CS 223 Project 1 Report

A. G. Colaço(acolaco@uci.edu)
Pratyoy Das(pratyoyd@uci.edu)
Yinan Zhou(yinanz17@uci.edu)
Donald Bren School of Computer Science,
University of California Irvine.

May 26, 2023

T

Contents

1 Data Preparation

We pre-process the raw data using Python Pandas dataframe prior to the simulation. To start, we parsed each data file to extract the timestamp information within each SQL command. Each extracted timestamp is converted into an integer time value between 0 and 1,200,000 which represent a particular millisecond within the range of 20 minutes. A time value of 200 means the corresponding command "arrives" at the simulator 200 milliseconds after the experiment run starts. The commands in all three data files, one for queries and two for inserts, are mixed together and sorted in ascending order according to its time value. All the mixed commands are then written back to disk in a single file in the sorted order. Each command is prefixed by its corresponding time value. Two large mixed command files are generated at the end of data preparation stage, one for the *low_concurrency* data and one for the *high_concurrency* data.

2 Simulator Design

We have three components to the simulator design all developed in Java, namely the TransactionSender(TS), the OperationProducer and the Driver Program. The Driver is responsible for running each experiment setup described later in our report. It creates the TransactionSender objects as threads and performs the cleanup after the threads are executing. It also starts the Producer before any of the experiment setups begin. The TransactionSender does the following:

1. It creates a new connection to PostgresDB.
2. It polls the Producer for a new operation. The Producer is basically a single thread that loads the entire workload into memory and stores a synchronized variable cursor that returns the next operation and the time at which the operation has to be executed.
3. Once the TS thread receives the time and the operation, it polls a StopWatch, that is started by the Driver Program once all the threads are allocated, for global system time. It sleeps for the duration of (operation execution time - global system time) and then adds the operation to the transaction.
4. It repeats (2) until the transaction size is reached and then attempts to commit the transaction. If the commit is successful it logs the response time for queries and inserts in that transaction as well as the commit time.
5. Once the entire experiment is complete for that thread, it writes its logs to disk and exits.

The Driver exits once all the experiments are completed

3 Experiment Setup

We run our experiments on a Ubuntu 22.04 machine with 32 Dual-Core CPUs and 64GB RAM. The machine has all the pre-processed SQL commands and the simulator code locally. When we tried the experiment for the first time, we were not able to finish processing the *low_concurrency* commands within 20 minutes, therefore we modified the following PostgreSQL's settings to improve its processing power:

- The *wal_level* is set to minimal which disabled the Write Ahead Logging functionality completely
- The *max_wal_senders* is set to 0 so that Postgres does not reserve any thread for WAL purposes.
- The fsync and *archive_mode* are both set to off to prevent any logs or checkpoints to be written to disk.
- The *max_connections* is set to 1000 to accommodate for the highest multiprogramming level we experiment with.
- The $shared\_buffers$ is set to be 128MB.

The control variables we used in the experiments are:

3.1 Multiprogramming level

We vary the multiprogramming level by changing the number of TransactionSender threads.We experiment with the following thread counts: 50, 100, 150, 200 and 300. We also use this variable as the x-axis for the graphs we generate.

3.2 Transaction Size

Transaction size is set to be one of the following values: 1, 2, 5, 10, which determines the amount of SQL operations that will be given to the next idle TransactionSender upon request. Each transaction can have an arbitrary number of Read and Write operations in an arbitrary order, but the total number is fixed except when the remaining operations are less than the transaction size.

3.3 PostgreSQL's Isolation mode

Whenever a TransactionSender establishes a connection with Postgres, it specifies the connection isolation level to be one of the following three levels: *read_committed*, *repeatable_read*, and serializable. For each experiment run, all TransactionSender threads use the same level of isolation throughout the full run.

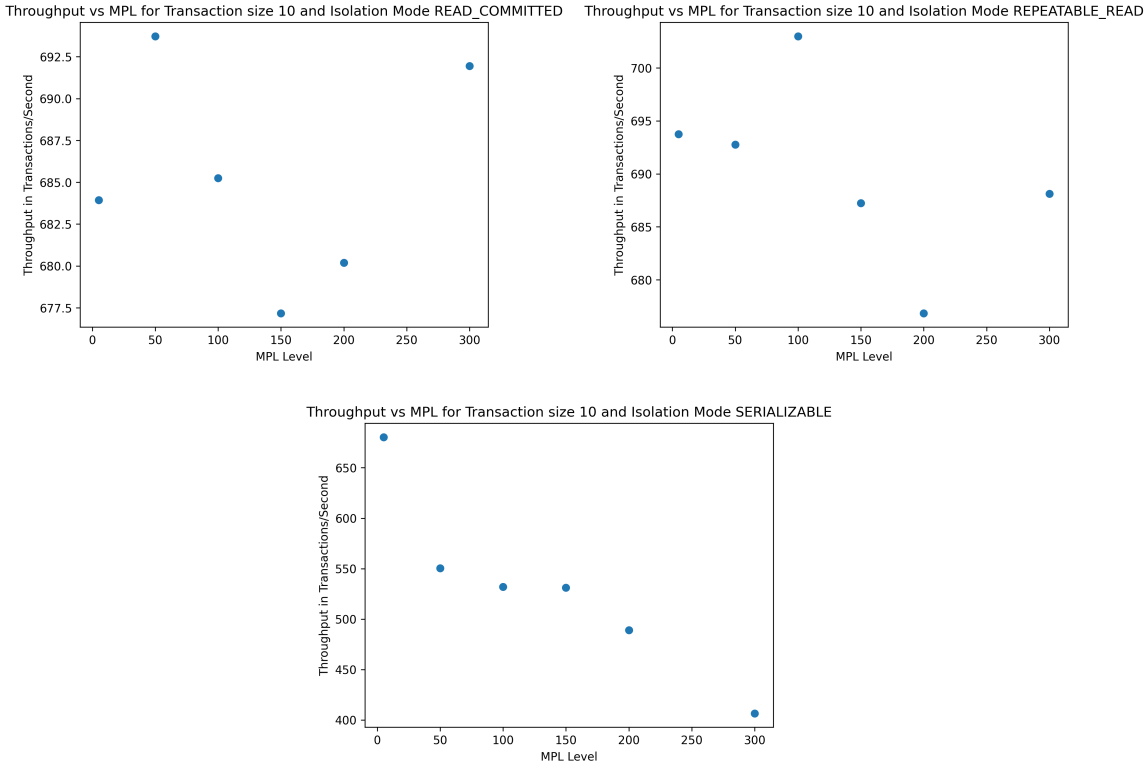

Figure 4.1: Clockwise from top left:Throughput vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

4 Experiment Results and Analysis

Based on the control variables above, we generate the experiment results in the following way. For each unique combination of concurrency level, MPL, transaction size and isolation mode, we process all the SQL commands and record the response time for each individual query. At the end of the experiment run, we take an average over all the response times for insert commands to obtain an average insert response time. Similarly, we calculate the average query response time. Each experiment run yields these two values as the final results. These data points are used to generate the graphs in the next subsection to demonstrate the effect of MPL on the insert, or query, commands at different settings.

4.1 Low Concurrency

4.1.1 Throughput vs MPL (txn sizes 2, 5 and 10)

From 4.1, we get the throughputs for Repeatable Read and Read Committed are quite stable. On the other hand, the throughputs for serializable transactions degrade considerably with MPL Level, which is understandable as they are the highest level of transaction isolation -emulating serial execution of transactions.

