Concurrent Linked List Performance Analysis

Project Members: Srikanth Kannan
Giridara Varma Elumalai

Abstract

The purpose of the experimentation is to implement and compare operations like search, insert and delete in a concurrent linked list using the following three approaches

a. coarse-grained locking
b. fine-grained locking using lazy synchronization
c. lock-free synchronization

Methodology

The implementation is done in java. An interface IConcurrentList is defined with following four methods.

a. add
b. remove
c. contains
d. checkList

The interface is implemented by following classes

a. CoarseGrainedList.java
b. FineGrainedList.java
c. LockFreeList.java

ApplicationThread.java will call above methods defined by IConcurrentList interface based on given probability distribution which is passed as parameter.

Experimental Setup

The application expects three optional arguments. Following are the information about the arguments.

a. Argument 0 – Refers to maximum number of threads that is used in the application. For example if this argument value is passed as 8, then application will run for given probability distribution (argument 2) for 2 threads, then do the same for 4 threads, followed by 6 and 8 threads.
b. Argument 1 – Refers to maximum number of operations. If value is not passed, then by default 1000000 is used.
c. Argument 2 – Refers to relative distribution of various operations. It is given as comma separated values. First value in comma separated refers to search, followed by add and then delete operation. For example, if the intention is to test Read-Dominated work load, then input can be given as 90,9,1. If the intention is to test mixed domination work load, then input can be given as 70,20,10.

Our experimentation is done for maximum number of threads as 32(argument 0). Application will run all three locking approaches with number of threads as 2, then 4, followed by 6,8,10,12,16,18,20,22,24, 26,28,20 and 32. Maximum number of operations (argument 1) is set as 1000000 in all runs.
For First run, distribution parameter (argument 2) is set as 90,9,1. Then application is run for 70,20,10 and finally for 0,50,50.
Below is the information about the machine in which experiment was performed

RAM Size:	32 GB
Operating System:	Cent OS
Programming Language:	Java

Testing Strategy

I. Manual Testing

ApplicationThread.java is added with static set of operations and it was run for different work load configurations with 2 to 8 threads over key space of ten. The output of every operation is logged in log file and validity of outputs are checked manually.

Sample set of static operations

```
getConcurrentList().add(10);
getConcurrentList().add(20);
getConcurrentList().add(30);
getConcurrentList().add(40);
getConcurrentList().add(50);
getConcurrentList().remove(40);
getConcurrentList().remove(30);
getConcurrentList().contains(30);
getConcurrentList().contains(20);
```

Output

Printing Linked list:
10 20 50

II. Automated Testing

The consistency of the lists are verified by checking that the keys are in order and there are no duplicates in the list. The checklist method iterates over the entire list and asserts the correctness. Then the final list is also printed for manual cross verification.

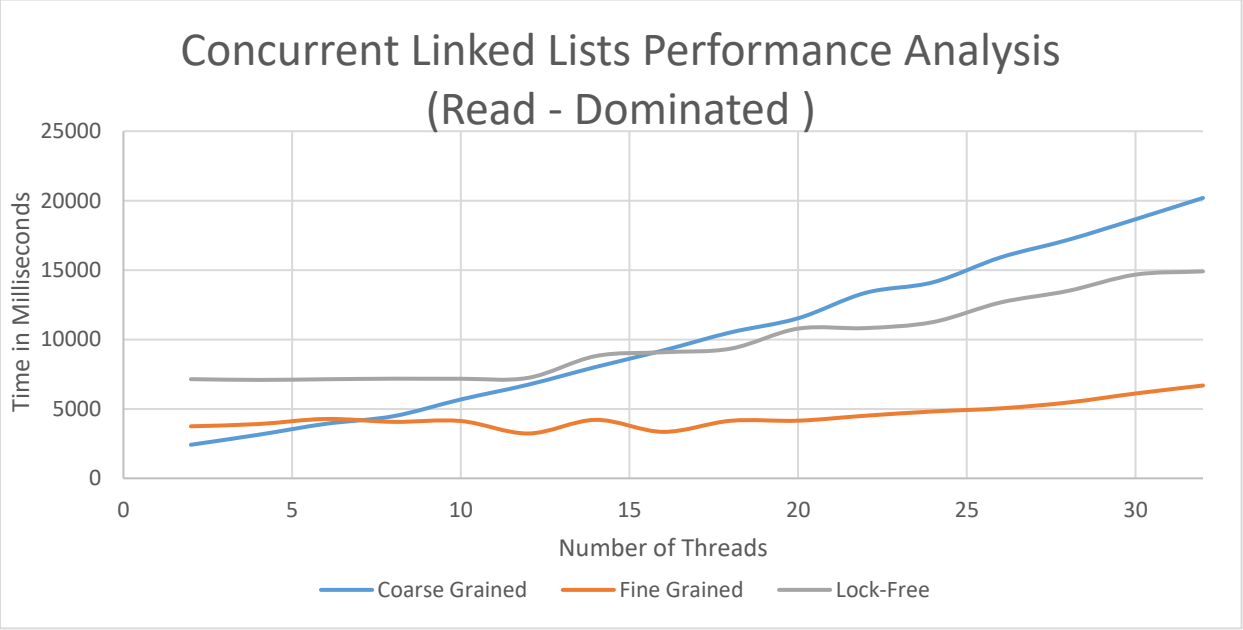
Sample Output:
CoarseGrainedList: 0->1->2->4->8->9->|
FineGrainedList: 0->2->4->5->6->8->|
LockFreeList: 1->2->3->4->7->9->|
RunTime 2: 374,465,327
CoarseGrainedList: 0->6->|
FineGrainedList: 1->3->4->5->7->9->|
LockFreeList: 0->1->3->6->7->8->|
RunTime 4: 647,709,449

