COMP 512 Distributed Reservation System overview.

Project Report II

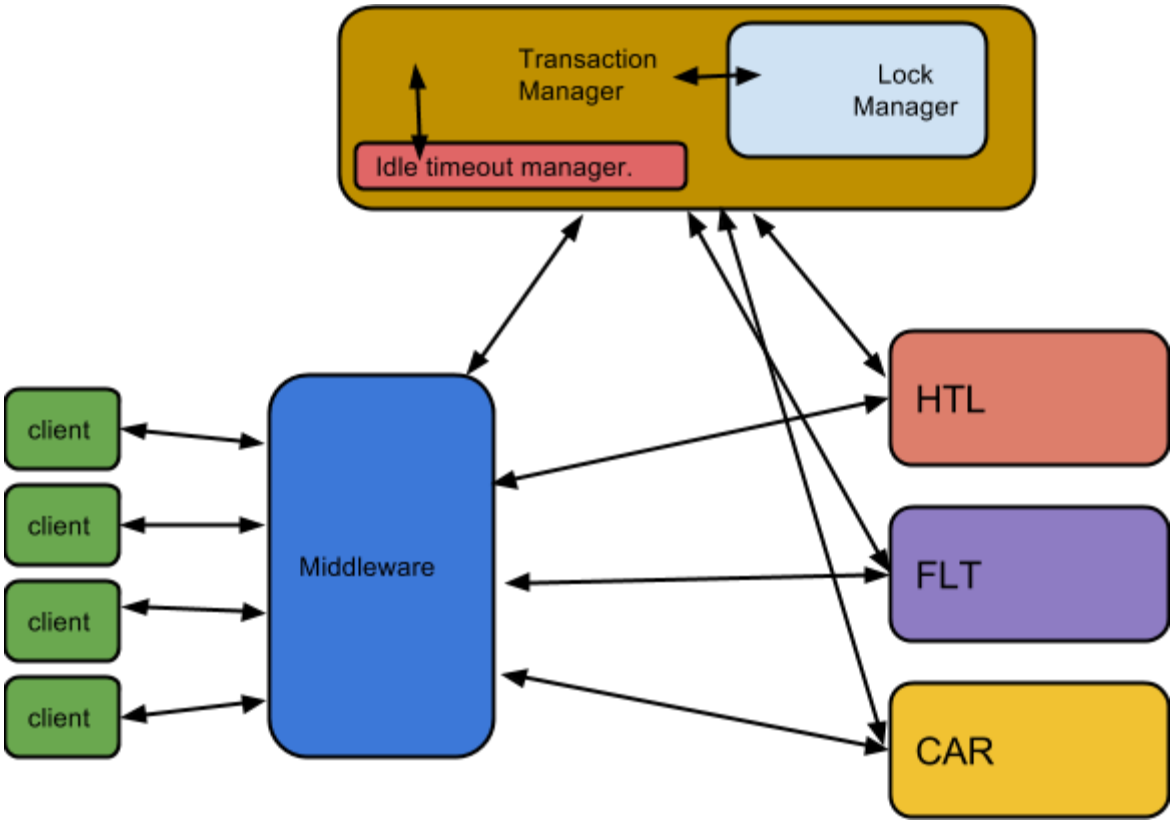
Submitted by

Joseph Vinish D'silva
Dhirendra Singh

Table Of Contents

System Architecture

The system architecture of the reservation system is as depicted in the diagram below



Client request are sent to the middleware that communicates with the Transaction Manager and the resource RMs.

The centralized lock manager forms an internal component of the Transaction manager.

When a client initiates a transaction, the middleware forwards the request to the transaction manager that generates a unique transaction id and returns it to the client.

Any further communications from client references this transaction id.

When an RM receives a request, it will look in its transaction buffers whether the transaction is present. If not, it will ask the Transaction Manager to register itself as a participant in that transaction.

The RM also issues a lock request for the object requested, to the Lock Manager and waits (till DEADLOCK timeout) for a lock. Upon obtaining a lock successfully, it will continue to process the data.

For read requests, RM first checks if the data is available in the transaction specific buffer. If found it's returned from the transaction buffer. Otherwise the data is read from the main data store and returned.

For write requests, the data is read into the transaction buffer if not already present (a new copy is made) and this copy is returned. All modifications are performed on this copy.

When the client initiates a commit, middleware passes on the request to transaction manager, who in turn sends it out to any RMs that had registered for the transaction. The RMs flush the copy from the transaction buffer for any modified data and then discards the transaction buffers.

For aborts, either initiated by the client or by the Transaction manager for idle timeouts, the RMs simply discard their transaction buffer.

The Transaction Manager also keeps track of transactions that has been idle for a while and aborts it freeing the locks. If the client at some point in time tries to continue, a transaction aborted exception will be thrown.

