MULTI-CORE PROGRAMMING

PROJECT – 2

IMPLEMENTATION OF CONCURRENT LISTS

Team Members:

Rahul Aravind Mehalingam(rxm151730)

Praveen Erode Murugesan(pem150030)

Problem Statement:

The aim of the project is to implement the different variations of Concurrent Linked List such as

I. Coarse-Grained
II. Fine-Grained (Lazy Synchronization)
III. Lock-Free

Experimental Setup:

CPU cores: 16

Operating System: CentOS

Programming Language: Java

No of threads experimented: 2 to 32

Correctness Test Strategy:

Verifying the correctness of the concurrent data structure proved to be a NP-Hard problem. Therefore, the consistency of the implemented lists is tested manually. Each list is subjected to 10 operations for 32 threads. Since this experiment have a very few entries in the list, it is easy to check the consistency of the list and it proved to be correct which is evident from the included screenshot of the sample output.

Note - Sample output included at the last

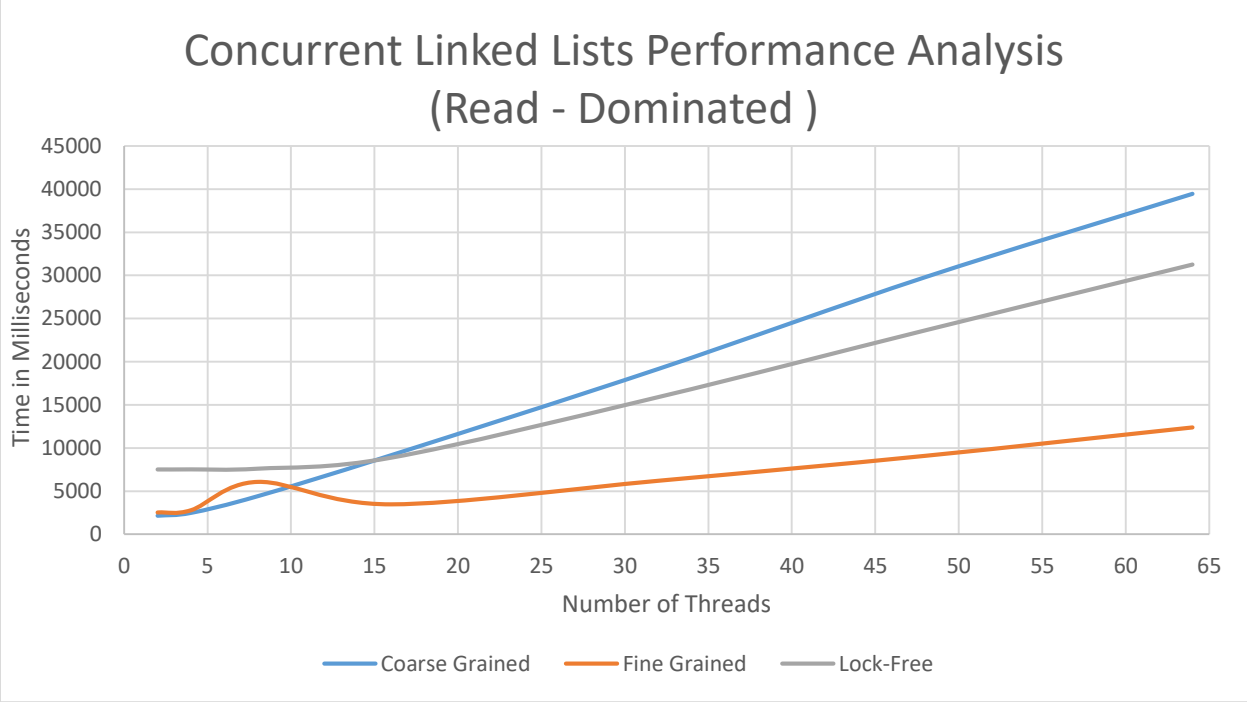
Testing in TACC Machine:

The implemented concurrent linked lists are tested with high load and their results are tabulated and pictographically represented as follows. Each list is subject to Number of Threads X 1Million operations by keeping the key space bounded to 1000. The performances of the algorithms are analyzed by varying the number of threads.