Test Result

For all the below tests, the number of operations is fixed as 1000000.

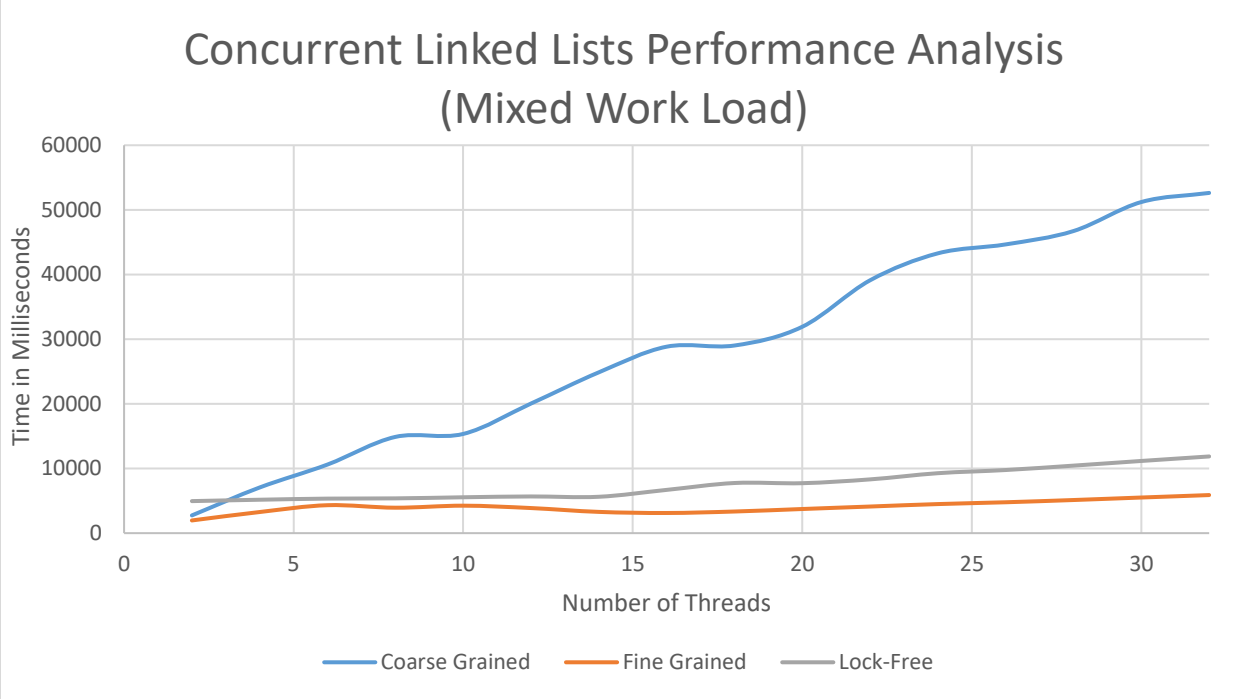
Read-Dominated Work Load (90% Search, 9% Insert and 1% Delete):

