CS5421 – Serializability Project

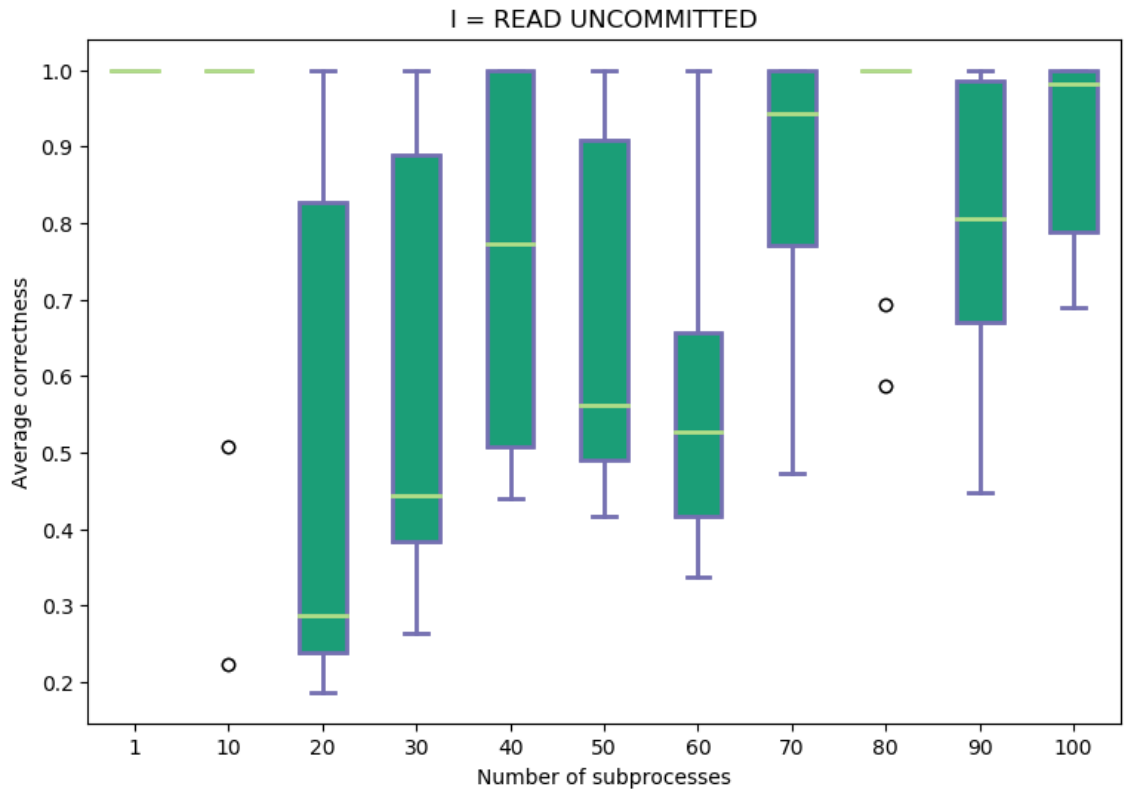
By – Vipul Sharma (A0178385M)

