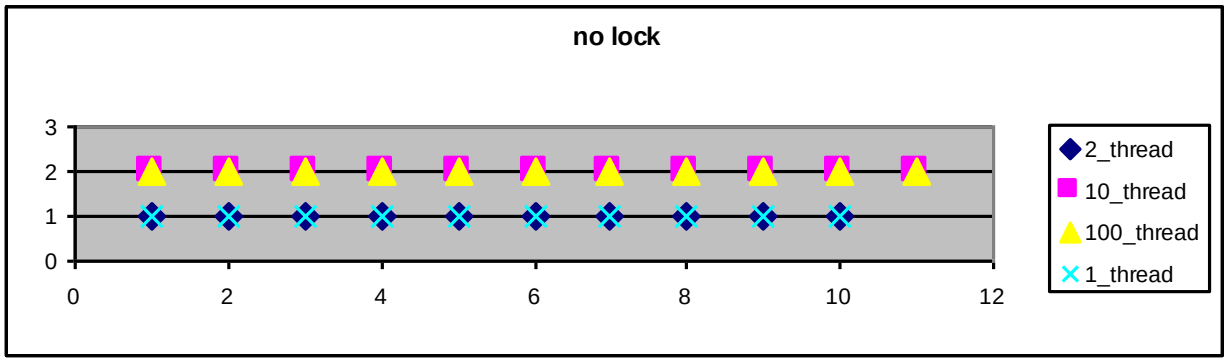
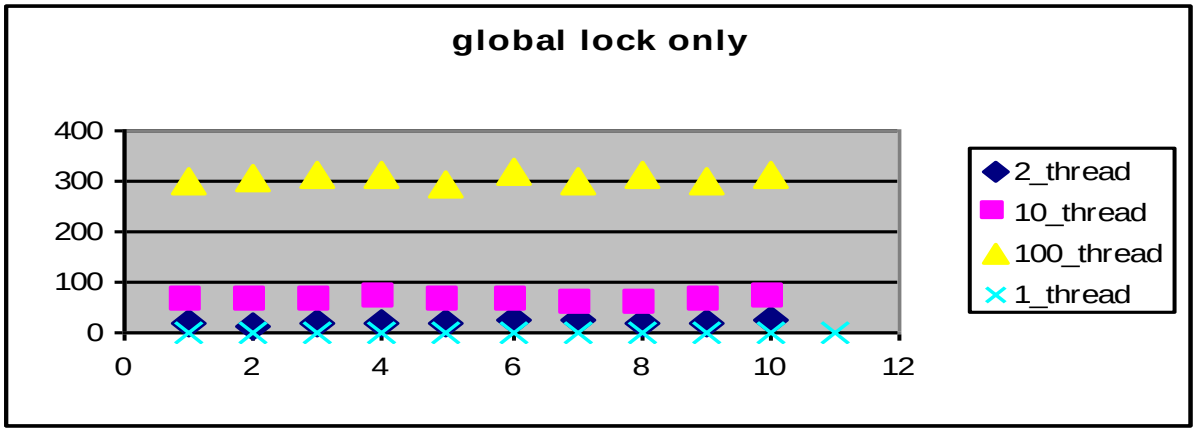
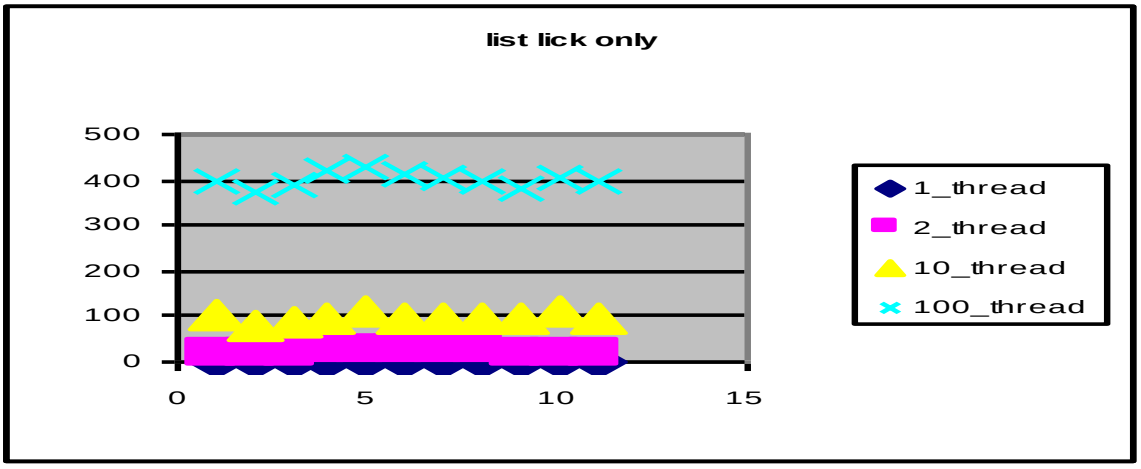
Question1:

