Exercise 3

Submitted by:
Sahithya Kodam
Jhalak Patel

The implementations being studied:

- Coarse-grained Implementation
- Hand-over-hand Implementation
- Optimistic Implementation

The execution settings that were constant for all the runs of the three implementations:

- Time duration of the execution: 5 seconds
- Number of threads: 8

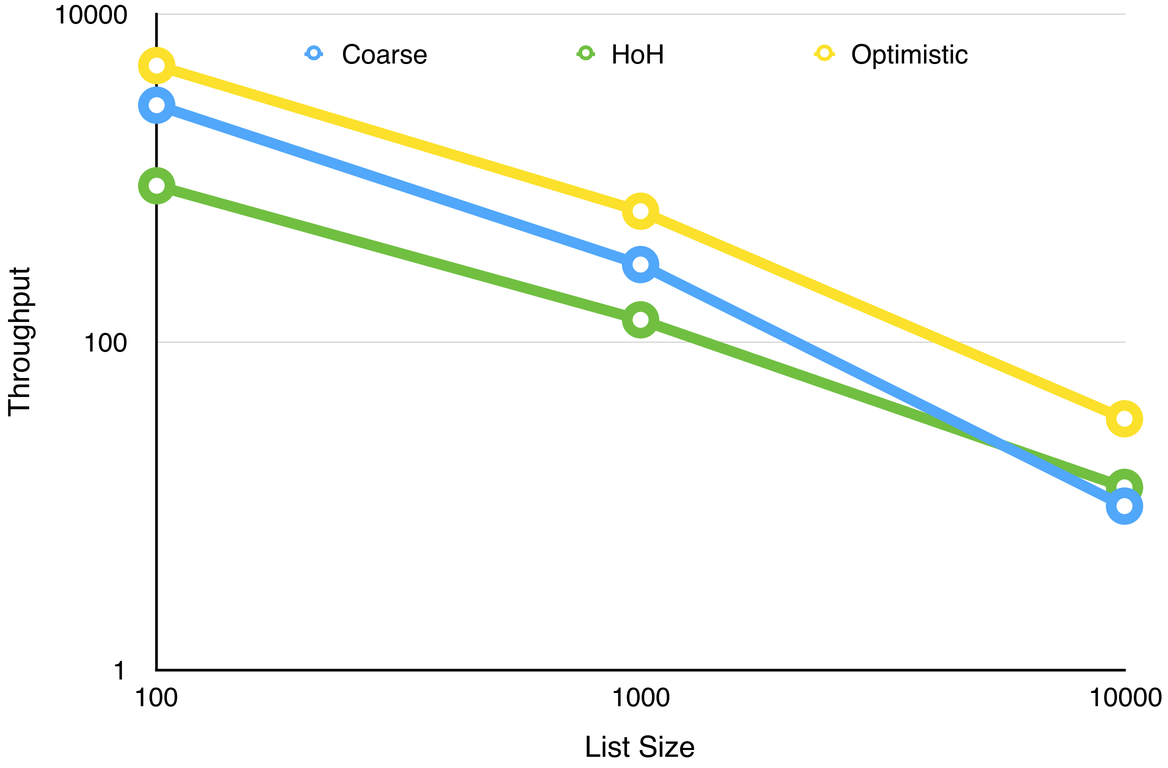
Each of the implementation was executed with

- Initial size: 100, 1000, 1000

For each initial size, runs were conducted with two different update i.e. number of inserts/removes ratios

- Update Ratios: 10, 100

Fig1. Throughput Vs Initial List Size on Logarithmic scale



Execution Statistics

			Success	Failure	Throughput
Coarse-grain Implementation	Initial Size : 100	Ratio : 10	6880880	6879856	2752
		Ratio : 100	4891215	4889083	1956
	Initial Size : 1000	Ratio : 10	740222	738419	295
		Ratio : 100	693622	693093	277
	Initial Size : 10000	Ratio : 10	25368	25144	10
		Ratio : 100	24611	24165	9
Hand-Over-Hand Implementation	Initial Size : 100	Ratio : 10	2231282	2219449	890
		Ratio : 100	1823135	1822896	729
	Initial Size : 1000	Ratio : 10	341180	339405	136
		Ratio : 100	335808	334911	134
	Initial Size : 10000	Ratio : 10	35125	34502	13
		Ratio : 100	30773	30935	12
Optimize Implementation	Initial Size : 100	Ratio : 10	11992838	11986373	4795
		Ratio : 100	10991837	10995280	4397
	Initial Size : 1000	Ratio : 10	1550925	1542856	623
		Ratio : 100	1547002	1550043	619
	Initial Size : 10000	Ratio : 10	85516	85666	34
		Ratio : 100	80733	80300	32