Read Dominated:

The read dominated flavor of test setup had 90% search queries, 9% insert queries and 1 % delete queries. The performance is as follows:

Number of threads	Performance of Algorithms in milliseconds		
	Coarse Grained	**Fine Grained**	**Lock-Free**
2	2163	2530	7515
4	2495	2789	7524
8	4427	6075	7606
16	9178	3471	8883
32	19163	6204	15901
48	29798	9095	23640
64	39441	12385	31259



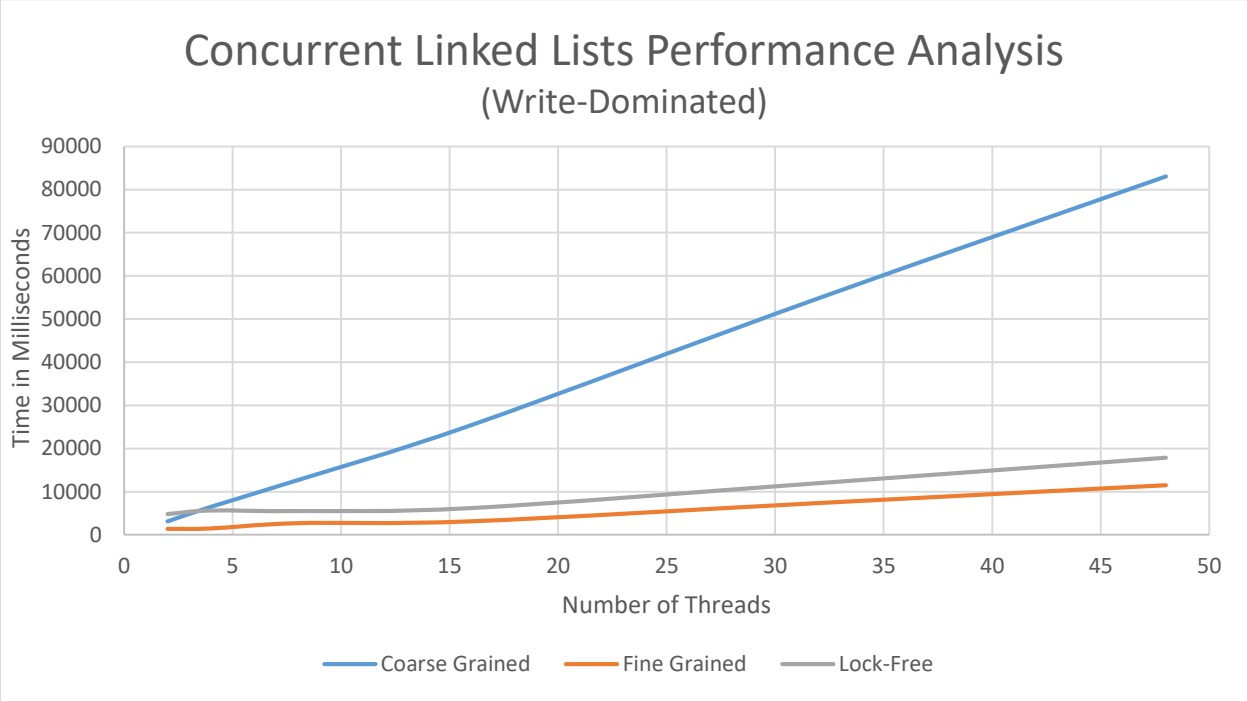
Inference:

It is seen that fine grained algorithm performs better if the data structure needs to be used in read dominated scenarios

Write Dominated:

In a write-dominated flavor of test set up, the data structure is subjected to 50% insert queries and 50% delete queries.

Number of threads	Performance of Algorithms in milliseconds		
	Coarse Grained	Fine Grained	Lock-Free
2	3128	1347	4836
4	6506	1450	5651
8	12689	2678	5523
16	25436	3113	6220
32	54827	7364	11995
48	83036	11515	17886



Inference:

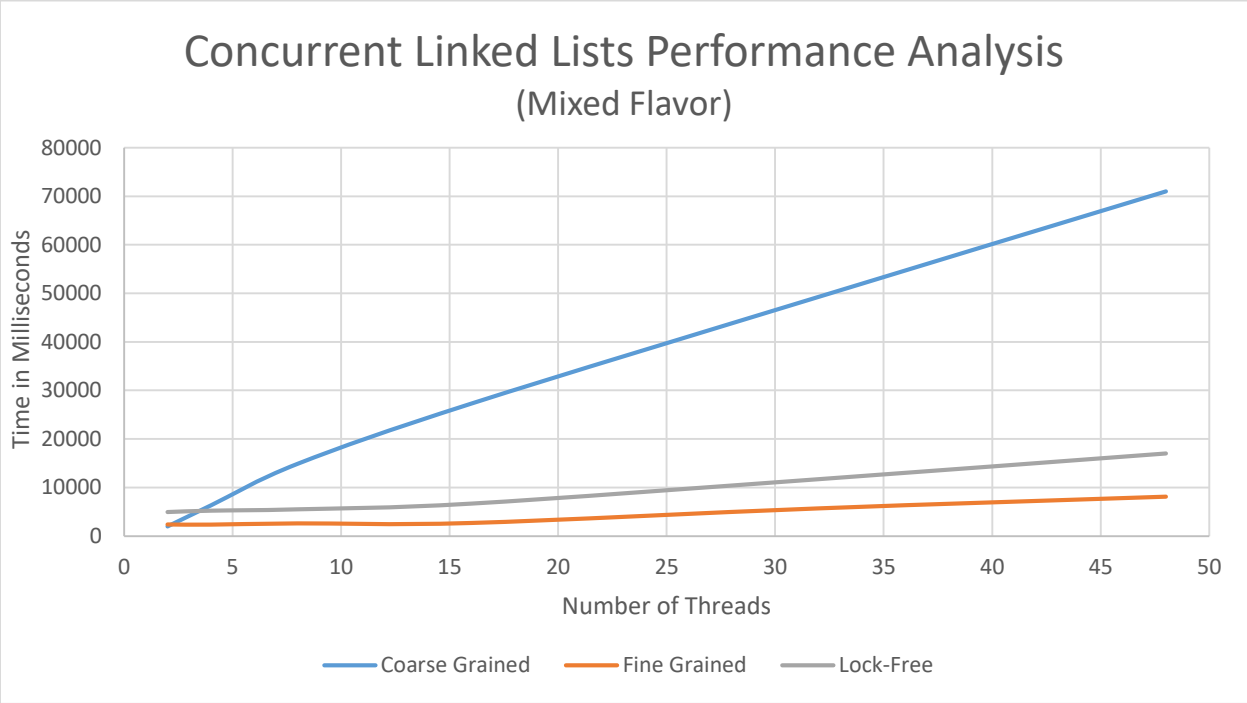
It is seen that fine-grained algorithm beats the lock-free algorithm in performance by a margin and it is way better than coarse-grained algorithm.

Mixed-Flavor:

In a mixed-flavor test setup the concurrent linked list is subjected to 70% search , 20% insert and 10% delete queries.

Number of threads	Performance of Algorithms in milliseconds		
	Coarse Grained	Fine Grained	Lock-Free
2	2004	2413	4924
4	6338	2382	5205

8	14904	2633	5487
16	27293	2722	6665
32	49263	5729	11703
48	70975	8137	17008



Inference:

It is seen that fine-grained algorithm beats the lock-free algorithm in performance by a margin and it is way better than coarse-grained algorithm.

Conclusion:

The concurrent linked lists are implemented using different algorithms are implemented and their performances are analyzed and compared to infer the best suited implementation for different workloads.

Sample output for correctness test strategy:

Threads: 32

Number of operations: 10

Flavor: Mixed

run:

Coarse-Grained Locking

Inserting 6

Deleting 3

Deleting 3

Inserting 1

Deleting 5

Inserting 4

Inserting 8

Inserting 3

Inserting 5

Inserting 7

Deleting 5

Inserting 3

Inserting 1

Inserting 7

Deleting 2

Inserting 6

Inserting 5

Inserting 8

Inserting 5

Deleting 8

Deleting 3

Inserting 5

Inserting 1

Inserting 3

Inserting 2

Inserting 3

Inserting 3

Deleting 8

Inserting 7

Deleting 8

Deleting 2

Deleting 2

Deleting 7

Inserting 0

Inserting 0

Deleting 2

Inserting 4

Inserting 9

Inserting 4

Inserting 1

Inserting 5

Inserting 9

Inserting 2

Inserting 3

Deleting 5

Inserting 6

Deleting 6

Inserting 9

Inserting 8

Inserting 7

Inserting 1

Deleting 8

Deleting 7

Inserting 8

Inserting 3

Deleting 2

Inserting 2

Inserting 0

Inserting 4

Inserting 5

Deleting 7

Deleting 8

Inserting 0

Inserting 1

Inserting 8

Inserting 8

Deleting 3

Inserting 0

Inserting 7

Inserting 6

Inserting 1

Deleting 2

Inserting 2

Inserting 7

Inserting 0

Inserting 7

Deleting 7

Deleting 0

Inserting 6

Deleting 5

Inserting 6

Deleting 1

Inserting 6

Inserting 4

Inserting 7

Deleting 8

Inserting 7

Inserting 3

Deleting 9

Inserting 1

Time: 0 msec.