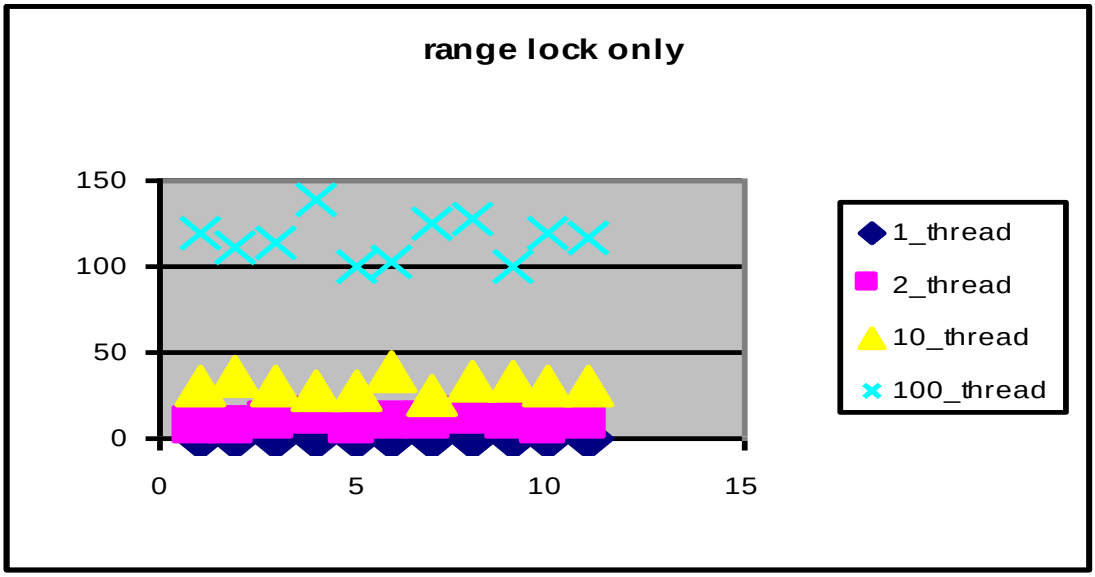
Due to the huge consuming time of creation 1000,000 nodes in some case, we changed the all the thread just create 10000 nodes for convince. And in the following table, if there is no specific unit shows after the numbers, the default unit should be second. And for some case if the running time is larger than 1hour, we just record '>1 hour' for convince.