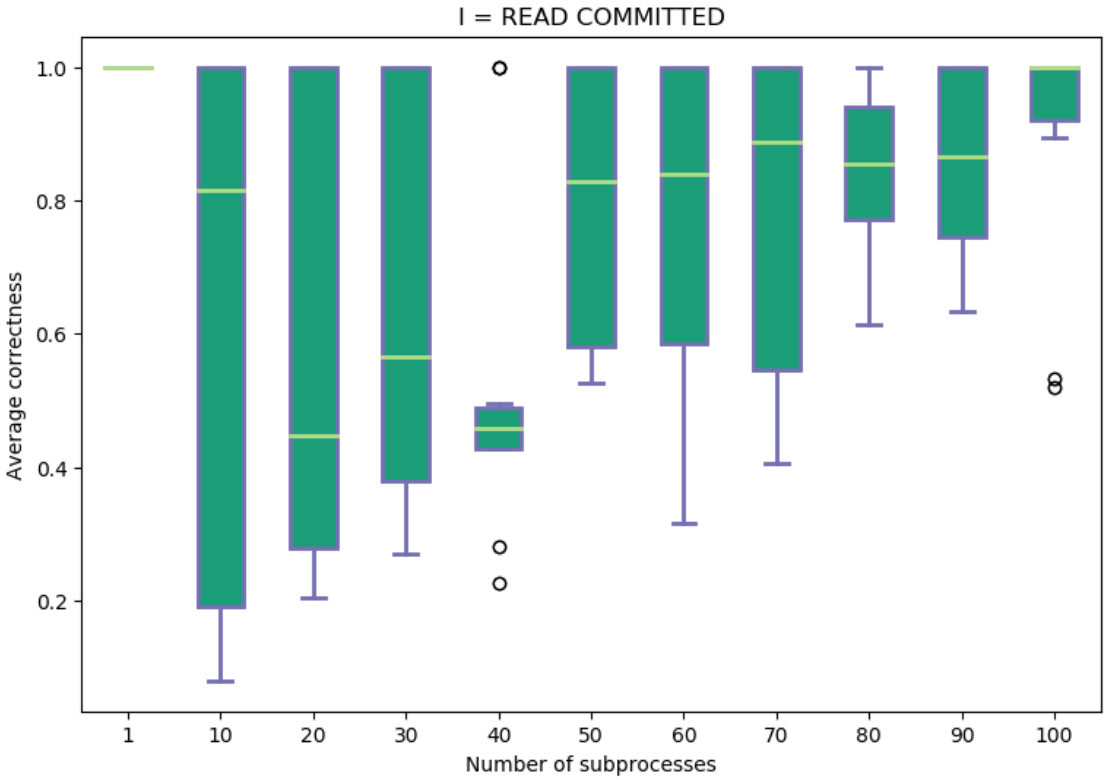
Question 3 –

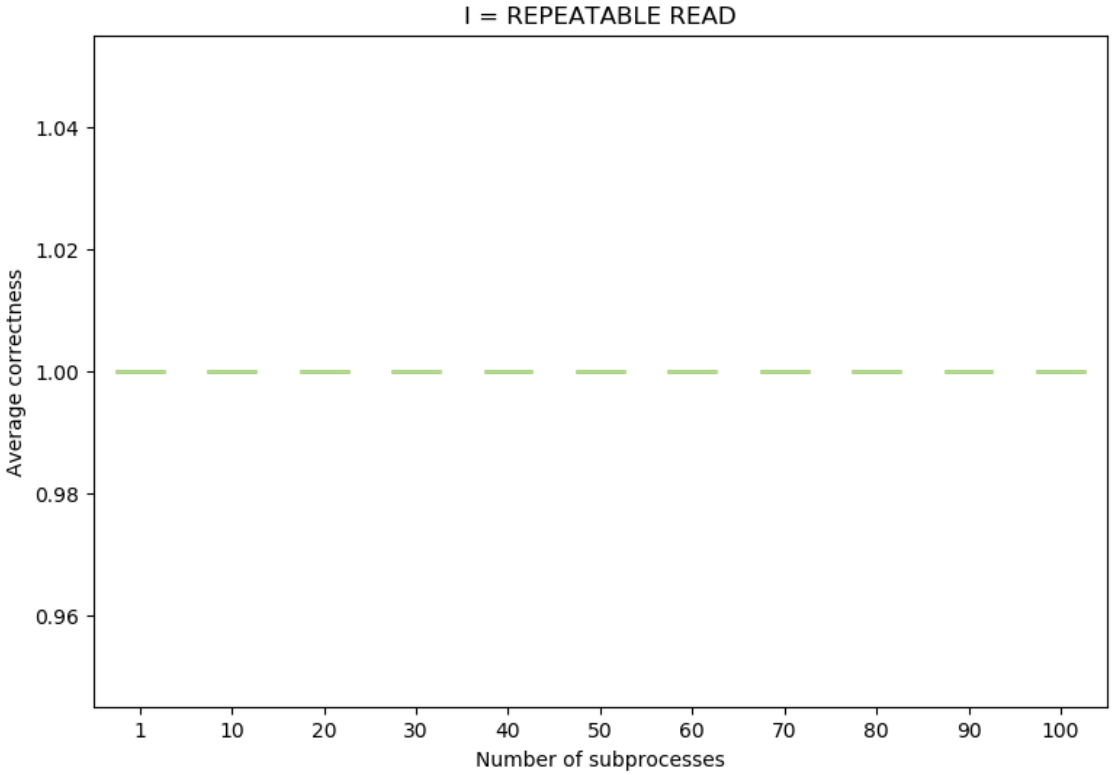
I created a script which takes S, E, P and I as inputs, where S is the number of sums, E is the number of exchanges in a subprocess, P is the number of subprocesses and I is the isolation level ('read uncommitted', 'read committed', 'repeatable read', 'serializable').
We set S = 100 and E * P = 2000. The P value ranges from 1 to 100 in the incremental of 10. For every P value and every isolation level, the script calls run_experiments.py file and record the results of execution time and correctness.