From 4.2, All three graphs exhibit a downward trend, ignoring spikes. The perfor-

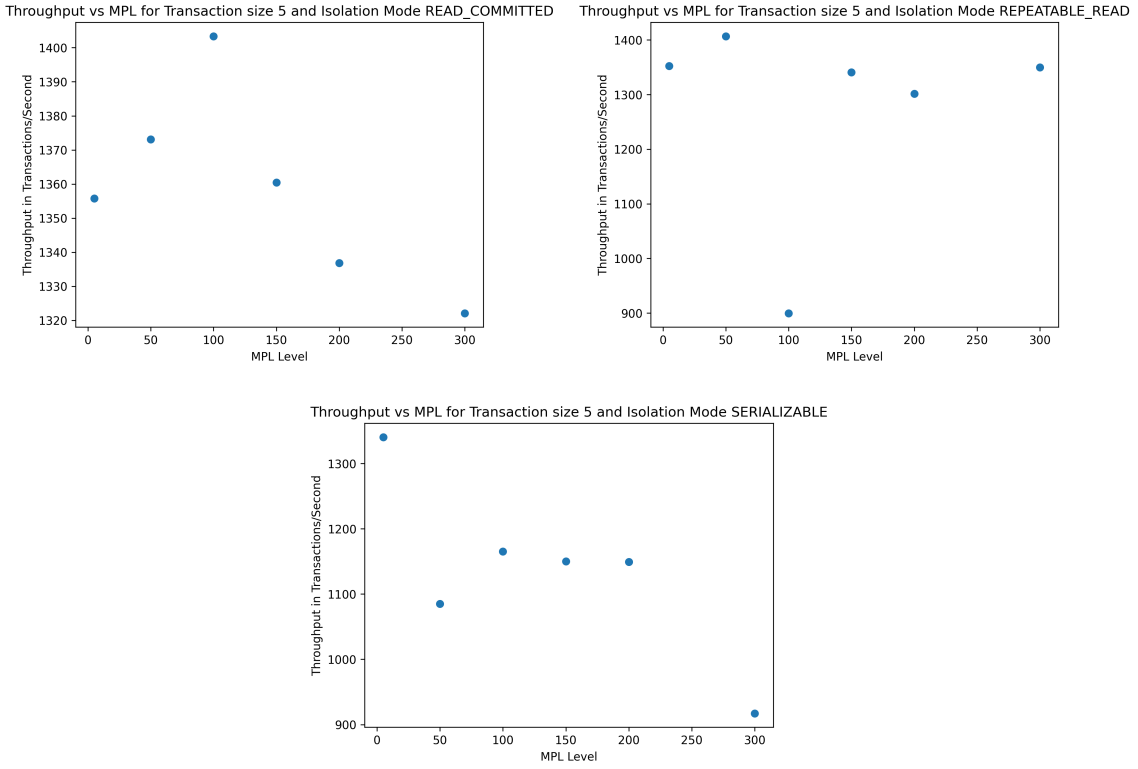

Figure 4.2: Clockwise from top left: Throughput vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

mance of the system is also much better than that with txn size 10 but worse than that with txn size of 2. In that way, the graphs are consistent with the trend observed of the insert times with the same insert times.

From 4.3, all three graphs exhibit a downward trend and the max and min transactions are between 400 txns/second of each other. The performance of the system is much better than that with txn size 10.

4.1.2 Query Response Times vs MPL (txn sizes 2, 5 and 10)

From 4.4, we notice an overall positive correlation between MPL Level and the Average response times, with no major fluctuations. Also it seems that there is a considerable fluctuations from 250 MPL level to 300. The serializable mode transactions perform worse than the other modes. The response times are less than the insert times on average.

Ignoring the spikes, from 4.5 it can be inferred that transaction size of 5 performs slightly better than transaction size of 10 across all MPL levels. Again, performance of select queries are better than insert queries.

Taking into account 4.6, we see spikes at 5/50 MPL levels for all 3 modes which might be attributed to system fluctuations. Other than that repeatable read response times are mostly stable, while the other 2 modes exhibit a positive correlation of MPL and response times. Of all the 3 transaction sizes, it seems that transaction size 5 gives the best performance.

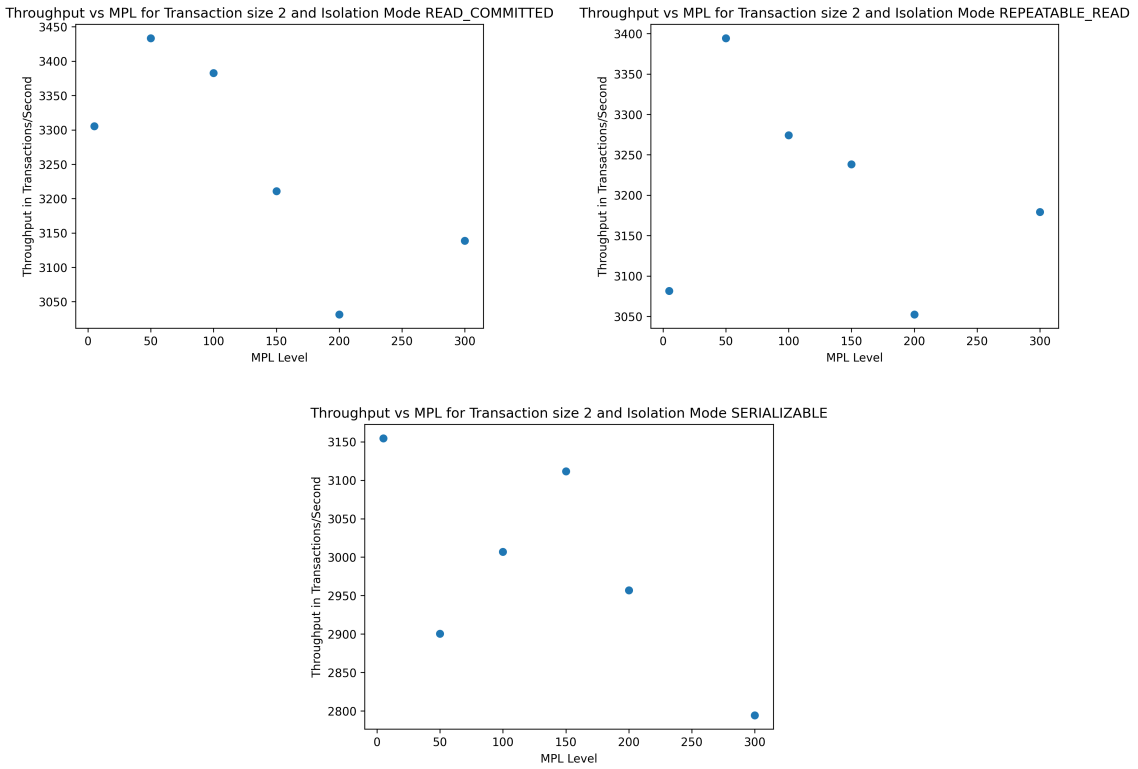

Figure 4.3: Clockwise from top left:Throughput vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