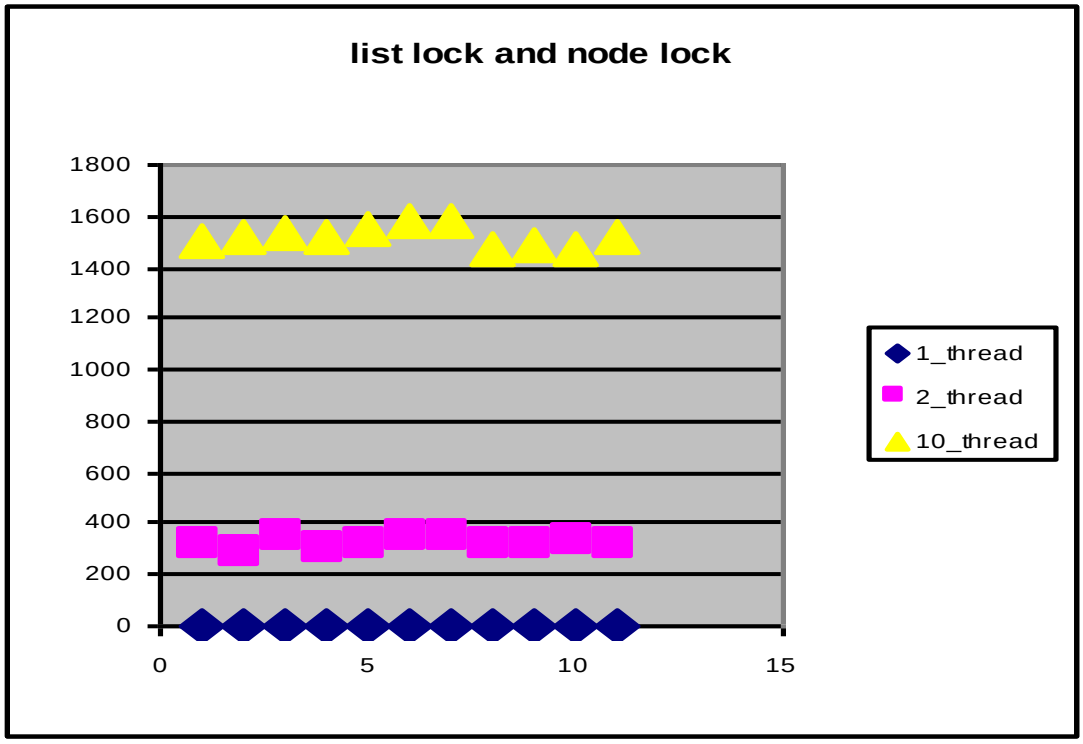
no lock/ #trial	1	2	3	4	5	6	7	8	9	10	avg
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	1	1	1	1	1	1	1	1	1	1	1
10_thread	2	2	2	2	2	2	2	2	2	2	2
100_thread	2	2	2	2	2	2	2	2	2	2	2
Global_lock_only											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	17	15	18	20	19	23	25	18	19	23	19.7
10_thread	60	64	63	66	62	65	57	54	62	70	62.3
100_thread	300	307	310	315	295	321	297	310	303	312	307
List_lock_only											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	15	17	16	20	24	25	22	21	18	19	19.7
10_thread	100	80	89	93	110	95	96	97	99	110	96.9
100_thread	400	370	390	420	425	410	405	395	380	405	400
range_lock_only											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	5	6	9	10	6	7	9	11	8	6	7.7
10_thread	30	35	31	27	29	39	26	33	34	30	31.4
100_thread	120	110	115	140	100	103	125	127	100	119	115.9
Node_lock_only											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	200	210	205	207	196	198	220	184	216	188	202.4
10_thread	1100	1220	1243	1125	1170	1143	1156	1240	1102	1250	1174.9
100_thread	>1 h	>1h	>1h	>1h	>1h	1>h	1>h	1>hour	1>hour	1>hour	1>hour
List_lock & Node_lock											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	310	280	340	297	320	350	343	312	318	330	320
10_thread	1500	1523	1534	1511	1554	1577	1583	1465	1487	1466	1520
100_thread	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h	>1 h
All Lock											
1_thread	1	1	1	1	1	1	1	1	1	1	1
2_thread	355	340	367	340	355	360	332	360	387	342	353.8
10_thread	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour	>0.5 hour
100_thread	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour	>1 hour