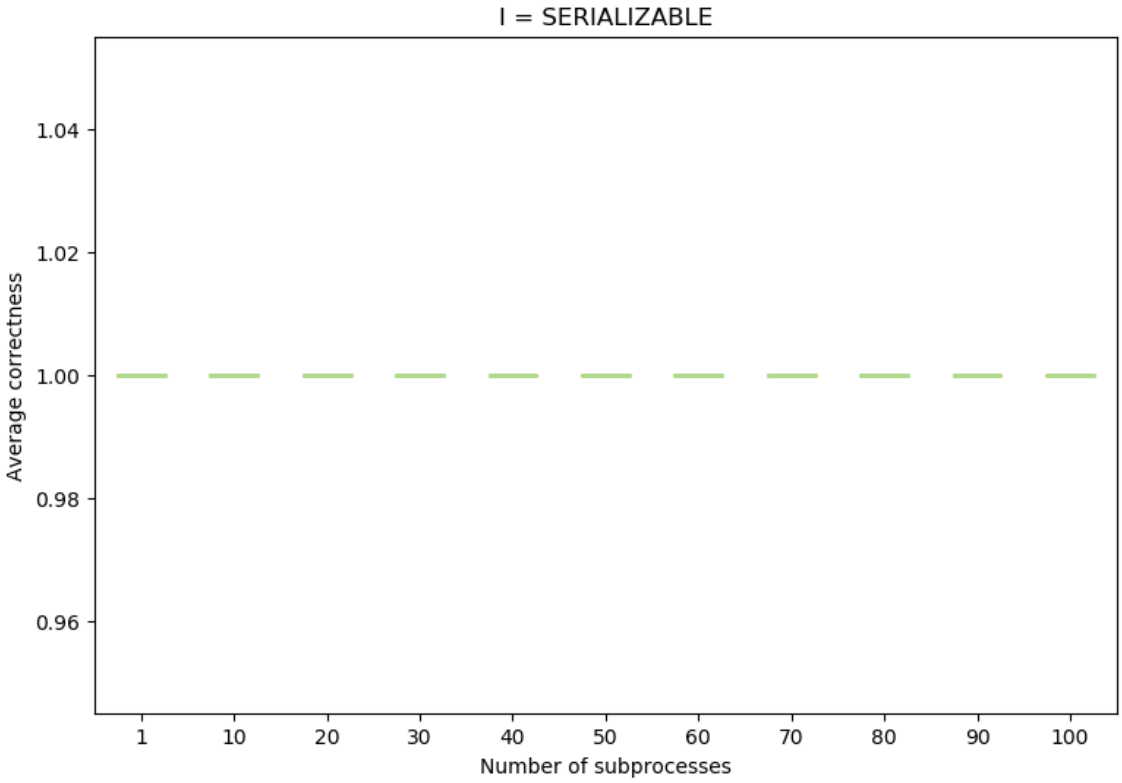
I have used Matplotlib to draw the box plots where X axis contains various P levels and Y axis contains either average execution time or average correctness.