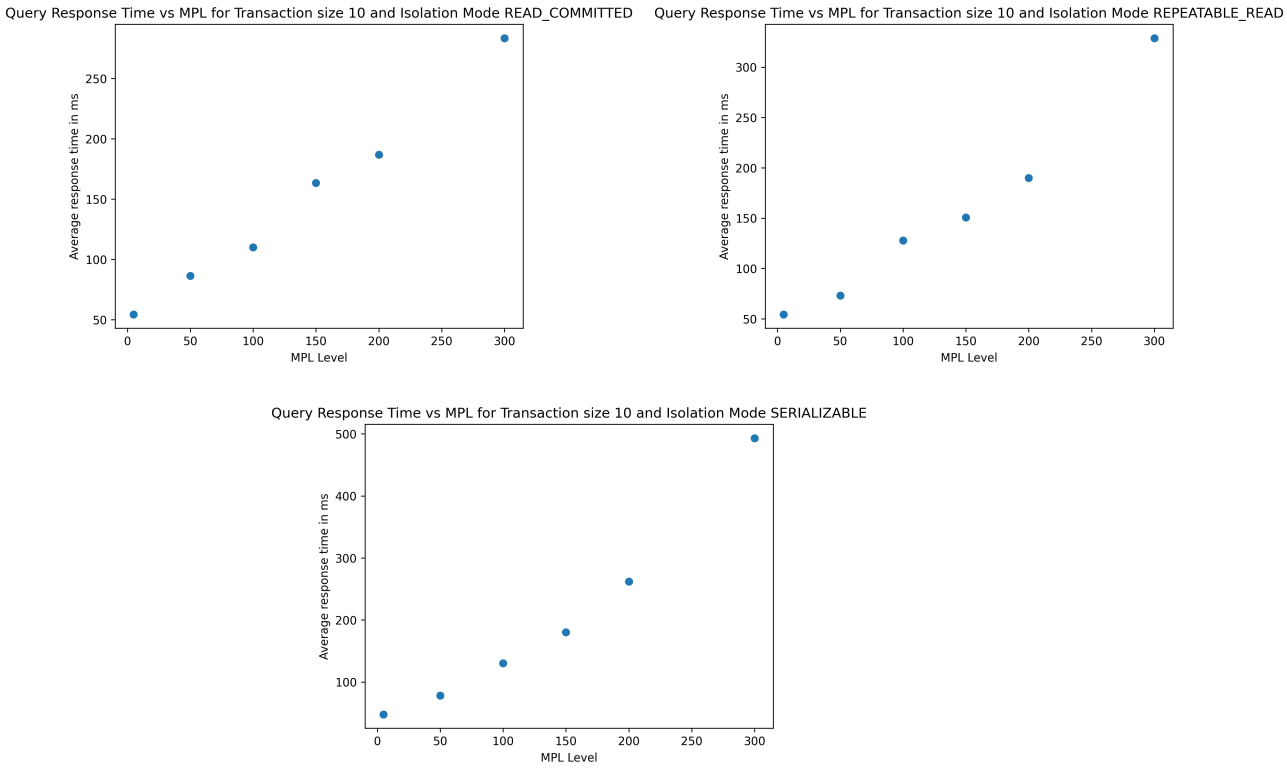

Figure 4.4: Clockwise from top left: Query Response Time vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

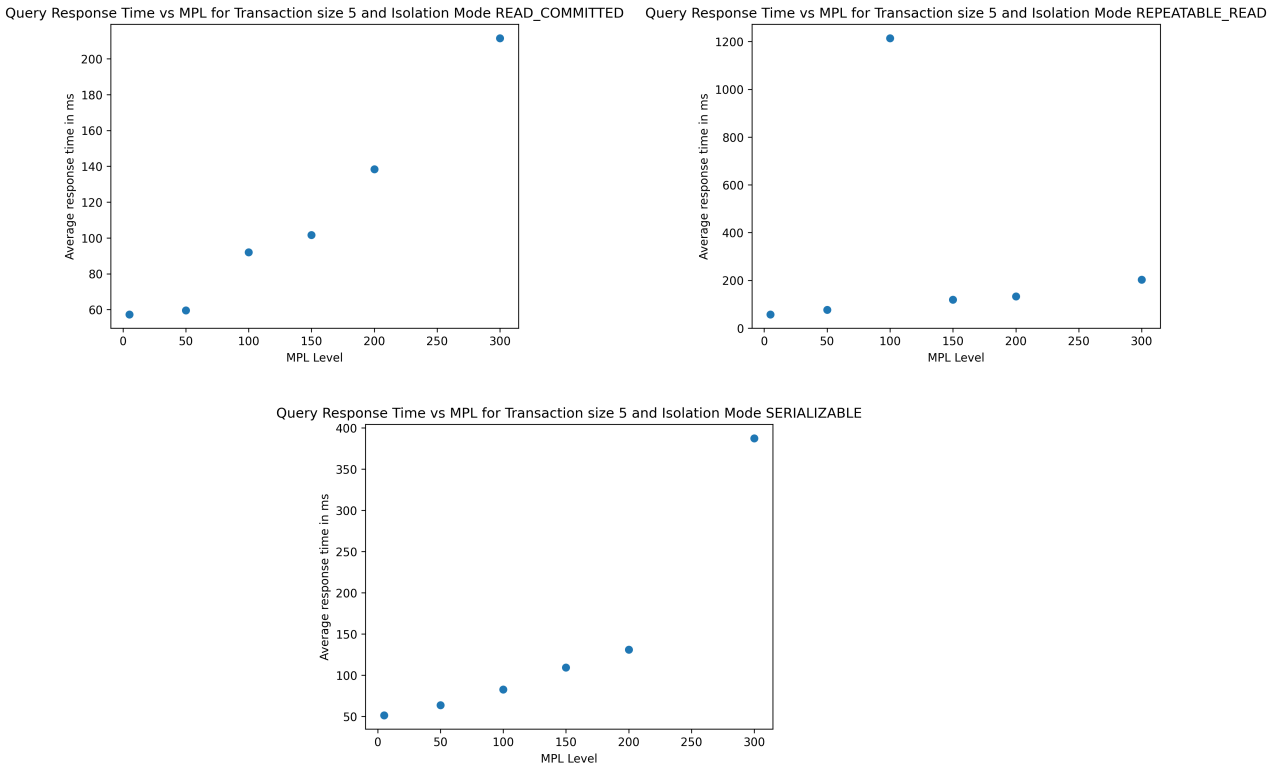

Figure 4.5: Clockwise from top left: Query Response Time vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

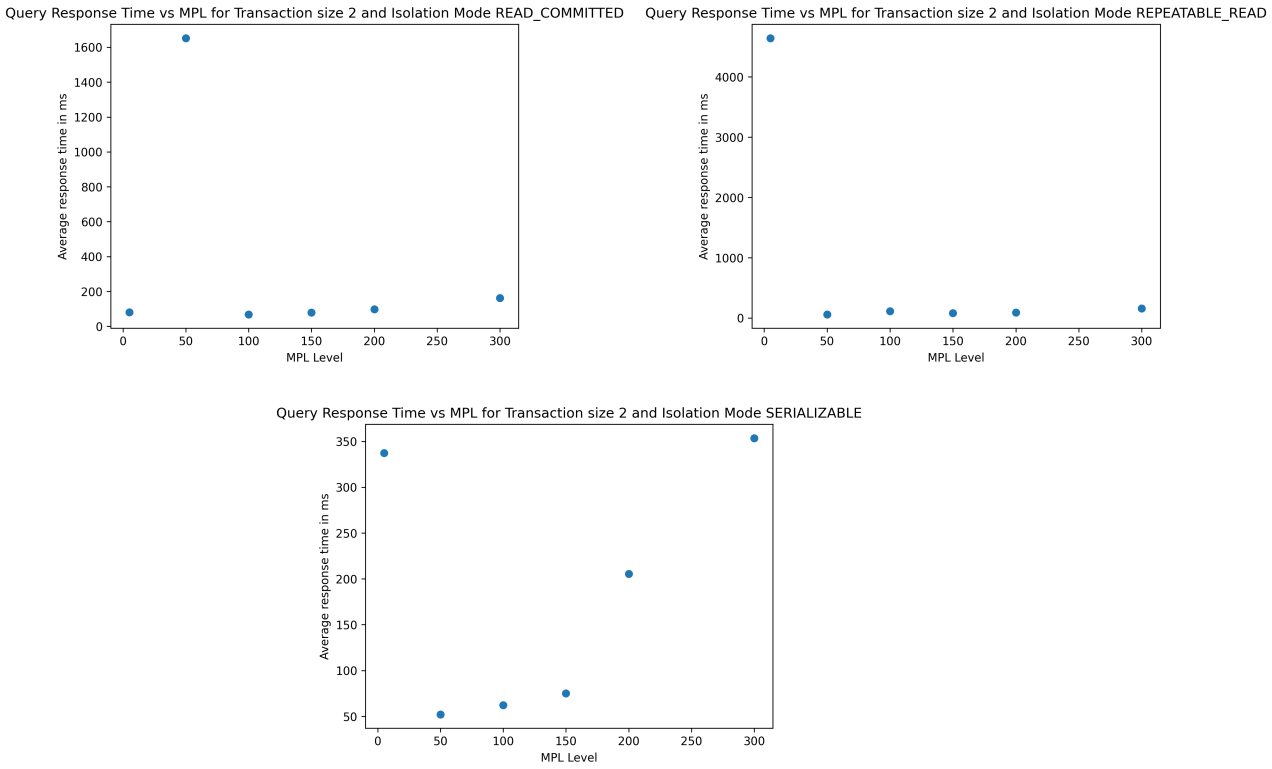

Figure 4.6: Clockwise from top left: Query Response Time vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