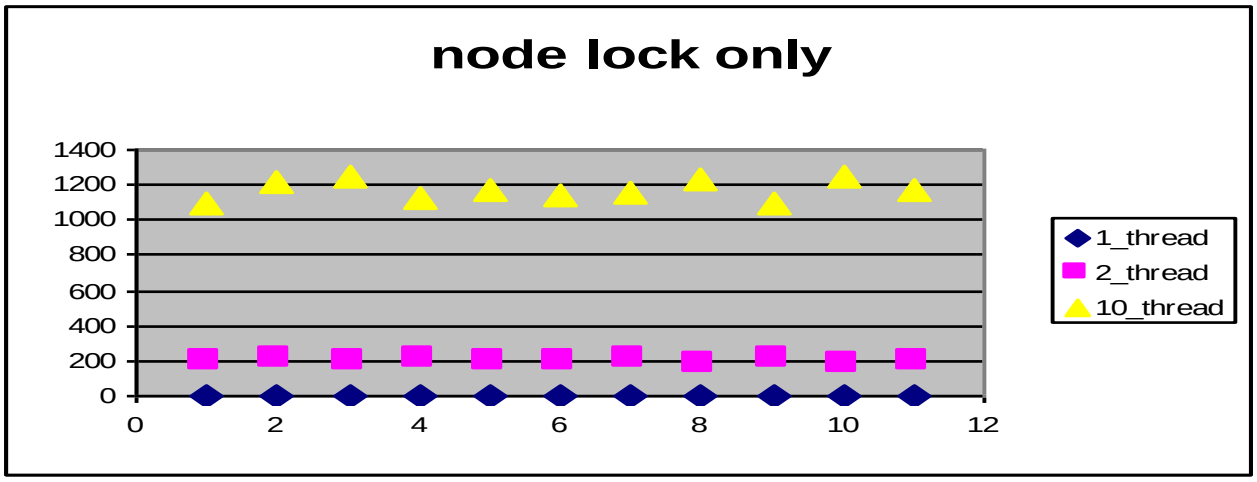

no lock
3
2
1
0
0
2
4
6
8
10
12
2_thread
10_thread
100_thread
1_thread

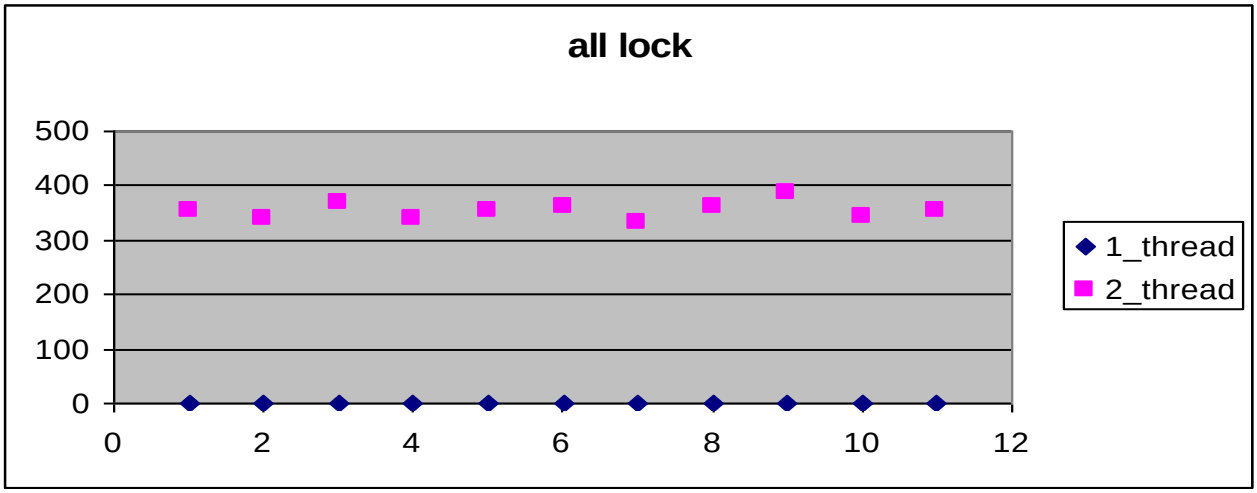

global lock only
400
300
200
100
0
0
2
4
6
8
10
12
2_thread
10_thread
100_thread
1_thread


