DBMS HOMEWORK - ASSIGNMENT 2

Aarushi Agarwal (2016216)
Surabhi S Nath (2016271)

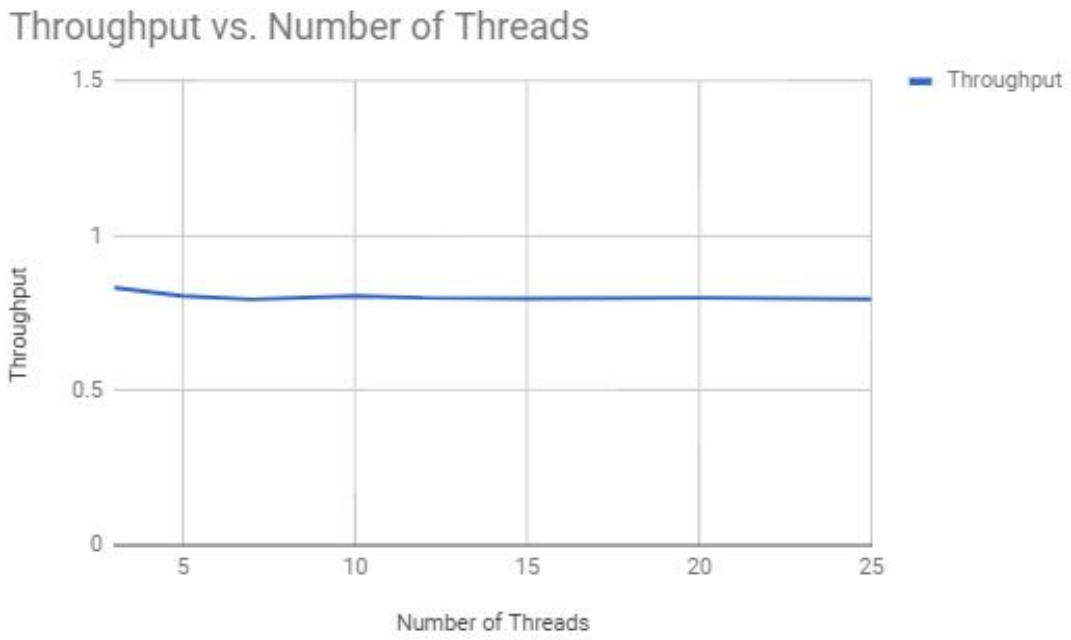
For sequential execution of transactions, a lock was taken on the entire database before every transaction ie, Reserve, Cancel, My_flights, Total Transactions and Transfer. This ensures each transaction is completed before next transaction starts. Hence sequential order was ensured.

For concurrent execution of transactions, locks are taken on the particular flight and particular passenger on which operation is being performed. Shared locks and Exclusive locks were dealt separately to allow multiple transactions to read data items simultaneously but prevent write-write, read-write or write-read. Lock ordering was applied - the flight was always locked before the passenger. Also, the flight with lower ID was locked before the flight with higher ID. This prevented deadlock in 2PL protocol used for concurrent execution.

SEQUENTIAL EXECUTIONS OF TRANSACTIONS

Graph 1 (a): Throughput vs Number of Threads

Number of Threads	Throughput
3	0.8333
5	0.8064
7	0.7954
10	0.8064
12	0.8
15	0.7978
20	0.8
25	0.7961

