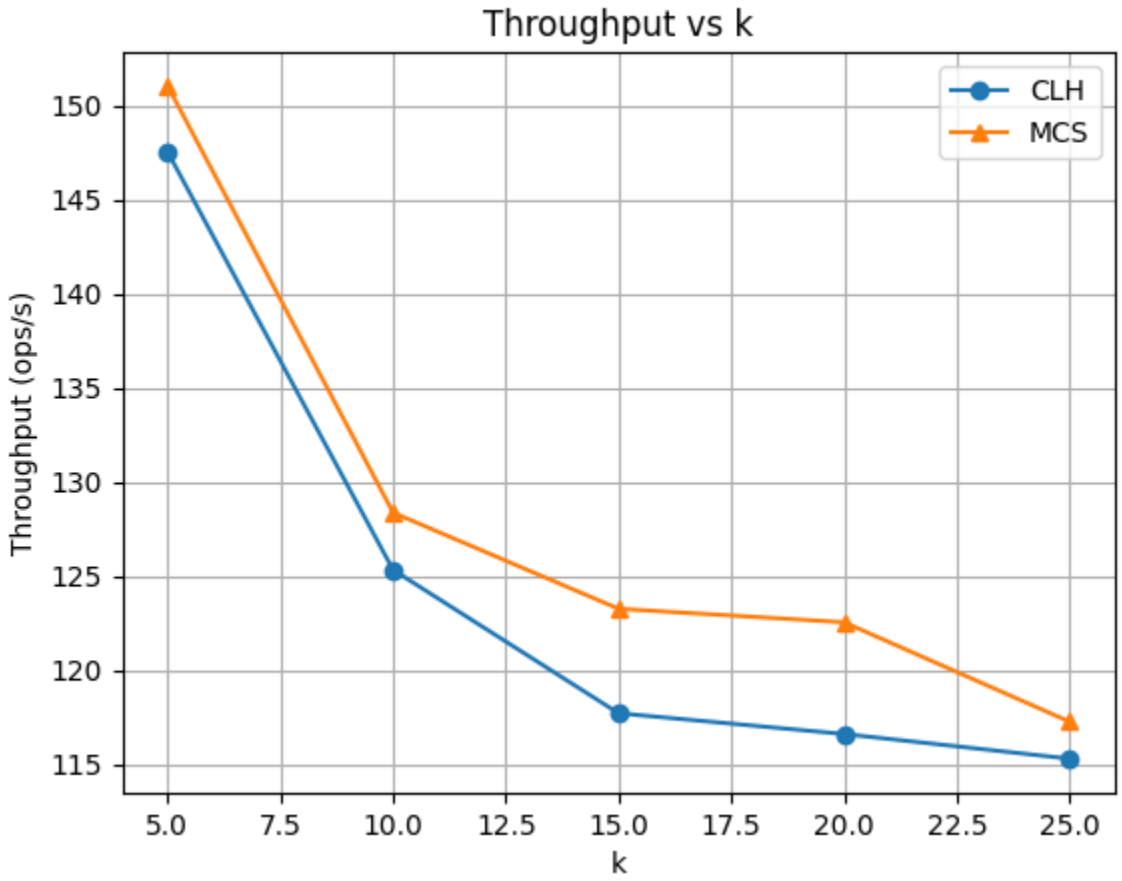
- Implementation of thread-local variables. There is an inbuilt implementation of the thread local variable, but I got some errors while using that.
- For thread-local, I created a vector, and each thread will access one element of the vector that is decided by the thread's id. Thread's ID would work like the index in this case.