Correctness Box Plots

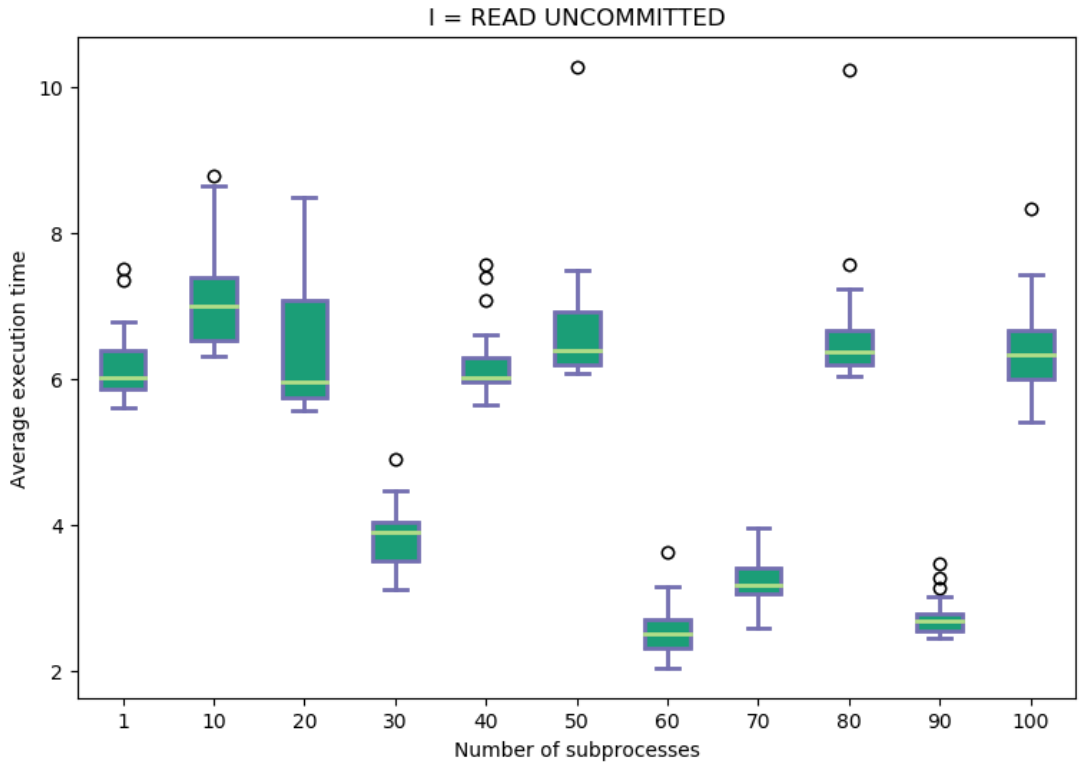


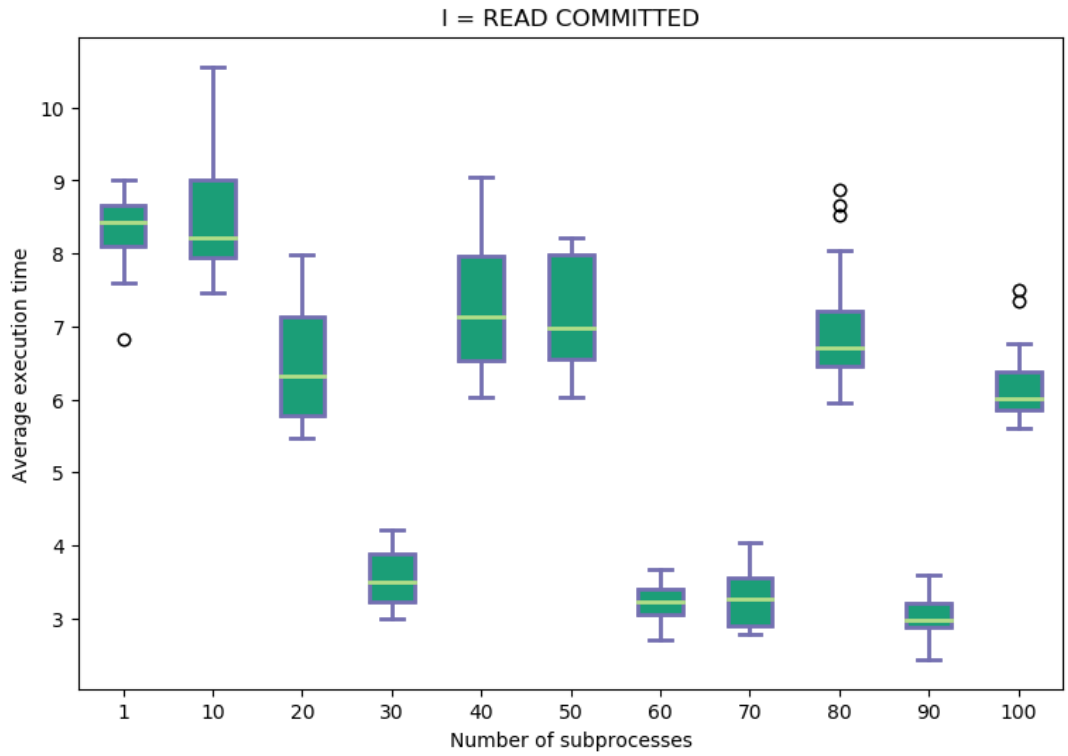

I = READ COMMITTED
Average correctness
1.0
0.8
0.6
0.4
0.2
1
10
20
30
40
50
60
70
80