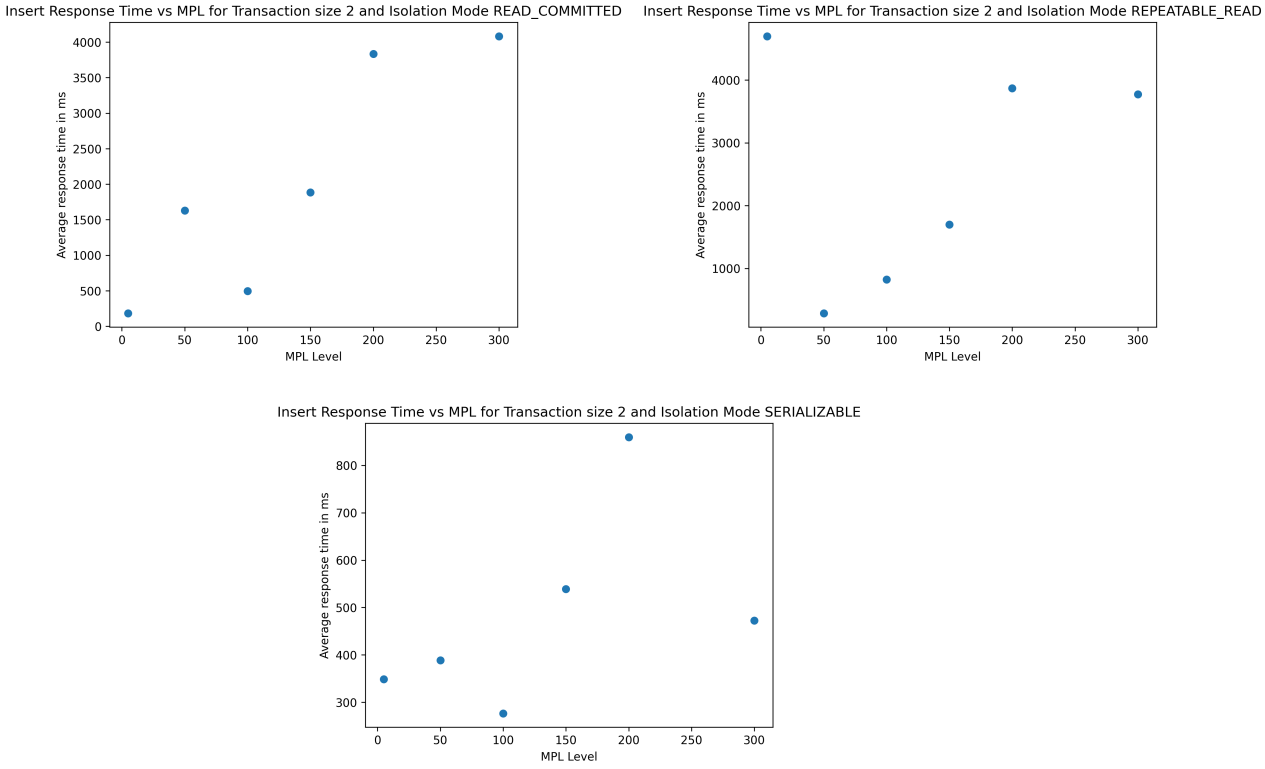

Figure 4.7: Clockwise from top left: Insert Response Time vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

4.1.3 Insert Response Times vs MPL (txn sizes 2, 5 and 10)

From 4.7, we notice the best performance for read repeatable transactions is at 50 MPL level while for serializable transactions, it is at 100 MPL. Also the response times for repeatable reads and read committed txns are comparable, while serializable transactions take less time.

From 4.8 we get the average response times are more than those for the corresponding experiments for transaction size 10 while they are less than those for transaction size 2 for read committed and repeatable read modes. Here too, serializable transactions take the least time.

From **??** we see the response times for repeatable reads and read committed txns are comparable, while serializable transactions take considerably less time. All isolation modes perform poorly as MPL increases. It seems that transaction size 2 is most optimal for the insert response times.

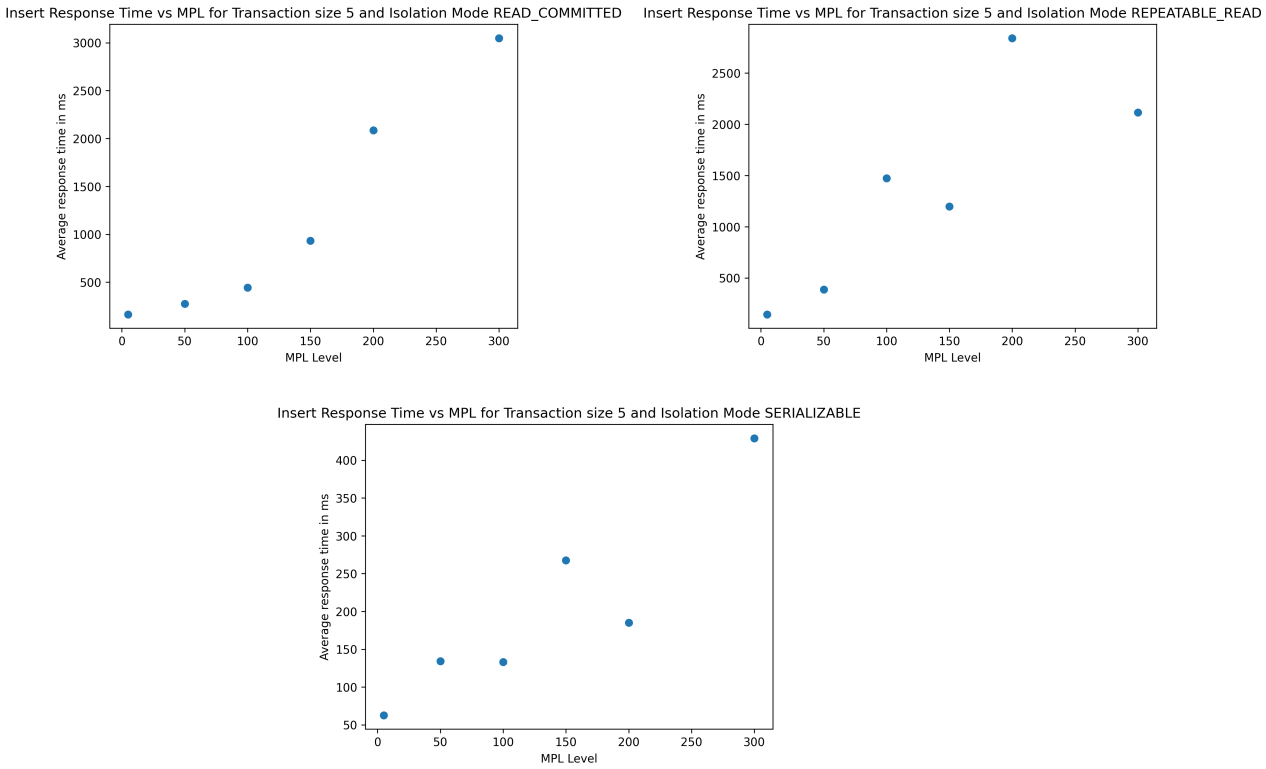

Figure 4.8: Clockwise from top left: Insert Response Time vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

