Distributed Assignment 3

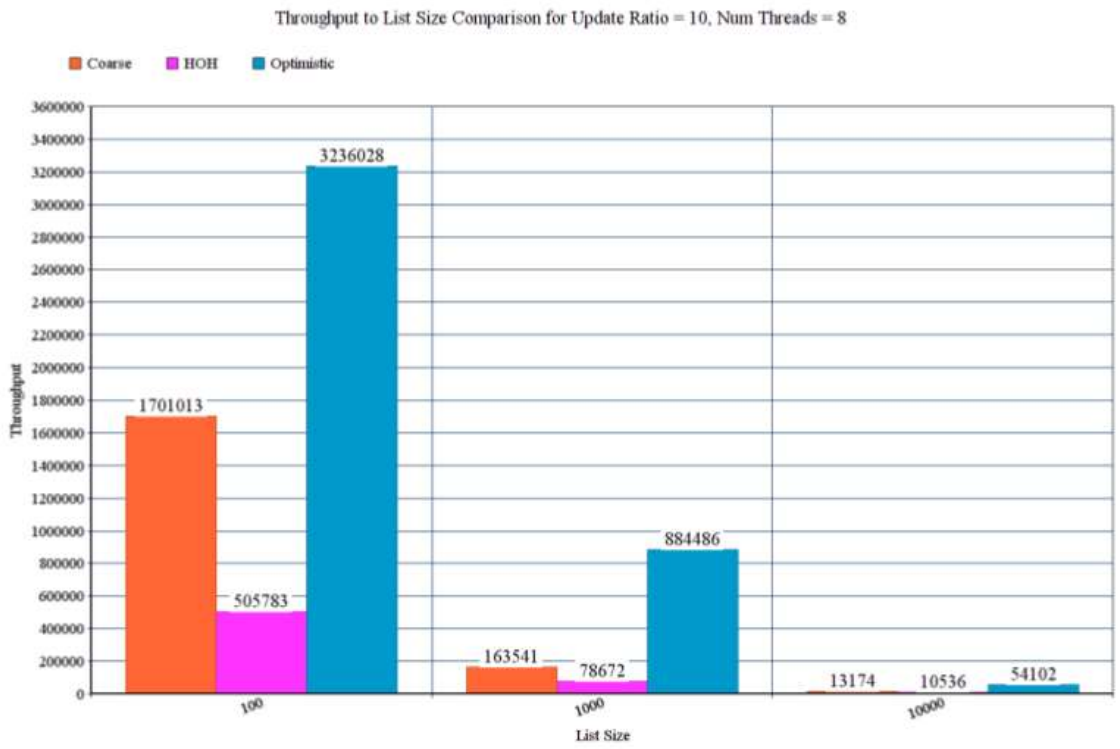
Names: Devesh Kumar Singh
Vaibhav Jain

Complete Results:

Machine= mc18.cs.purdue.edu. No. of cores = 32

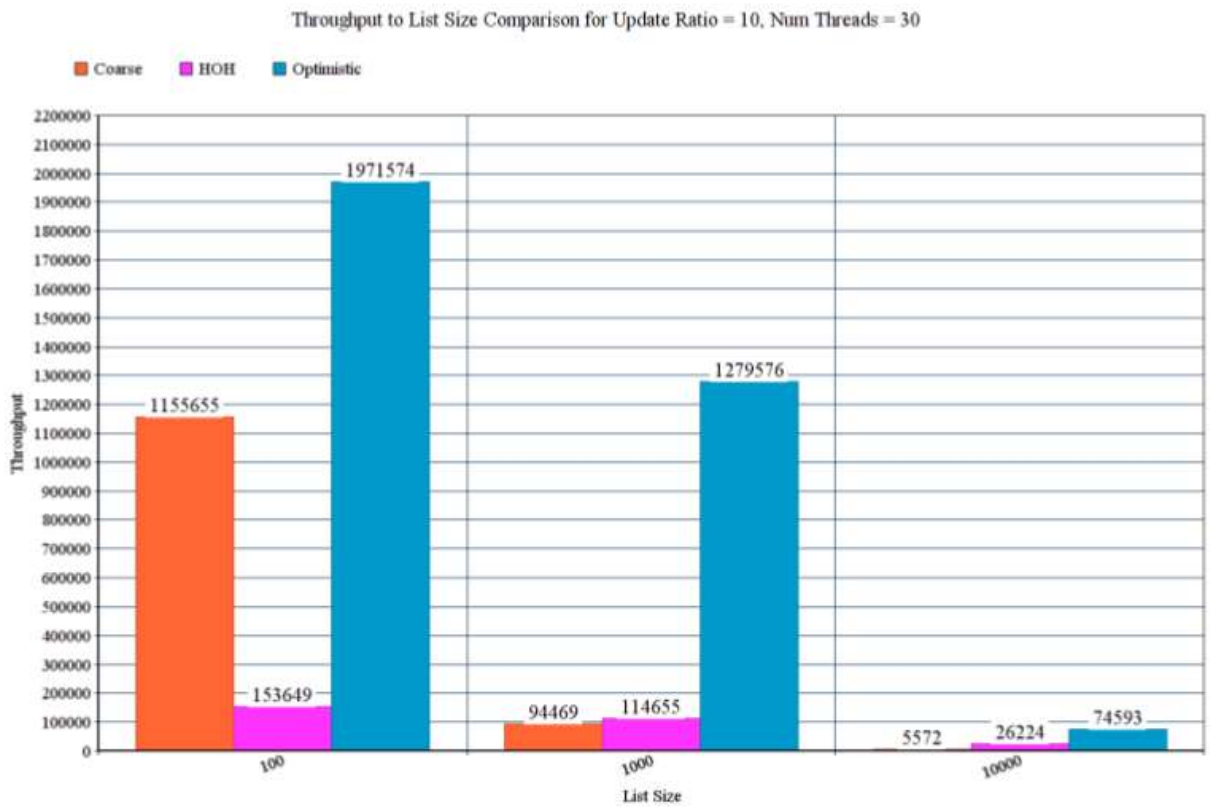
1) Threads = 8, Duration = 5s

Scheme	Update Ratio	Initial List Size	Finished Calls	Throughput	Insert_success	Insert_fail	Contain_success	Contain_fail	Remove_success	Remove_fail
Coarse	**10**	**100**	8504938	1701013	212942	211829	3815386	3838283	212944	213846
		1000	817697	163541	20425	20190	365560	370534	20380	20634
		10000	65872	13174	1613	1633	29477	29882	1651	1618
	100	**100**	4755961	951219	1188552	1188746	0	0	1188578	1190356
		1000	624055	124812	155969	156214	0	0	155984	155911
		10000	53891	10778	13482	13510	0	0	13395	13505
HOH	**10**	**100**	2528894	505783	63988	63754	1144724	1142559	63988	63060
		1000	393358	78672	9940	9855	179197	175223	9937	9936
		10000	52679	10536	1322	1343	23619	23837	1322	1263
	100	**100**	840517	168107	211599	211497	0	0	211550	211488
		1000	555490	111098	139738	139894	0	0	139729	138262
		10000	48668	9733	12212	12088	0	0	12200	12197
Optimistic	**10**	**100**	16179914	3236028	475592	476681	8111392	8097472	475585	474532
		1000	4422392	884486	115643	115500	2050522	2048200	115633	115967
		10000	270510	54102	6867	6716	122558	121449	6871	6655
	100	**100**	11278453	2255708	3083668	3098062	0	0	3079599	3103929
		1000	3891420	778294	1002532	1003742	0	0	1001302	1002423
		10000	259274	51855	65109	65087	0	0	65135	64730