Throughput Analysis with varying threads



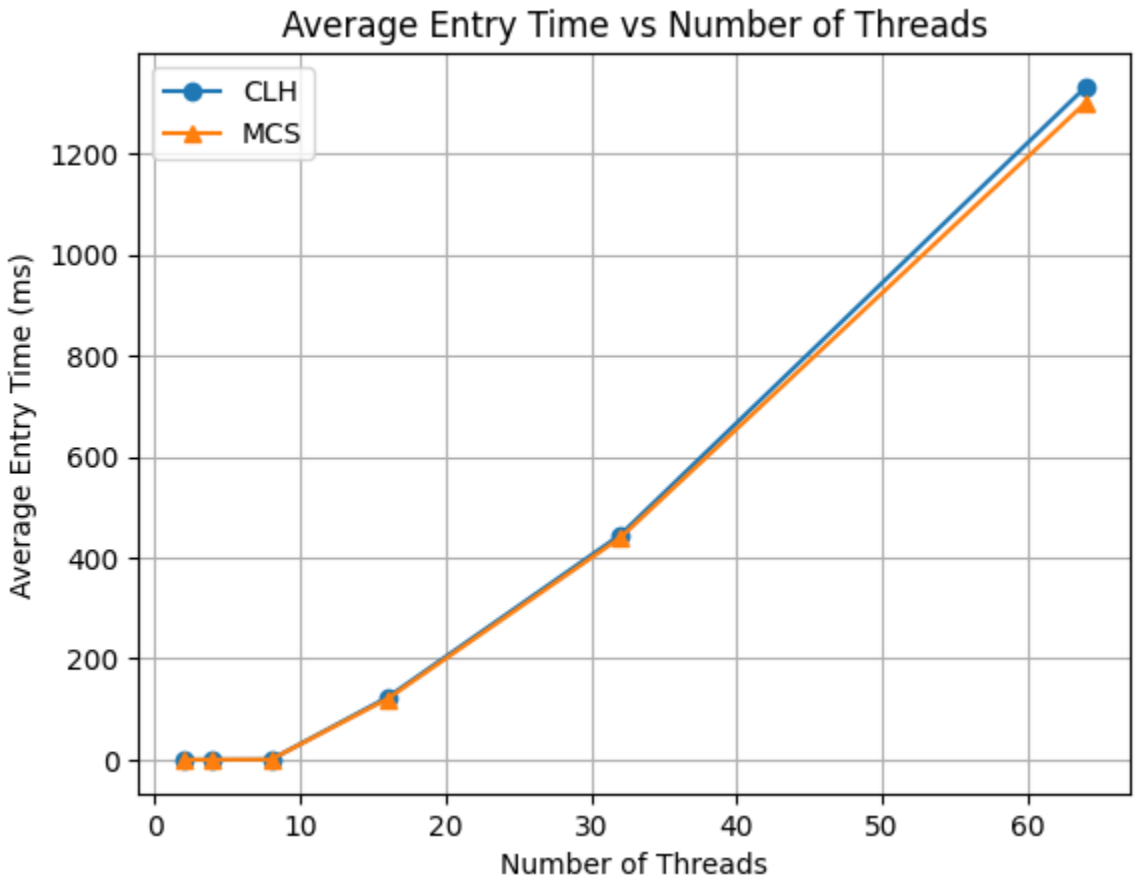
- n is varied; other parameters are: - k=15, lambda1=1, lambda2=2
- As the number of threads increases, throughput decreases for both MCS and CLH locks, as waiting queue for locks increases.
- Graphs almost overlap for MCS and CLH, but if we see values more precisely, MCS has slightly better throughput than CLH as threads in MCS spin on their own node rather than pred in CLH.
- After some thread number (32 here), increasing the number of threads doesn't affect throughput that much.