Table. 1 Statistics for Coarse-grained, Hand over Hand and Optimistic implementations

Discussion on General Throughput variation on various parameters:

1. Variation of Throughput with List Size:

It is observed that, for all the implementations, the throughput decreases with the increase in the initial list size. With increase in size of the list the time to traverse the list increases and hence the overall execution time increases. Since execution time is inversely proportional to the throughput, increase in the size of the list significantly decreases the throughput.

2. Variation of Throughput with Update Ratio:

For a fixed list size, increasing the update ratio (number of insert/remove operations) from 10 to 100 decreases the throughput. With update ratio 10, the rest 90% is the contains operation. As opposed to contains operation, the update operations have the added step of adding or removing a node from the list. A contains operation takes lesser execution time when compared to an update operation. This explains the slight decrease in the throughput for the increase in the update ratio.

Comparison of various Implementation for Throughput variation:

On a high level, it can be observed that in terms of throughput, the performance of the different implementations is:

Optimistic > Coarse > Hand over hand.

But as the initial size of the list is increased, hand over hand implementation over takes (results in better performance as compared to) Coarse-grained implementation which is evident from Fig 1.

When the initial size of the list is 10000, the throughput of Hand over hand is greater than coarse-grained.

Optimistic > Hand over hand > Coarse

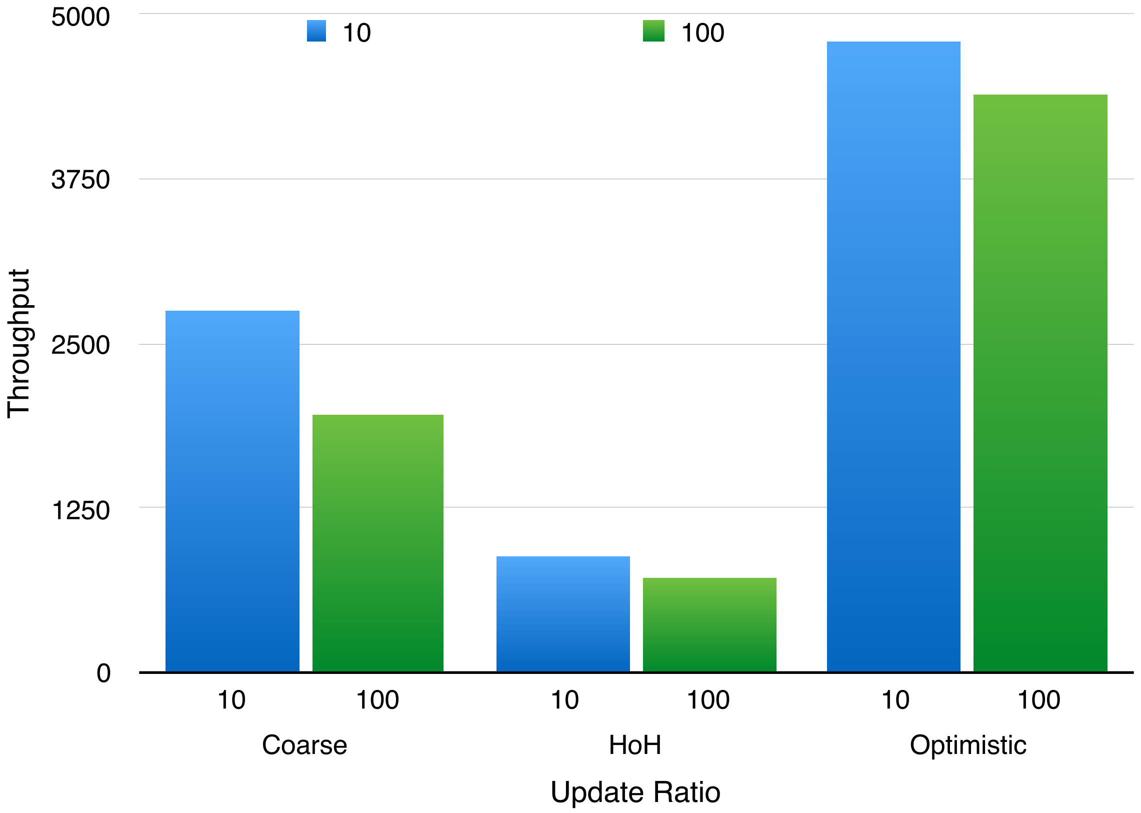
As the optimistic implementation acquires locks only at the position in the list where the operation is to be performed, this implementation achieves greater concurrency, which results in more number of operations and hence greater throughput.
Even though, for an operation, calling the validate() method requires re-traversal of the list, the throughput of optimistic is greater than hand-over-hand and coarse in all the scenarios.
Optimistic is greater than hand-over-hand by a factor of ~4 (initial list size = 100), by a factor of ~3 (initial size = 10000)
Optimistic is greater than coarse-grained by a factor of ~2 (initial size = 100), by a factor of ~3 (initial size = 10000).