list lick only
500
400
300
200
100
0
0
5
10
15
1_thread
2_thread
10_thread
100_thread


range lock only
150
100
50
0
0
5
10
15
1_thread
2_thread
10_thread
100_thread


list lock and node lock
1800
1600
1400
1200
1000
800
600
400
200
0
0
5
10
15
1_thread
2_thread
10_thread


node lock only
1400
1200
1000
800
600
400
200
0
0
2
4
6
8
10
12
1_thread
2_thread
10_thread


all lock
500
400
300
200
100
0
0
2
4
6
8
10
12
1_thread
2_thread

Question2: What is the average number of active threads per second in each measurement scenario, for each workload, for each trial?

Table for number of thread per second

no lock/# trial	1	2	3	4	5	6	7	8	9	10
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	2	2	2	2	2	2	2	2	2	2
10_thread	5	5	5	5	5	5	5	5	5	5
100_thread	50	50	50	50	50	50	50	50	50	50
Global_lock_only										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.117647	0.133333	0.111111	0.1	0.105263	0.086957	0.08	0.111111	0.105263	0.086957
10_thread	0.166667	0.1562 5	0.1587 3	0.151515	0.1612 9	0.153846	0.175439	0.185185	0.1612 9	0.142857
100_thread	0.333333	0.325733	0.322581	0.31746	0.338983	0.311526	0.3367	0.322581	0.330033	0.320513
List_lock_only										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.133333	0.117647	0.125	0.1	0.083333	0.08	0.090909	0.095238	0.111111	0.105263
10_thread	0.1	0.125	0.1123 6	0.107527	0.090909	0.105263	0.104167	0.103093	0.1010 1	0.090909
range_lock_only										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.4	0.333333	0.222222	0.2	0.333333	0.285714	0.222222	0.181818	0.25	0.333333
10_thread	0.333333	0.285714	0.322581	0.37037	0.344828	0.25641	0.384615	0.30303	0.294118	0.333333
100_thread	0.833333	0.909091	0.869565	0.714286	1	0.970874	0.8	0.787402	1	0.840336
Node_lock_only										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.01	0.009524	0.009756	0.009662	0.010204	0.010101	0.009091	0.01087	0.009259	0.010638
10_thread	0.009091	0.008197	0.008045	0.008889	0.008547	0.008749	0.008651	0.008065	0.009074	0.008
100_thread	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s
List_lock & Node_lock										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.006452	0.007143	0.005882	0.006734	0.00625	0.005714	0.005831	0.00641	0.006289	0.006061
10_thread	0.006667	0.006566	0.006519	0.006618	0.006435	0.006341	0.006317	0.006826	0.006725	0.006821
100_thread	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027	< 0.027

	/s	/s	/s	/s	/s	/s	/s	/s	/s	/s
All Lock										
1_thread	1	1	1	1	1	1	1	1	1	1
2_thread	0.005634	0.005882	0.00545	0.005882	0.005634	0.005556	0.006024	0.005556	0.005168	0.005848
10_thread	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s	< 0.005 /s
100_thread	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s	< 0.027 /s

Question3:

The more lock we used (or higher locking granularity), the less number of threads per unit time, and the lower “throughput” of the system have. The more lock would introduce more waiting and conflict, therefore the exact unnecessary locks could drop throughput of the system.

Question4:

Yes, there is an effect of workload done on performance. The more workload we have (or the more threads we have), the higher performance depending on the observation. Because multi-threads could make the resting time of CPU as short as possible, therefore the performance would be improved.

Qestion5:

Yes, our program’s performance was improved when we employed pthread_mutex_trylock. The lock function attempts to acquire the mutex. If the mutex is already locked, it suspends the calling thread until the mutex becomes available. The call returns once the calling thread has acquired the mutex. The tryLock function attempts to acquire the mutex. If the mutex is available, the call returns with the mutex locked and returns true. Otherwise, if the mutex is locked by another thread, the call returns false.