Number of Threads	Coarse Grained(ms)	Fine Grained(ms)	Lock Free(ms)
2	2419	3747	7143
4	3139	3910	7089
6	3928	4277	7136
8	4475	4068	7178
10	5681	4129	7171
12	6749	3231	7239
14	8020	4214	8806
16	9211	3347	9082
18	10512	4147	9341
20	11528	4153	10784
22	13361	4512	10820
24	14121	4817	11257
26	15911	5039	12665
28	17174	5462	13500
30	18660	6111	14670
32	20194	6686	14905



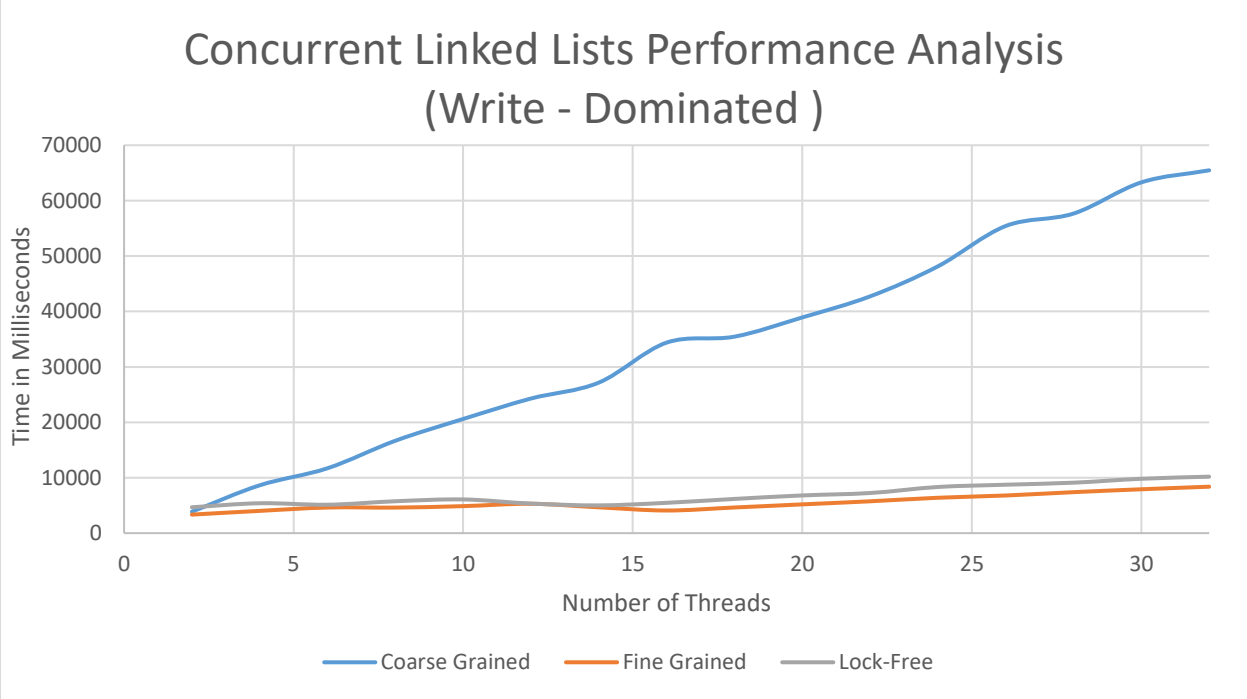
Mixed Work Load (70% Search, 20% Insert and 10% Delete):

Number of Threads	Coarse Grained(ms)	Fine Grained(ms)	Lock Free(ms)
2	2746	1964	4957
4	7065	3269	5168
6	10616	4323	5341
8	14888	3935	5378
10	15345	4251	5547
12	20045	3881	5677
14	24872	3287	5628
16	28842	3119	6690
18	29022	3346	7755
20	31920	3736	7727
22	39109	4118	8315
24	43281	4499	9272
26	44663	4778	9763
28	46730	5126	10446
30	51218	5510	11165
32	52625	5890	11864