Throughput Analysis with varying k



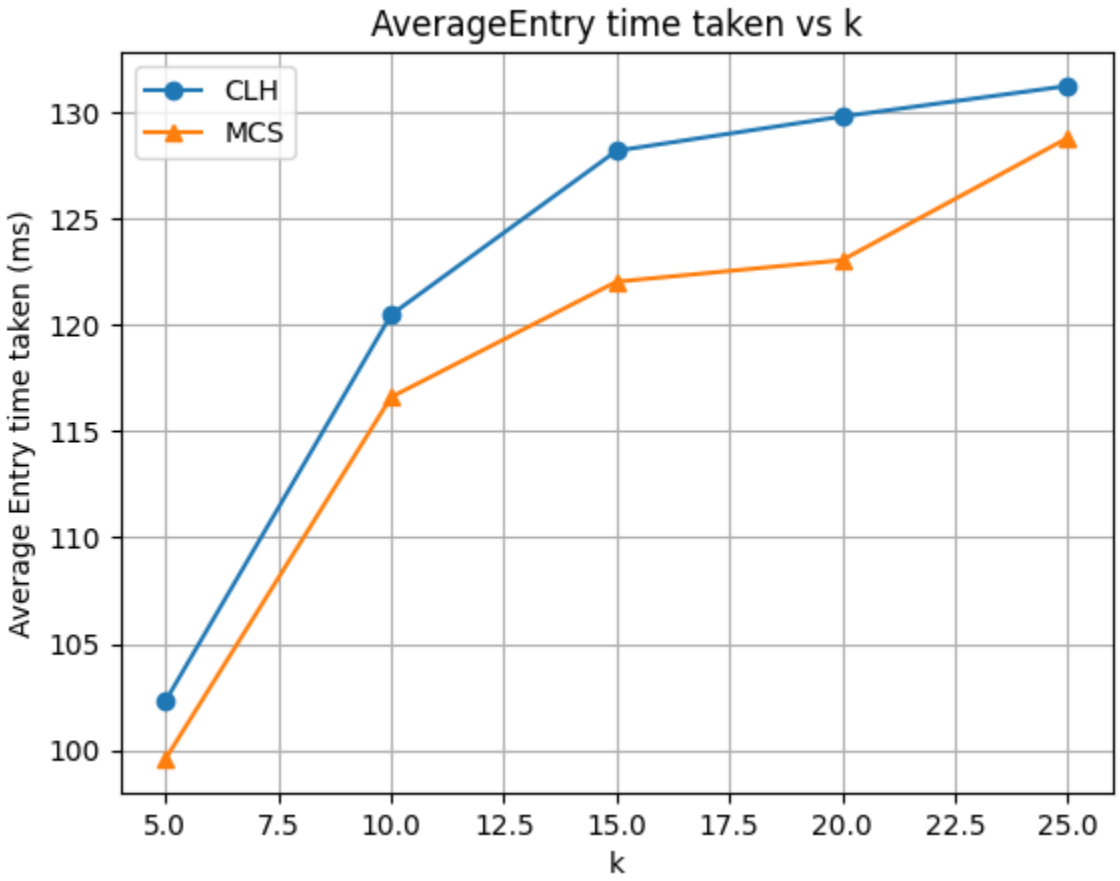
- k is varied; other parameters are: - n=16, lambda1=1, lambda2 = 2.
- Throughput decreases as k increases, but the decrease is not as significant as it decreases when the number of threads increases. Here, values vary from 150-115, while with the increasing number of threads, it varies from 50-17500.
- MCS has higher throughput than CLH because threads spin on its own memory in MCS while on its predecessor memory in CLH. So, MCS has fewer cache misses due to invalidation and high throughput.

Average Entry Time Analysis with varying threads



- n is varied, other parameters: - k = 15, lambda1 = 1, lambda2 = 2
- Average entry time increases significantly as the number of threads increases for both MCS and CLH locks, as waiting queue of threads for acquiring locks increases.
- In the figure, both MCS and CLH plots almost overlap, but by seeing the values more precisely, MCS has slightly less average entry time than CLH locks. Recess for this may be that in MCS, the thread spins on its own node while the CLH thread spins on the predecessor node.

Average Entry Time Analysis with varying k



- k is varied; other parameters are: - n=16, lambda1=1, lambda2 = 2.
- As k increases, the Average entry time taken also increases, but not as significant as it increases with increasing number of threads. Here, it varies from 100 to 130 ms, while with the number of threads, it varies from 4e-3 to 1300.
- CLH has more average entry time than MCS as threads spin on their own node in MCS, while in CLH, they spin on the predecessor node, so cache miss due to invalidation is less.