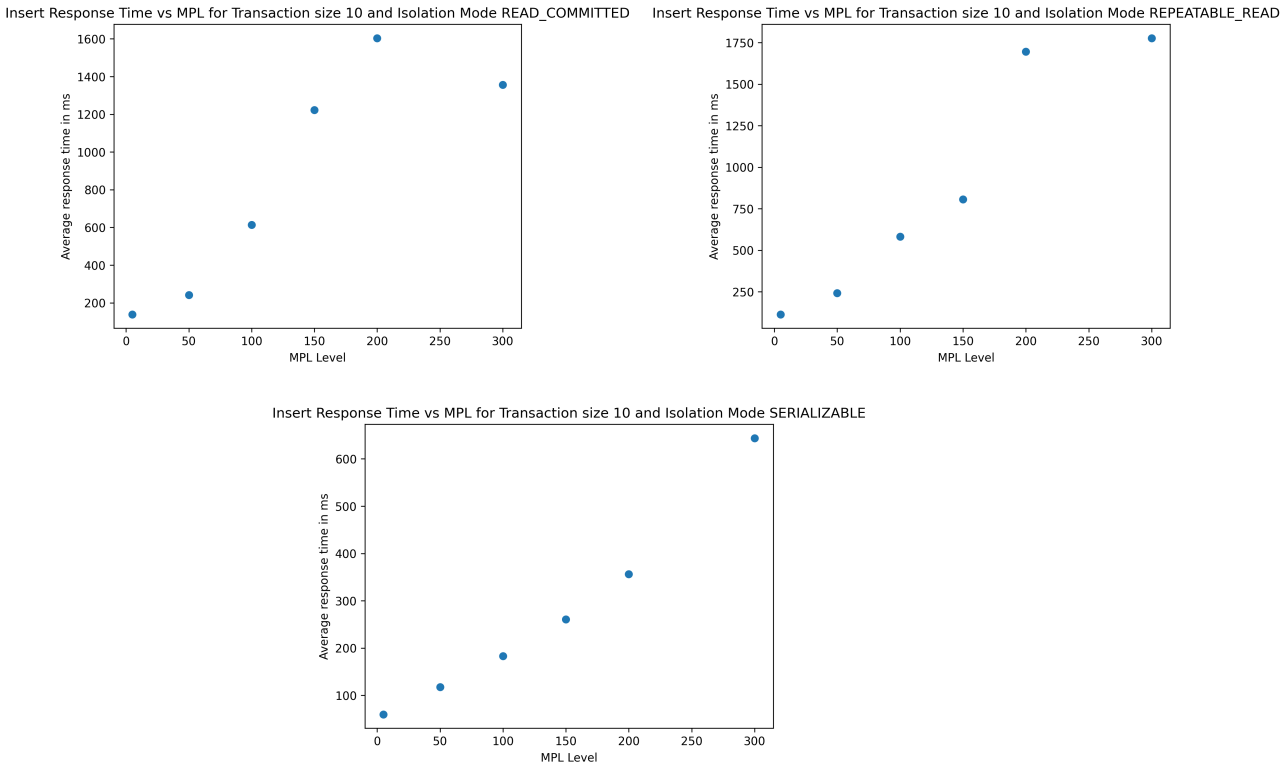

Figure 4.9: Clockwise from top left: Insert Response Time vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

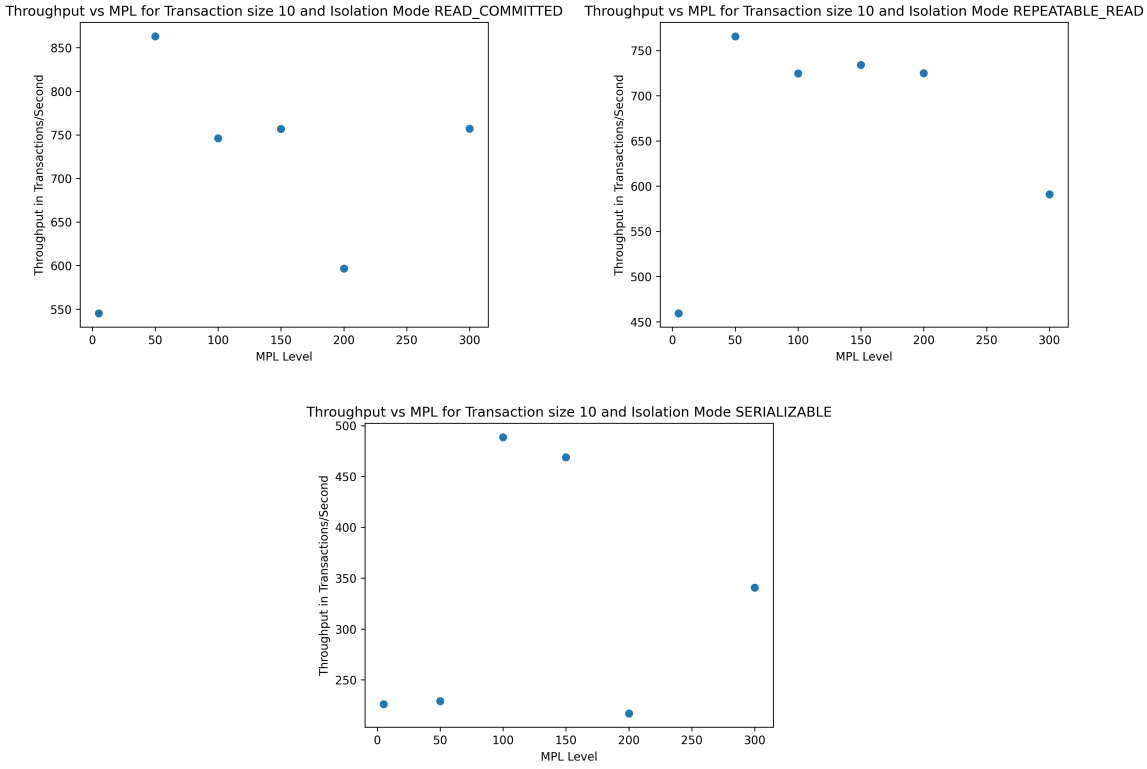

Figure 4.10: Clockwise from top left:Throughput vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

4.2 High Concurrency

4.2.1 Throughput vs MPL (txn sizes 2, 5 and 10)

From 4.10, We notice maximum throughput for 50 MPL for all isolation modes. Also we notice that serializable mode has the least throughput. The throughputs are less than the throughputs observed for low concurrency transactions.

From 4.11, we notice maximum throughput for 50 MPL for all isolation modes except serializable mode. Also we notice that serializable mode has the least throughput. The throughputs are actually more than the throughputs observed for low concurrency transactions!

From 4.12, we notice maximum throughput for 50 MPL for all isolation modes. Also we yet again notice that serializable mode has the least throughput. The throughputs are actually more than the throughputs observed for low concurrency transactions, except for serializable transactions. Transaction size of 2 has the highest throughput among the three transaction sizes.

.

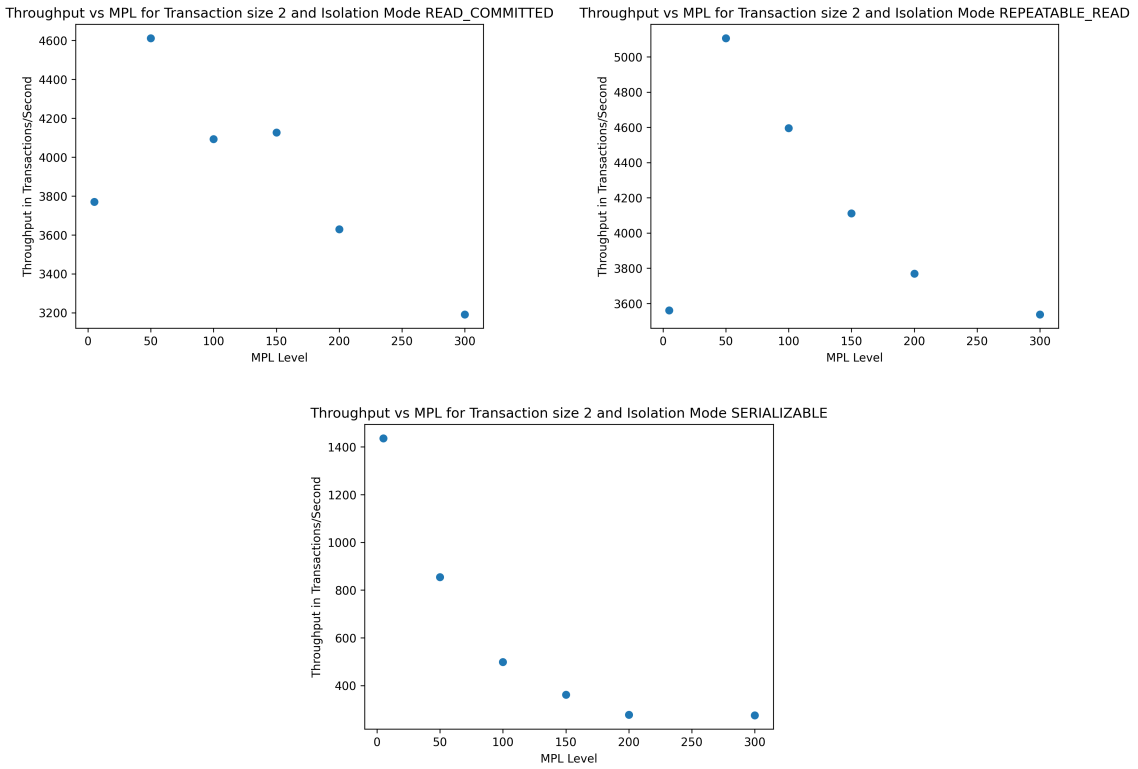

Figure 4.11: Clockwise from top left: Throughput vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