90
100
Number of subprocesses

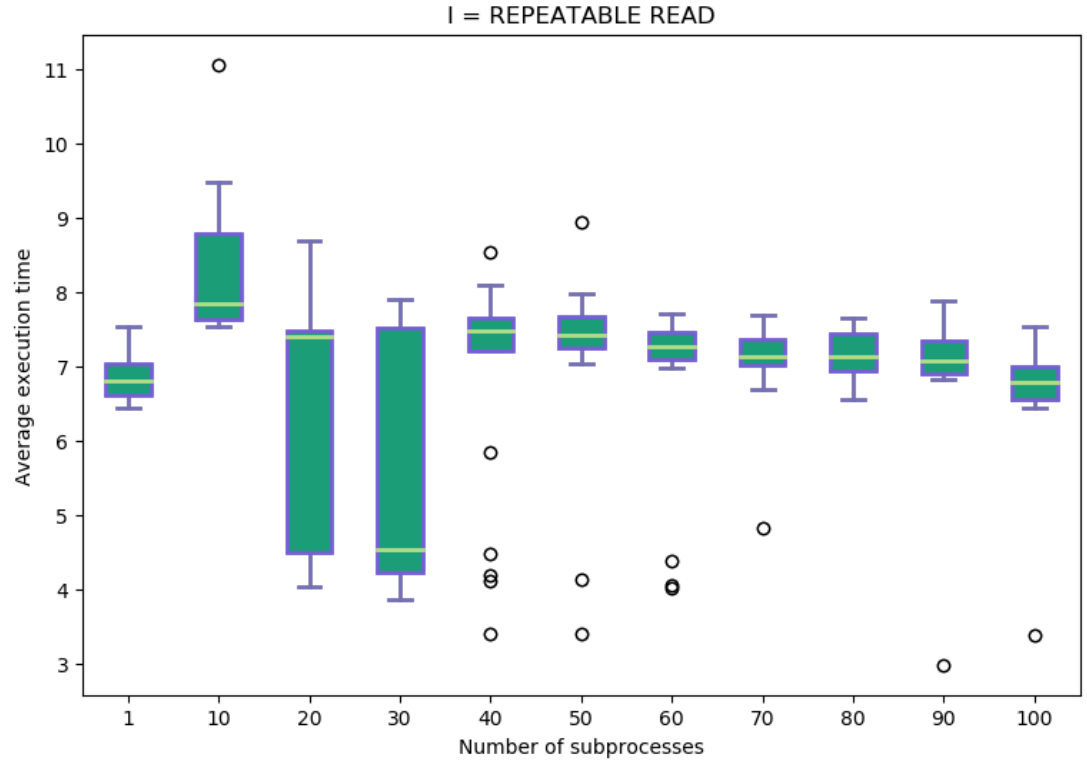

I = REPEATABLE READ
Average correctness
1.04
1.02
1.00
0.98
0.96
1
10
20
30
40
50
60
70
80
90
100
Number of subprocesses