Hand-over-hand implementation acquires locks on the current and predecessor nodes as it traverses the list. Acquiring and releasing locks as it traverses the list decreases the concurrency that can be achieved as compared to optimistic implementation. Hence the throughput of HoH is lesser than that of optimistic.

Coarse-grained implementation acquires the lock on the entire list before each operation and releases the lock after completing the implementation. Concurrent threads (concurrent operations) are blocked from accessing the list for a longer time, which results in lesser concurrency and hence lesser throughput.

As the size of the list increases, the performance difference between hand-over-hand and coarse –grained decreases and at initial list size = 10000, the performance of hand-over-hand implementation overtakes coarse-grained. Following the similar trend, we also observed much improved performance (almost 2x) of HoH over coarse grained for initial list size = 20000.

Fig. 2. Throughput Vs. Update Ratio for initial list size 100



Reason for the anomaly i.e. better Coarse grained performance as compared to HoH:
Even though coarse-grained implementation locks the entire list for each operation, when the list size is small, the time to traverse the list is less and hence coarse-grained has a greater throughput at smaller list sizes but as the initial list size increases, the throughput of hand-over-hand increases as compared to that of coarse-grained.

Fig. 2 shows compares the throughput of Coarse-grained, hand-over-hand and optimistic implementation at update ratio 10 and 100, when the initial list size is 100. The throughput when the update ratio is 10 is greater than when the update ratio is 100. At update ratio 10, the majority (90%) of the operations are contains operation. Contains takes lesser execution time, resulting in more number of operations and hence greater throughput.

Fig. 3, Fig. 4 and Fig. 5 show the throughput of the each of the three implementations for update ratios 10 and 100 at initial list size 100, 1000 and 10000. The general observation for each of the implementation is that the throughput increases with increase in size. At a particular size, the throughput for a greater update ration is lesser, though the difference is very less (with increasing list size). As with increase in list size, the operations (remove and insert) are fairly isolated among various threads thus more concurrency is observed and hence the list traversal becomes the bottleneck in implementation which results in similar throughput variation with different update ratio.