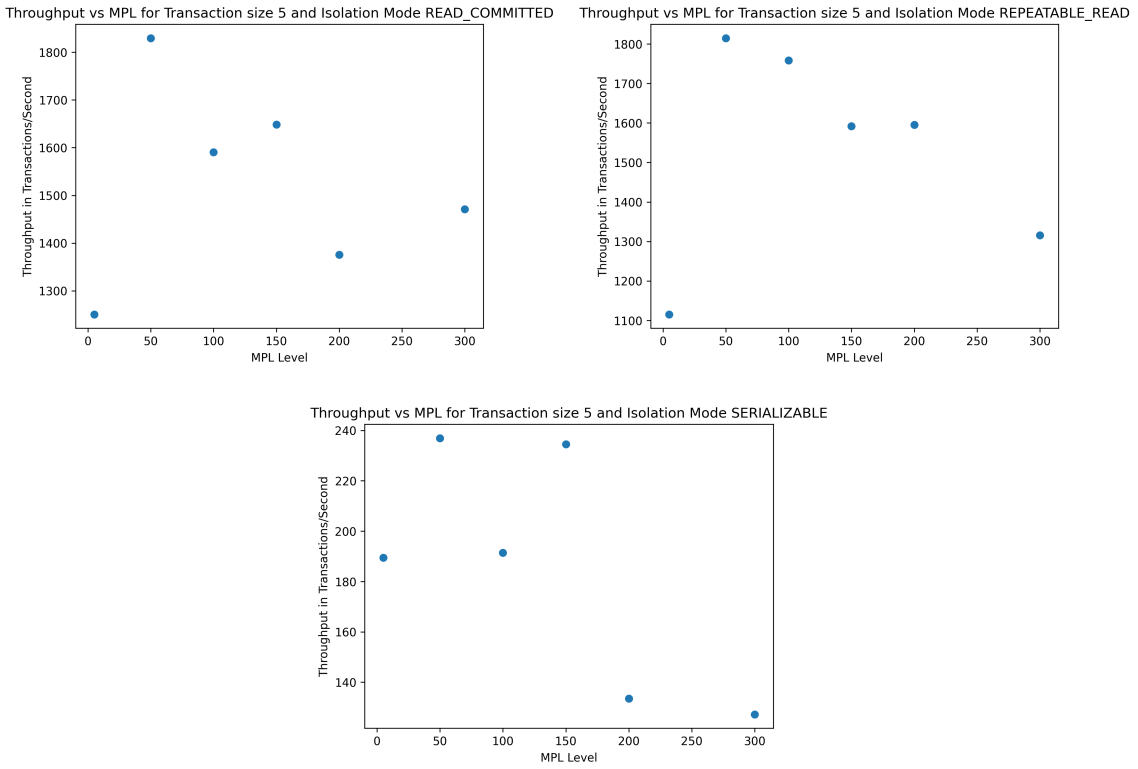

Figure 4.12: Clockwise from top left: Throughput vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

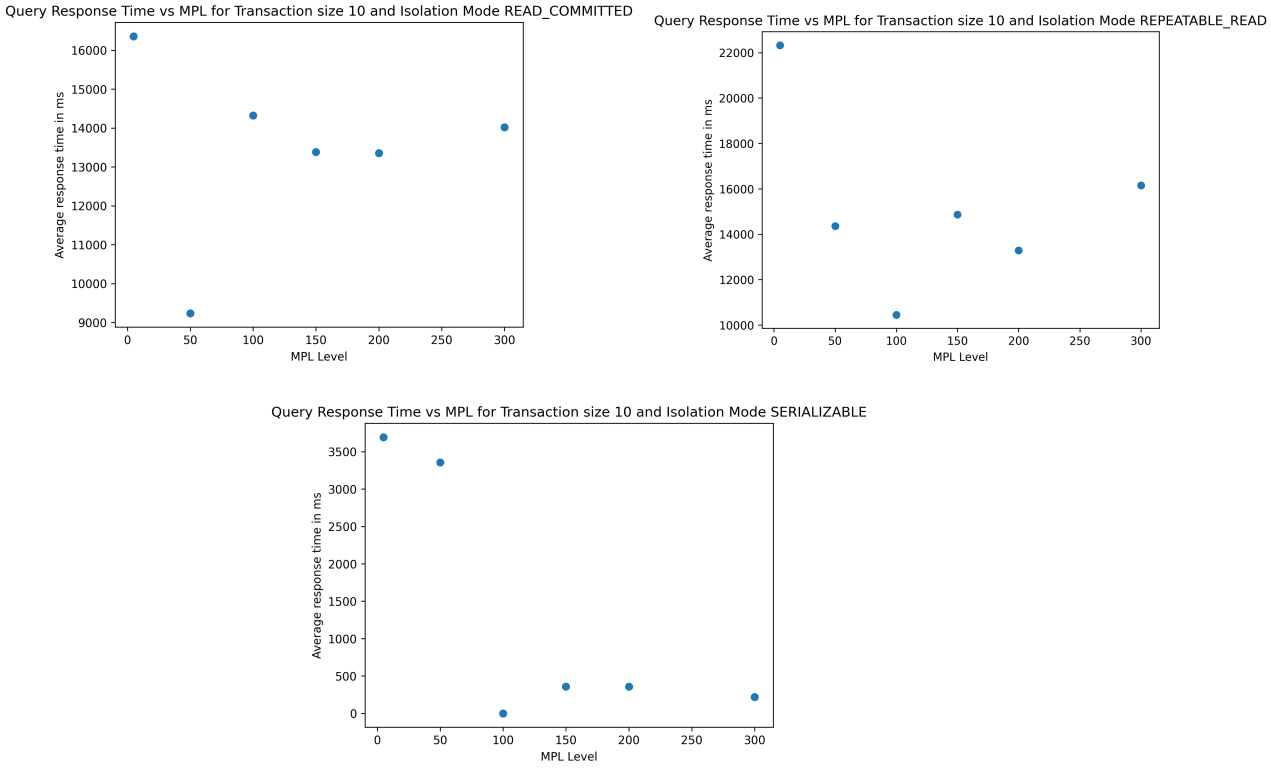

Figure 4.13: Clockwise from top left: Query Response Time vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

4.2.2 Query Response Times vs MPL (txn sizes 2, 5 and 10)

From 4.13, the response times for repeatable reads and read committed txns are comparable, while serializable transactions take considerably less time. We notice the best performance around 50/100 MPL levels for all three isolation modes. The times taken for high concurrency queries are several orders of magnitude higher than the times taken by the low concurrency queries.

Taking into account 4.14, We notice the best performance at MPL levels 50 and 100 for Read repeatable and read committed transactions, while it's exactly opposite for serializable transactions. Also serializable transactions take the most time among all isolation modes.

From 4.15 we notice the best performance at MPL levels 50 and 100 for all isolation modes. The performances of all isolation modes are quite comparable. Also transaction size of 5 performs slightly better than 2 and 10 mostly, although we get impressive response times for serializable transactions with transaction size 10.