Memory: 17 MB / 96 MB.

1 ---> 2 ---> 3 ---> 4 ---> 6 ---> 7

Fine-Grained Locking

Inserting 3

Deleting 4

Inserting 6

Inserting 2

Inserting 7

Deleting 2

Deleting 9

Deleting 9

Inserting 5

Deleting 1

Inserting 0

Inserting 0

Inserting 3

Deleting 9

Deleting 5

Deleting 2

Deleting 7

Inserting 6

Inserting 3

Inserting 8

Inserting 6

Inserting 2

Inserting 0

Deleting 1

Inserting 1

Deleting 3

Deleting 1

Inserting 9

Inserting 5

Deleting 9

Inserting 8

Deleting 2

Deleting 6

Inserting 6

Inserting 2

Inserting 3

Deleting 9

Inserting 6

Inserting 8

Inserting 6

Inserting 7

Deleting 3

Inserting 2

Deleting 6

Deleting 7

Inserting 1

Inserting 0

Inserting 6

Deleting 1

Inserting 6

Inserting 4

Inserting 6

Inserting 3

Inserting 4

Deleting 6

Inserting 0

Inserting 6

Inserting 4

Inserting 3

Inserting 4

Inserting 0

Inserting 5

Inserting 5

Inserting 5

Inserting 5

Deleting 3

Deleting 8

Deleting 4

Inserting 5

Deleting 9

Inserting 5

Inserting 5

Deleting 1

Inserting 0

Deleting 6

Deleting 9

Deleting 5

Deleting 5

Inserting 3

Deleting 8

Deleting 1

Deleting 7

Inserting 4

Inserting 3

Inserting 2

Deleting 1

Inserting 6

Inserting 4

Inserting 7

Time: 0 msec.

Memory: 8 MB / 96 MB.

0 ---> 2 ---> 3 ---> 4 ---> 6 ---> 7

Lock-Free Algorithm

Inserting 2

Deleting 2

Inserting 9

Deleting 0

Deleting 2

Inserting 8

Inserting 6

Deleting 0

Deleting 4

Inserting 8

Deleting 9

Inserting 1

Deleting 8

Inserting 3

Deleting 3

Deleting 3

Deleting 2

Deleting 6

Inserting 0

Inserting 3

Inserting 6

Inserting 9

Deleting 1

Deleting 7

Inserting 3

Inserting 7

Inserting 2

Inserting 9

Inserting 0

Inserting 0

Inserting 5

Deleting 9

Inserting 0

Inserting 8

Inserting 5

Inserting 8

Inserting 3

Inserting 1

Inserting 1

Inserting 3

Inserting 2

Inserting 5

Inserting 3

Inserting 9

Inserting 1

Deleting 6

Deleting 5

Inserting 5

Deleting 3

Deleting 0

Inserting 8

Deleting 0

Inserting 7

Deleting 5

Inserting 0

Inserting 4

Deleting 4

Inserting 3

Inserting 8

Inserting 3

Inserting 7

Deleting 0

Deleting 2

Inserting 2

Deleting 9

Inserting 1

Inserting 6

Inserting 5

Inserting 6

Inserting 5

Inserting 9

Inserting 7

Inserting 3

Deleting 6

Deleting 7

Inserting 0

Deleting 4

Inserting 6

Inserting 4

Inserting 9

Inserting 3

Inserting 6

Inserting 1

Deleting 5

Inserting 0

Deleting 6

Deleting 6

Deleting 4

Inserting 3

Deleting 1

Deleting 7

Deleting 3

Deleting 7

Deleting 4

Deleting 4

Inserting 2

Deleting 5

Deleting 5

Time: 16 msec.

Memory: 14 MB / 96 MB.

0 ---> 2 ---> 8 ---> 9

BUILD SUCCESSFUL (total time: 0 seconds)

The consistency of the lists are checked manually and it is observed that lists got constructed in an ascending order.

Concurrency Correctness:

The concurrency correctness of the algorithm is also checked automatically by the testCorrectness function call that is implemented for each locking algorithm.