Fig 3. Coarse grained implementation with different initial size and update Ratio

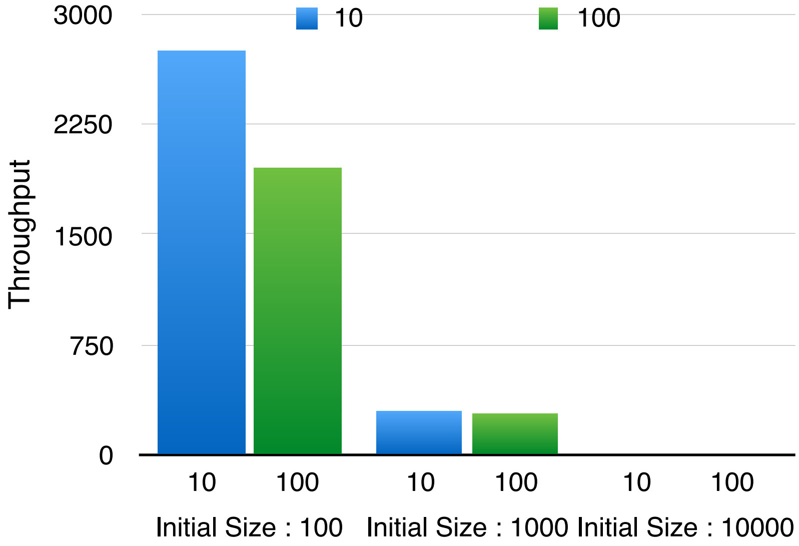


Fig 4. Hand over Hand implementation with different initial size and update Ratio

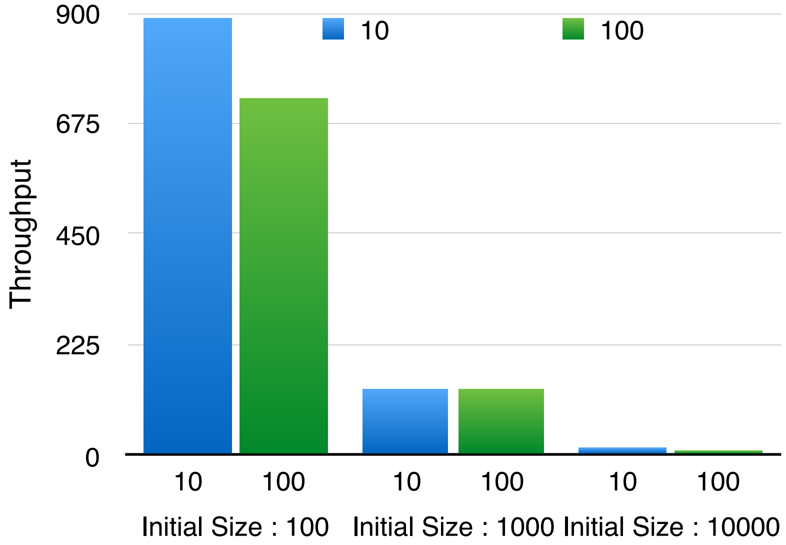
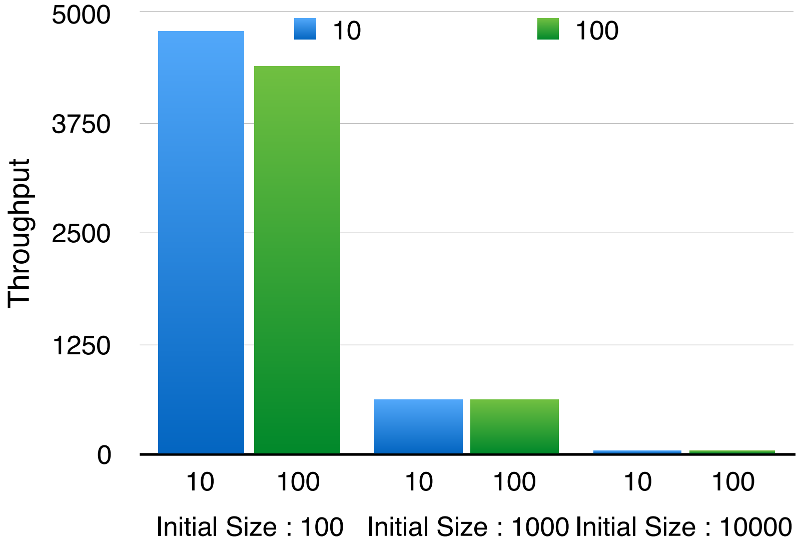


Fig 5. Optimistic implementation with different initial size and update Ratio


Throughput
🟦 10
🟩 100
Initial Size : 100
Initial Size : 1000
Initial Size : 10000