4.2.3 Insert Response Times vs MPL (txn sizes 2, 5 and 10)

From 4.16 The response times for repeatable reads and read committed txns are comparable, while serializable transactions take considerably less time. Serializable transactions perform better with increasing MPL while for the other MPL levels, we notice the best

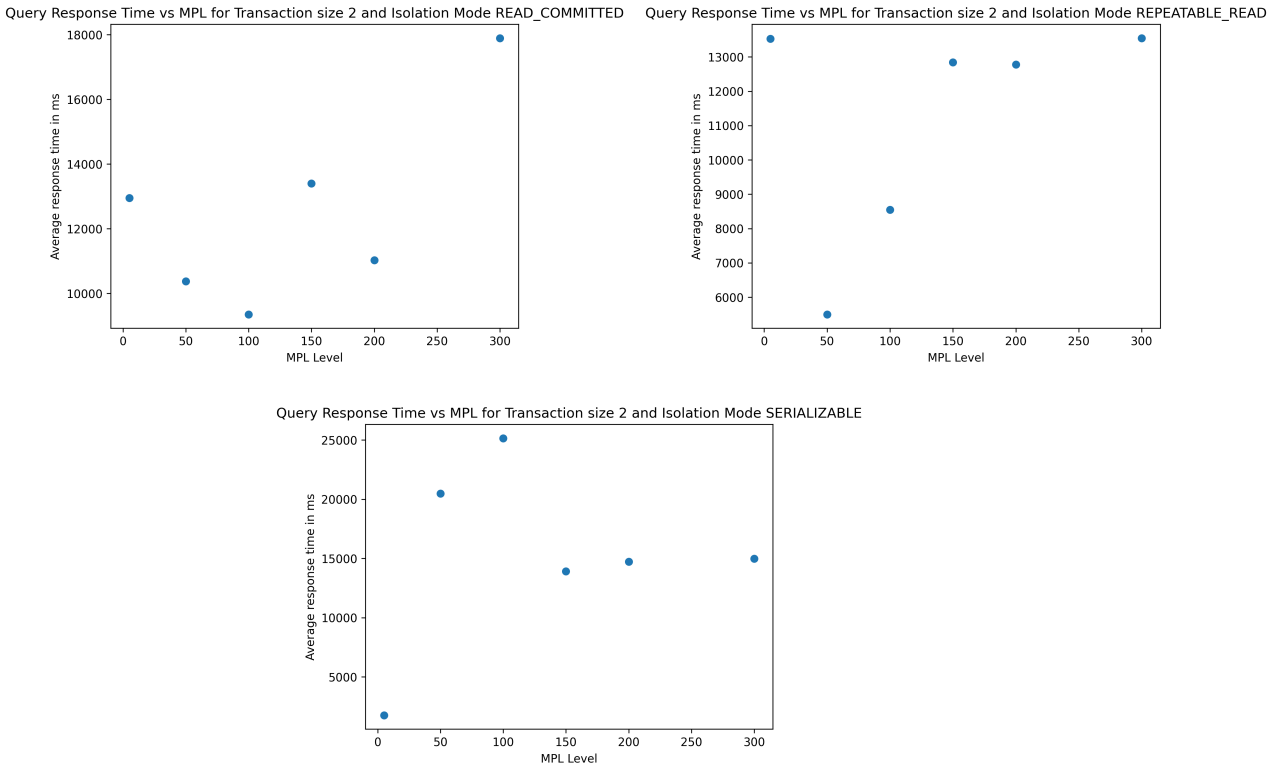

Figure 4.14: Clockwise from top left: Query Response Time vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

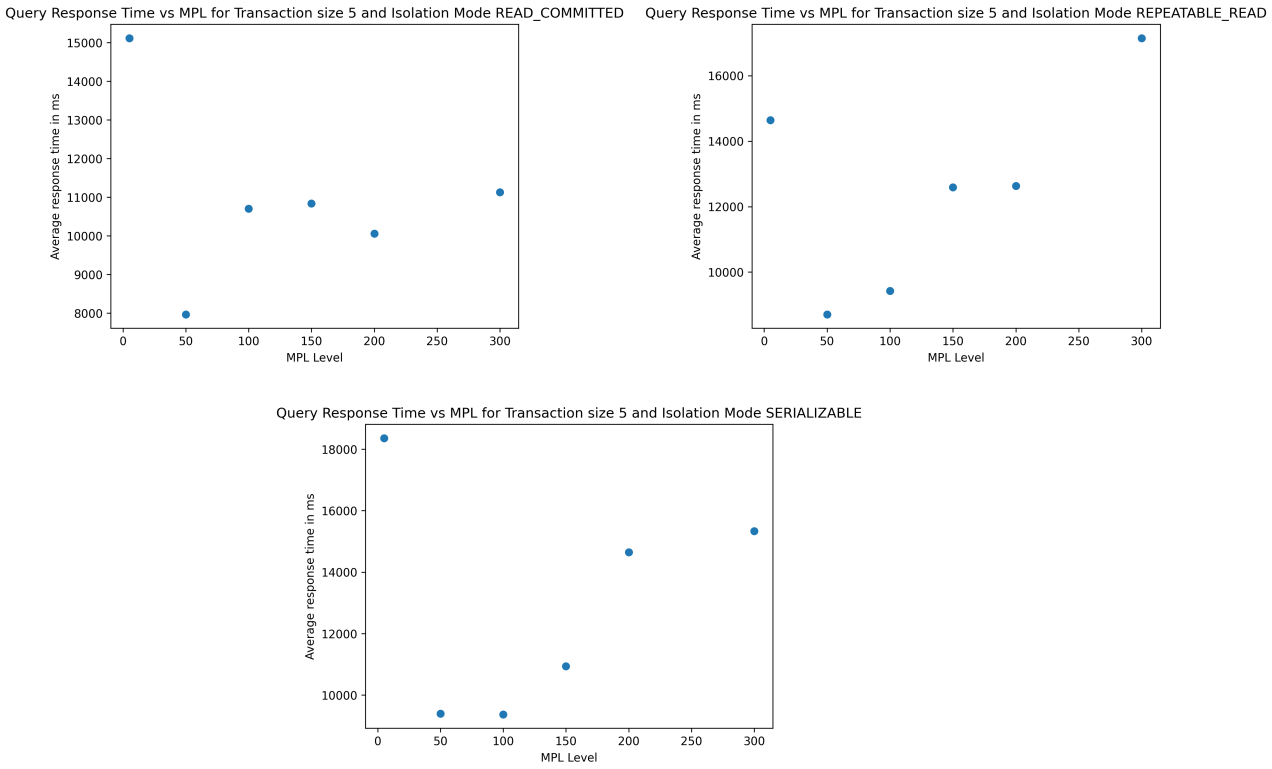

Figure 4.15: Clockwise from top left: Query Response Time vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

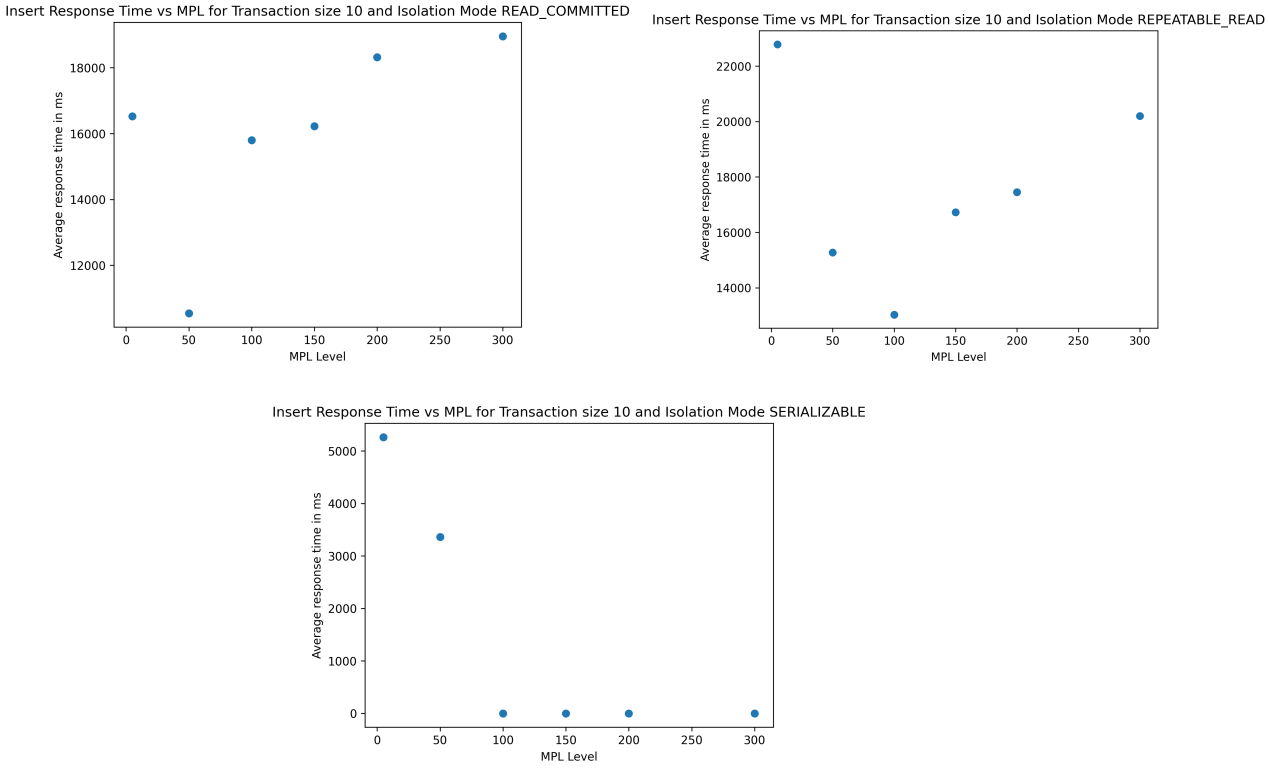

Figure 4.16: Clockwise from top left: Insert Response Time vs MPL for txn size 10 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

performance around 50/100 MPL levels. The times taken for high concurrency inserts are several orders of magnitude higher than the times taken by the low concurrency inserts.