2) Threads = 30, Duration = 5s

Scheme	Update Ratio	Initial List Size	Finished Calls	Throughput	Insert_success	Insert_fail	Contain_success	Contain_fail	Remove_success	Remove_fail
Coarse	**10**	**100**	5778164	1155655	145453	143947	2597819	2600655	145468	145113
		1000	472340	94469	11803	11827	213123	212108	11809	11691
		10000	27861	5572	698	715	12497	12581	701	670
	100	**100**	4359064	871830	1089448	1090803	0	0	1089461	1089529
		1000	529172	105836	132119	132392	0	0	132117	132566
		10000	30547	6109	7630	7717	0	0	7739	7463
HOH	**10**	**100**	768226	153649	19304	19268	346402	349609	19296	19408
		1000	573279	114655	14371	14704	260890	257754	14380	14247
		10000	131120	26224	3238	3298	58980	59137	3272	3328
	100	**100**	738519	147707	186248	186044	0	0	186218	185810
		1000	551731	110348	139353	138333	0	0	139326	138771
		10000	140343	28068	35205	35097	0	0	35257	34990
Optimistic	**10**	**100**	9857741	1971574	286970	286743	4925249	4932223	286976	286867
		1000	6397774	1279576	175059	175384	3057171	3047139	174999	175070
		10000	374758	74953	9493	9434	169614	168721	9518	9596
	100	**100**	5209727	1041973	1420162	1428891	0	0	1419113	1432574
		1000	10316955	2063417	2984319	2984541	0	0	2981998	2988266
		10000	664224	132846	167291	167153	0	0	167352	167240



Summary:

1. In general, **as the list size increases, we see a drop in throughput**. The reason is increase in traversal time. For eg: insert(18000) in 10000 list size(range is twice the list size) will have to traverse many more nodes than insert(199) in 100 nodes list. This increase in traversal time proportionally increase the completion time of any process which implies more delay for waiting processes leading to lesser throughput. In HOH, increase in list size will also increase the lock-unlock time.
2. **When the update ratio is increased, we see reduction (very small in some) in throughput**. Although, lock is acquired for complete list in all operations, contains operation does not involve modifying any pointers like insert or delete and simply returns true or false depending on node present or absent in list. Hence, contains will complete a little faster, giving chance to other processes waiting for locks. However, the difference is negligible as the saving is dwarfed by traversal and waiting delay for locks.
3. **Coarse Grained V/s Hand Over Hand V/s Optimistic Locking**
   Ideally, we expected throughput for Coarse grained < Hoh < Optimistic. In coarse grained list, lock is acquired for the complete list for any operation. Thus, all other processes have to wait for the lock until the process who has lock finished. In hand over hand, lock is only acquired for current and predecessor nodes. Thus, any other process who needs to work on other nodes is not blocked, provided that there are no locks on the way (locks are acquired in head to tail order). In Optimistic, no locks are acquired during the traversal and locks are only acquired for nodes at which operation has to take place (current and predecessor). Thus, probability of a process being blocked in coarse grained is more, followed by HOH, followed by Optimistic. More the delay in waiting for locks,

lesser will be the throughput, which implies lesser number of total insert, remove and contains operations performed by all threads. However, we didn't get the expected results for 8 threads. We observed:

HOH < Coarse grained < Optimistic.

We think this is because of the time spent on locking and unlocking nodes. For eg : insert(200) may have to lock and unlock 200 nodes, whereas coarse grained will lock and unlock only once. This may have overshadowed the time we saved by concurrent operations who operated on lesser value nodes and were not blocked.

To confirm our suspicion, we ran the experiment with 30 threads and found that HOH performs better than Coarse for list size = 1000 and 10000. The locking overhead was surpassed with the parallel optimization HOH gives over coarse in these cases.

Other observations:

1) As we increase the number of threads, contention increases which leads to more wait time for locks for every one and more number of validation failures in optimistic, leading to reduction in throughput.
2) Operation(insert/remove/contains) fail or pass with almost equal probability for every case. Since, the numbers present in list are randomly generated and the arguments to methods are randomly generated as well, this is expected thing to happen.