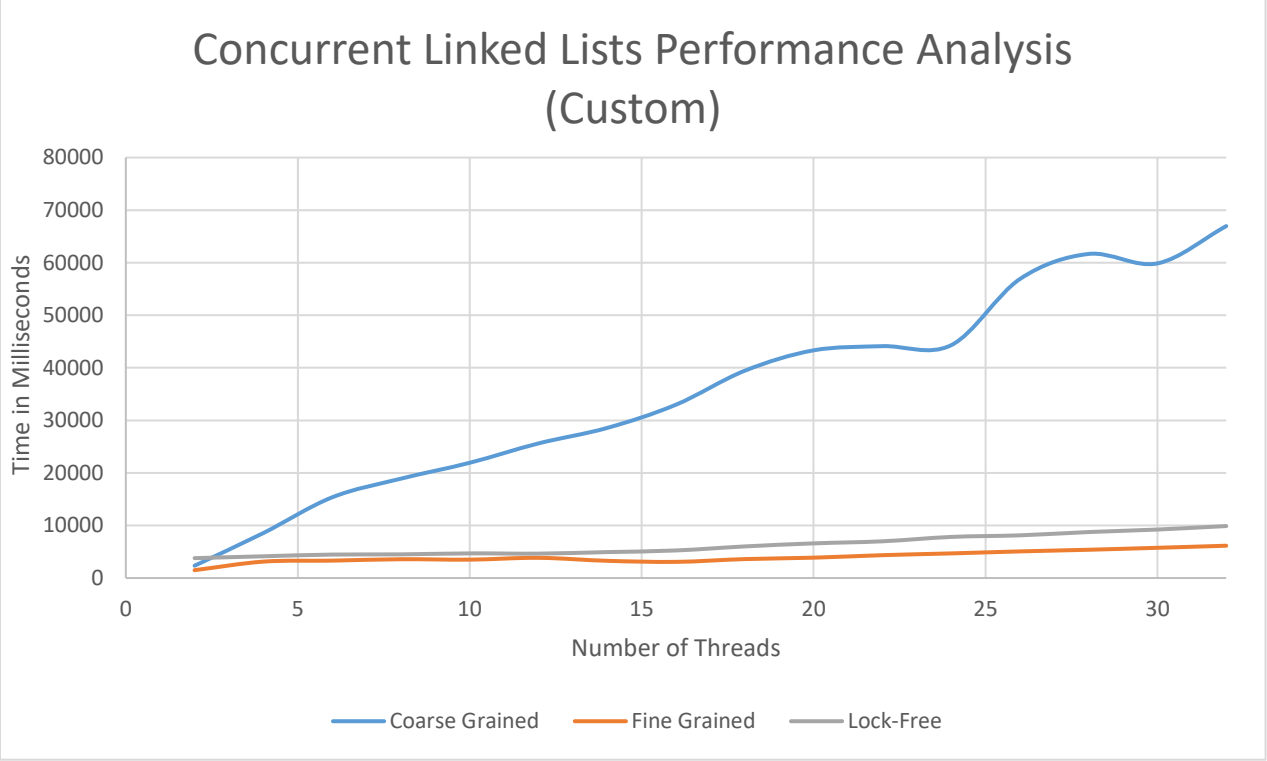
Write-Dominated Work Load (0% Search, 50% Insert and 50% Delete):

Number of Threads	Coarse Grained(ms)	Fine Grained(ms)	Lock Free(ms)
2	3869	3346	4673
4	8636	4021	5398
6	11712	4628	5129
8	16675	4614	5758
10	20601	4874	6083
12	24293	5302	5340
14	27165	4662	5013
16	34384	4088	5471
18	35459	4629	6175
20	38925	5194	6806
22	42728	5745	7245
24	48144	6386	8329
26	55428	6797	8740
28	57670	7397	9106
30	63299	7919	9815
32	65475	8385	10192



Custom Work Load (50% Search, 25% Insert and 25% Delete):

Number of Threads	Coarse Grained(ms)	Fine Grained(ms)	Lock Free(ms)
2	2347	1495	3781
4	8562	3119	4136
6	15354	3309	4459
8	18908	3580	4501
10	21928	3496	4690
12	25612	3843	4643
14	28560	3275	4935
16	32985	3077	5245
18	39445	3605	6011
20	43321	3871	6594
22	44118	4348	6988
24	44295	4686	7818
26	56896	5064	8126
28	61665	5381	8750
30	59868	5756	9226
32	66960	6144	9891



Conclusion

Following are the observations noticed from the results of each work load type.

Read Dominated Work Load:

Fine grained lock is performing far better than lock free synchronization. Coarse grained lock performance is worse of the three. The time taken by coarse grained lock is proportionately increasing with number of threads.

Mixed Work Load:

Fine grained lock is still winning the performance race, but the difference in time taken by lock free synchronization and fine grained lock is less compared to read dominated work load configuration. Coarse grained lock performs poorly as before.

Write Dominated Work Load:

Fine Grained lock and lock free synchronization takes almost same time with former slightly better than lock free synchronization. Coarse grained lock performs poorly as before.

Custom Work Load:

Custom work load (70% search, 25 % add, and 25% write) provides almost same output as write dominated work load.

Finally, it is observed that in all cases Fine grained lock outperformed in the experimentation.