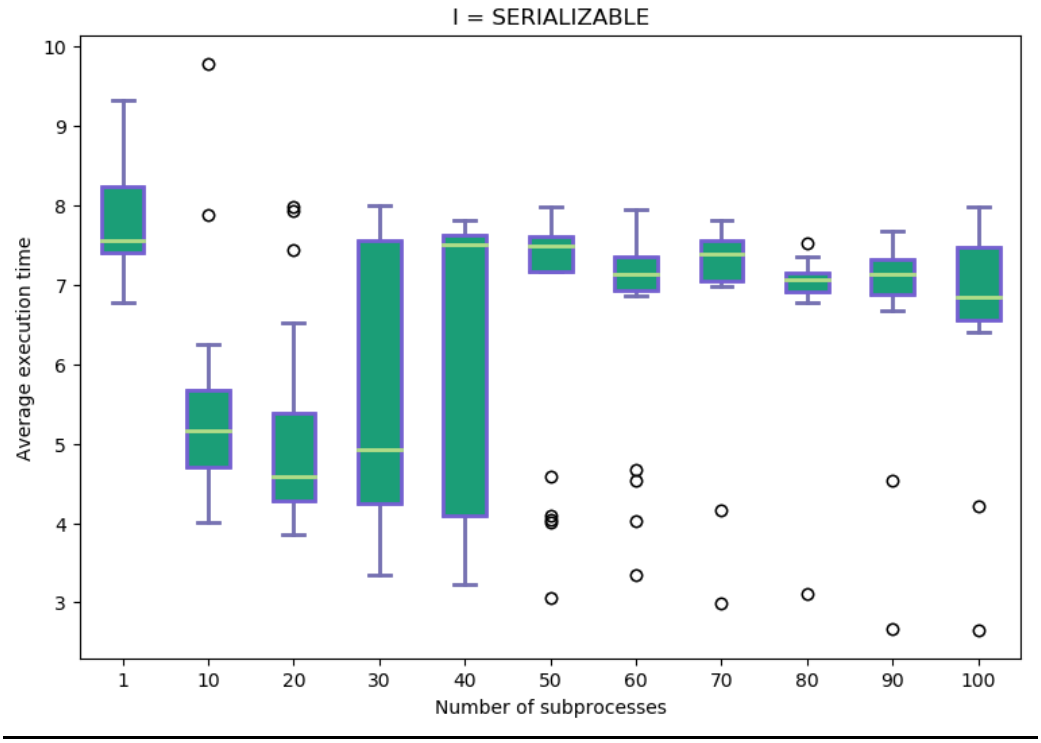


Execution time Box Plots




I = READ COMMITTED
Average execution time
Number of subprocesses
1
10
20
30
40
50
60
70
80
90
100
3
4
5
6
7
8
9
10


I = REPEATABLE READ
Average execution time
Number of subprocesses
1
10
20
30
40
50
60
70
80
90
100
3
4
5
6
7
8
9
10
11



For ‘Serializable’, the correctness measure was always 1. ‘Serializable’ being the highest isolation level, prevents against dirty reads, phantom reads and ensures consistent results.

‘Repeatable Read’ also gave correctness measure as 1. This could be because we are only performing the experiments with select and update queries and not insert queries. So, there is no conflict and data is consistent.

‘Read Uncommitted’ and ‘Read Committed’ gave quite varying values of correctness, ranging from 0.1 to 1. But amongst them, ‘Read Committed’ performed better in terms of correctness as shown in the plots. This could be because ‘Read Committed’ ensures whatever data is read by a transaction has already been committed and persistently stored, hence preventing dirty reads.

It was also observed that I few deadlocks occurred during the experiments for ‘Repeatable Read’ and ‘Serializable’. It could be because of the processes trying to concurrently access the shared resources. Deadlock occurs when all the processes are holding onto some resources and waiting for other resources held by others. In such a case, the system hangs and needs to be restarted.

Another observation related to execution time was that ‘Read Uncommitted’ and ‘Read Committed’ took lesser time as compared to other levels. One reason is that Postgres treats

these 2 levels as the same. Also, they allow concurrent reads and writes and hence, better in terms of execution time.
'Repeatable Read' and 'Serializable' had more or less the same average execution timings.

So, it can be said that there is a trade-off between performance and accuracy when it comes to database transactions. According to the use-case, one needs to define the isolation level which then acquires locks on the data and its usage accordingly.