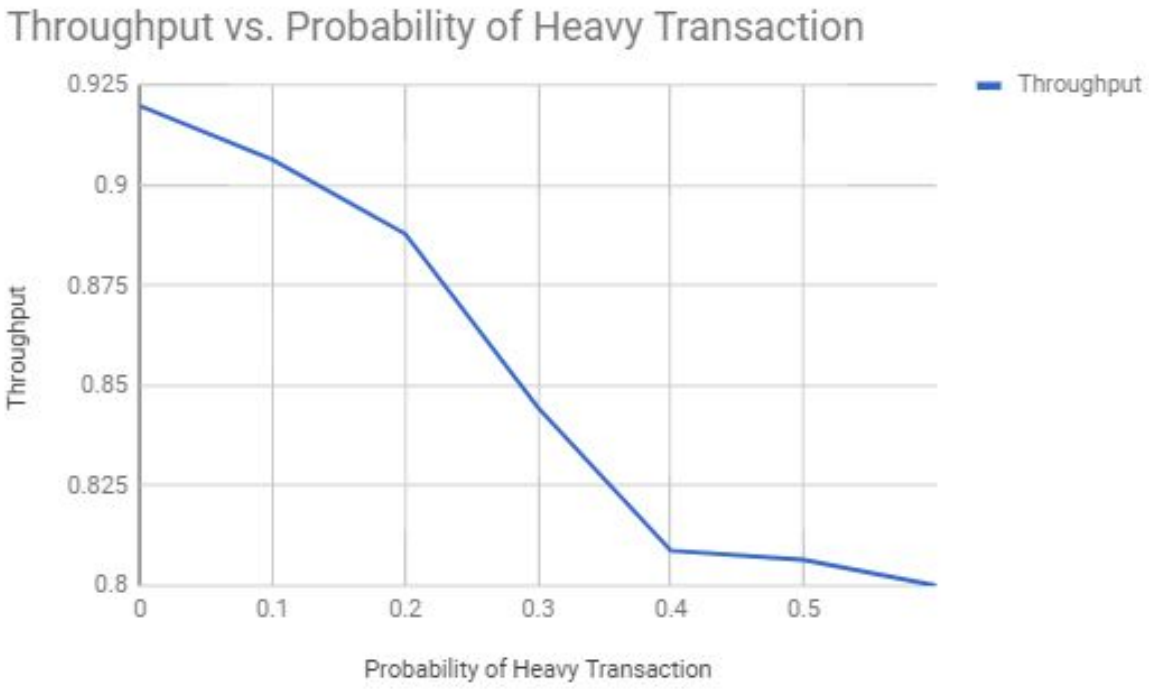


We observe that the nature of this graph is a straight line parallel to the X axis. This indicates that the throughput remains a constant irrespective of the number of threads. This is true since for sequential execution, as the number of threads increase, the number of transactions also increase and the time also increases proportionately. Hence the ratio of number of transactions to the time taken remains a constant.

Graph 1 (b): Throughput vs Transaction Mix

Probability of Heavy Transaction	Throughput
0	0.92
0.1	0.9065
0.2	0.8888
0.3	0.8445
0.4	0.8088

0.5	0.8064
0.6	0.8



We have taken **Transfer** as the heaviest Transaction due to the following reason:

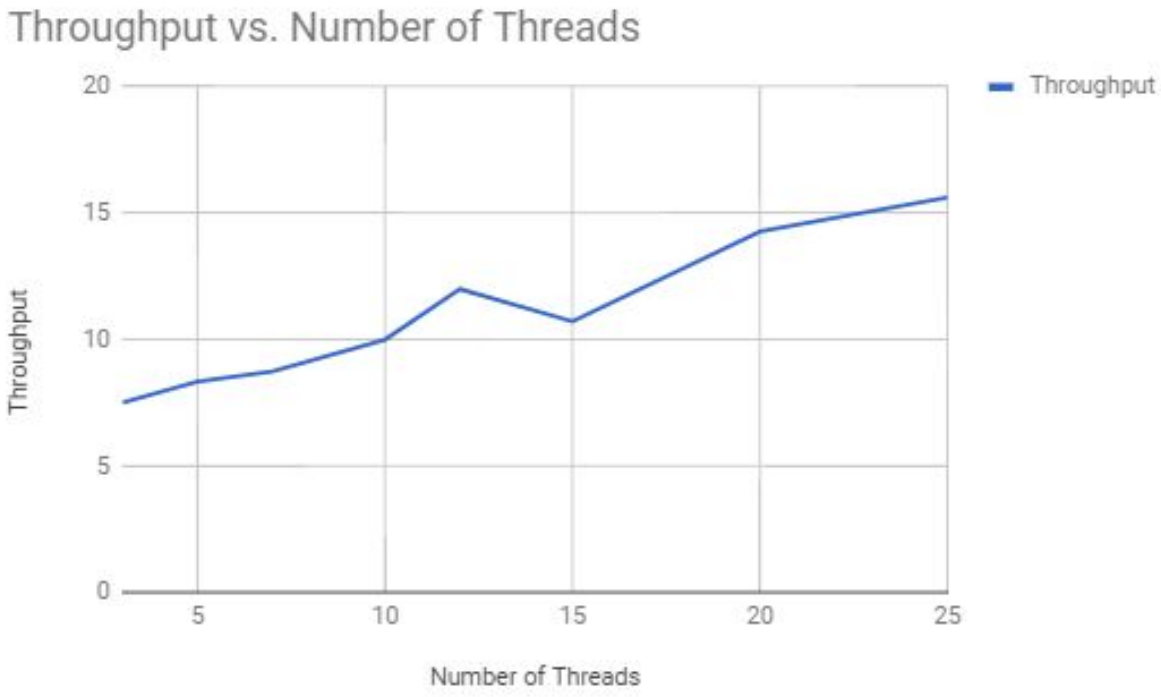
1) Transfer is equivalent to a cancel operation followed by a reserve operation. Hence it is a combination of two transactions which makes it heavier compared to other transactions.