From 4.17, here also, we notice the best performance for read committed and repeatable transactions are at the 50/100 MPL levels. But interestingly, serializable transactions degrade with increase of MPL level. Also, serializable transactions take much more time than what was observed for corresponding transaction size of 10.The times taken for high concurrency inserts are several orders of magnitude higher than the times taken by the low concurrency inserts.

From 4.18 we consistently see the best performance at 50 MPL level here . The average response times are better both txn sizes of 2 and 10 for read committed and repeated reads. Here too, serializable transactions take the most time. The times taken for high concurrency inserts are several orders of magnitude higher than the times taken by the low concurrency inserts.

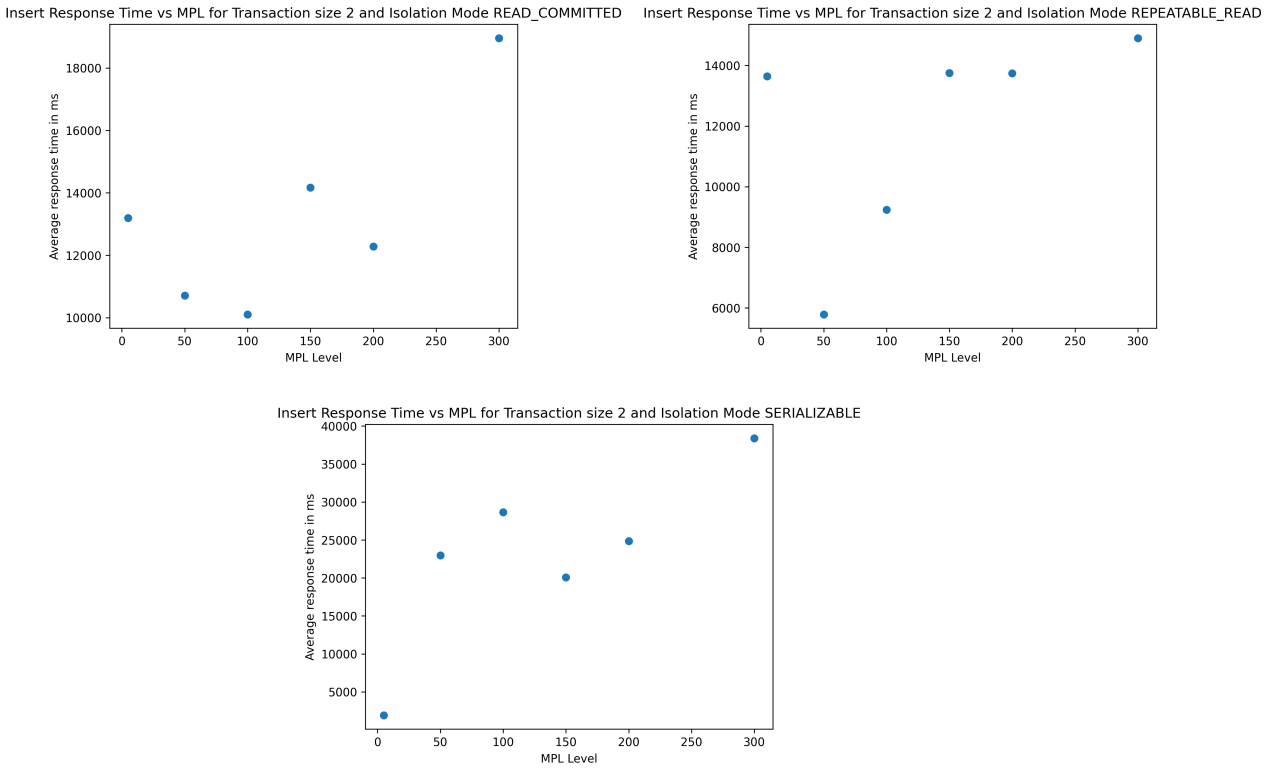

Figure 4.17: Clockwise from top left: Insert Response Time vs MPL for txn size 2 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable

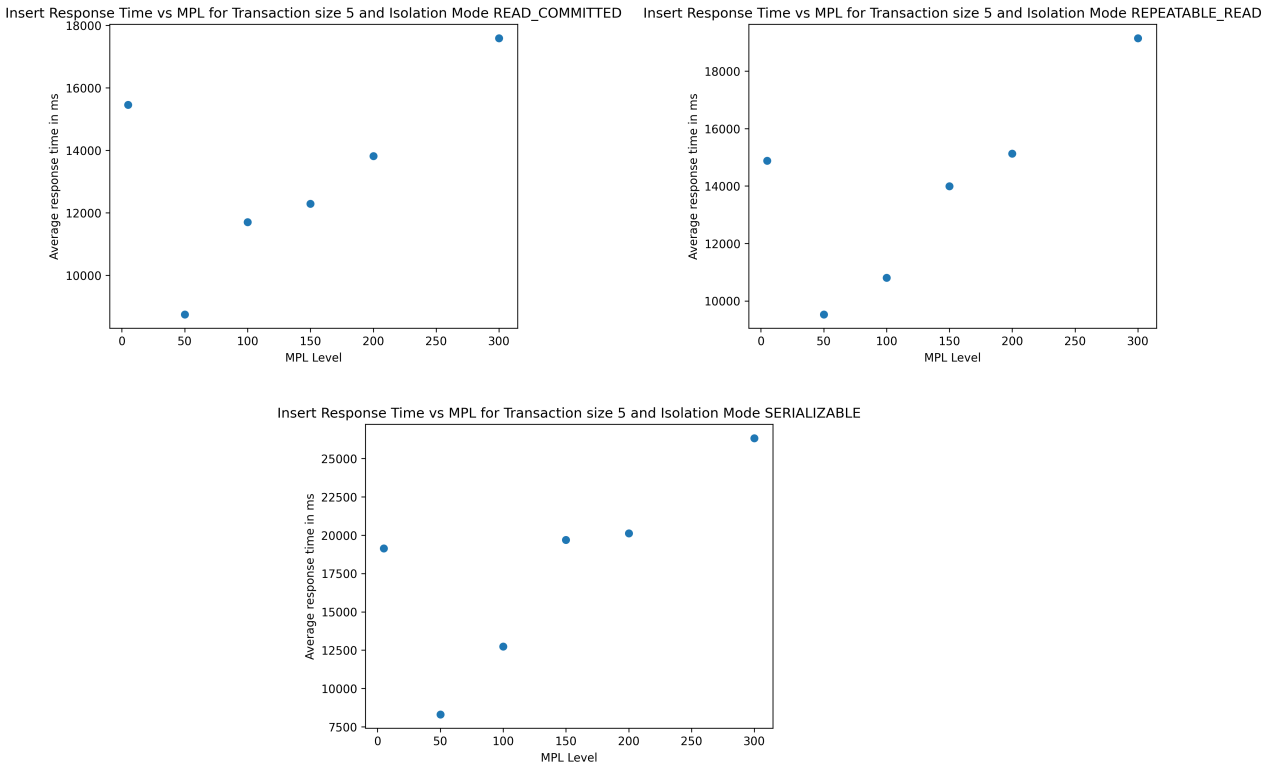

Figure 4.18: Clockwise from top left: Insert Response Time vs MPL for txn size 5 and modes i) Read Committed, ii) Repeatable Read and iii) Serializable