As we can see, in the above graph the throughput decreases on increasing the probability of the heaviest transaction. This is because as we increase the the amount of heaviest transaction, more time is consumed, for executing the same number of transactions and hence reduces the throughput ie, the ratio of number of transactions to the time taken.

CONCURRENT EXECUTION OF TRANSACTIONS

Graph 2 (a): Throughput vs Number of Threads

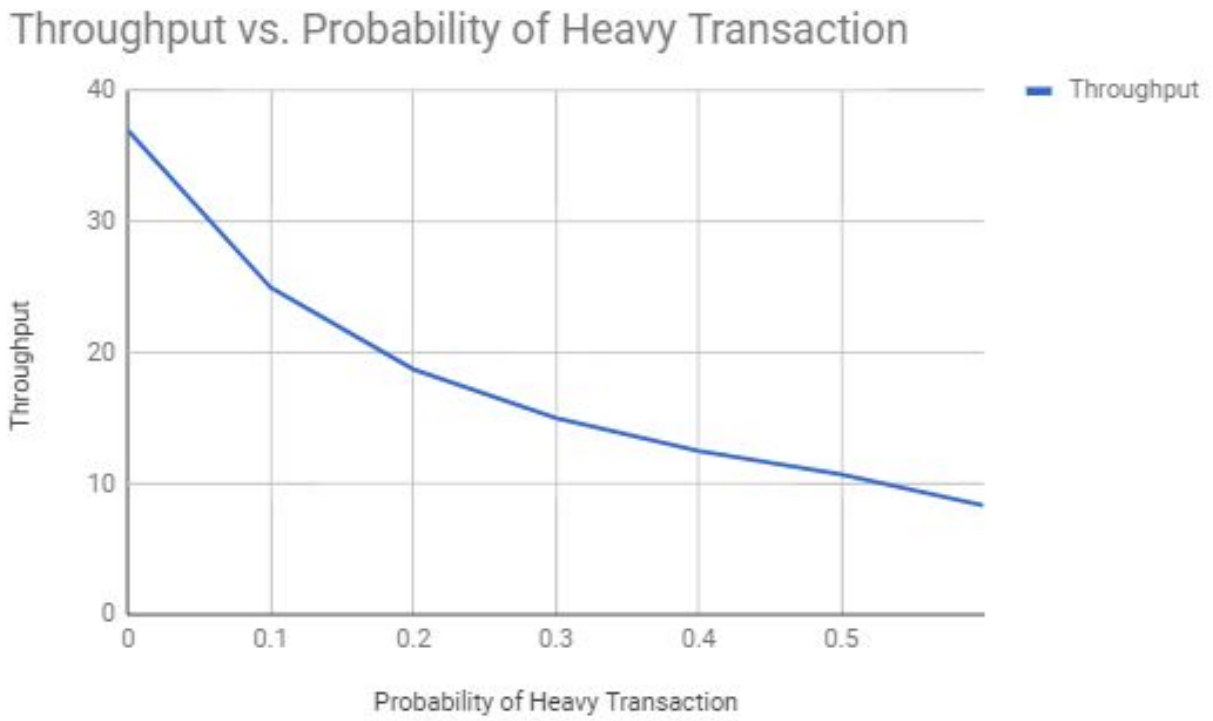
Number of Threads	Throughput
3	7.5
5	8.3333
7	8.75
10	10
12	12
15	10.7142
20	14.2857
25	15.625



As we can see, in the above graph the throughput increases on increasing the number of threads. Though the number of transactions also increase with increase in number of threads, the time does not increase as much since execution is concurrent (slight increase due to waiting time of threads). Thus, more transactions are performed in nearly the same time, hence throughput increases. Also we can note that for the same number of threads, throughput is much higher in the concurrent order case as compared to the sequential order case, due to increased concurrency levels. We also see the slight decrease in values when the number of threads increase beyond a limit, this is because as thread pool size increases, overhead time increases.

Graph 2 (b): Throughput vs Transaction Mix

Probability of Heavy Transaction	Throughput
0	37
0.1	25
0.2	18.75
0.3	15
0.4	12.5
0.5	10.71
0.6	8.33



We have taken **Transfer** as the heaviest transaction due to the following reasons:

1) It uses exclusive locks which makes other threads wait while a thread is performing a transaction which adds to overhead time.
2) Transfer is equivalent to a cancel operation followed by a reserve operation. Hence it is combination of two transactions which makes it heavier compared to other transactions.

As we can see, in the above graph the throughput decreases on increasing the probability of the heaviest transaction. This is because as we increase the the amount of heaviest transaction, more time is consumed, for executing the same number of transactions and hence reduces the throughput ie, the ratio of number of transactions to the time taken. We also note that, for the same probability of heaviest transaction, throughput is higher in case of concurrent execution than that in serial execution.