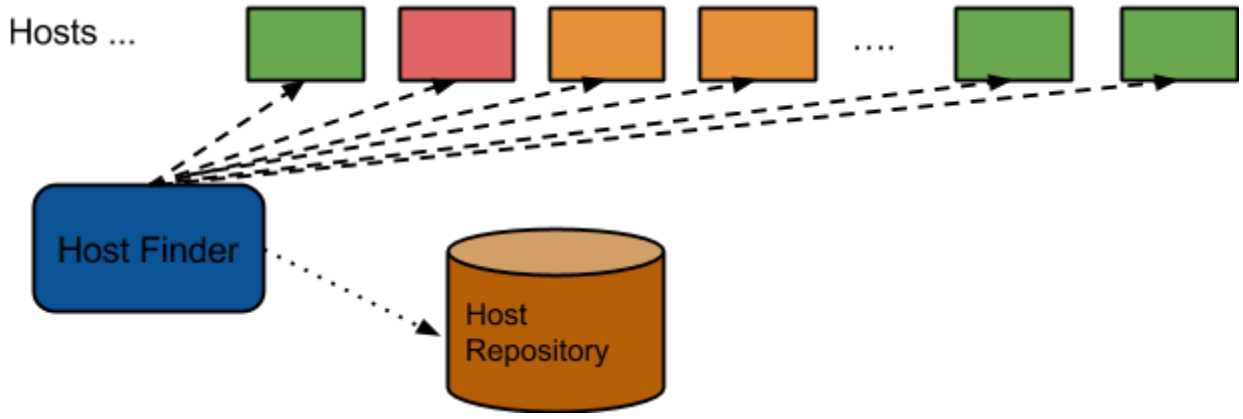
The deadlock and idle timeout intervals are externally configurable.

Reserve Itinerary.

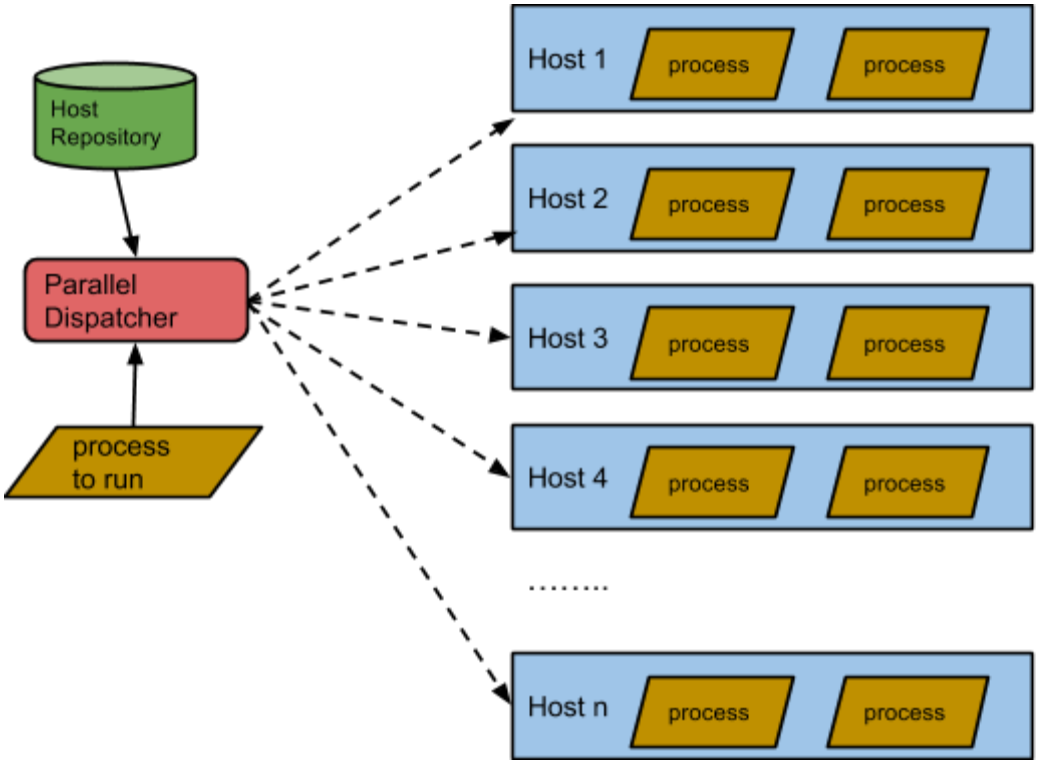
Reserve itinerary has special implementation in the sense that the middleware always request the RMs to confirm the availability (who puts a write lock), however no actual modification takes place at this step. If all the RMs confirm on availability then the changes are made. If any RM is not able to confirm the availability of resources, no modifications are made and the locks are released when the transaction is committed/aborted.

Test Environment setup.

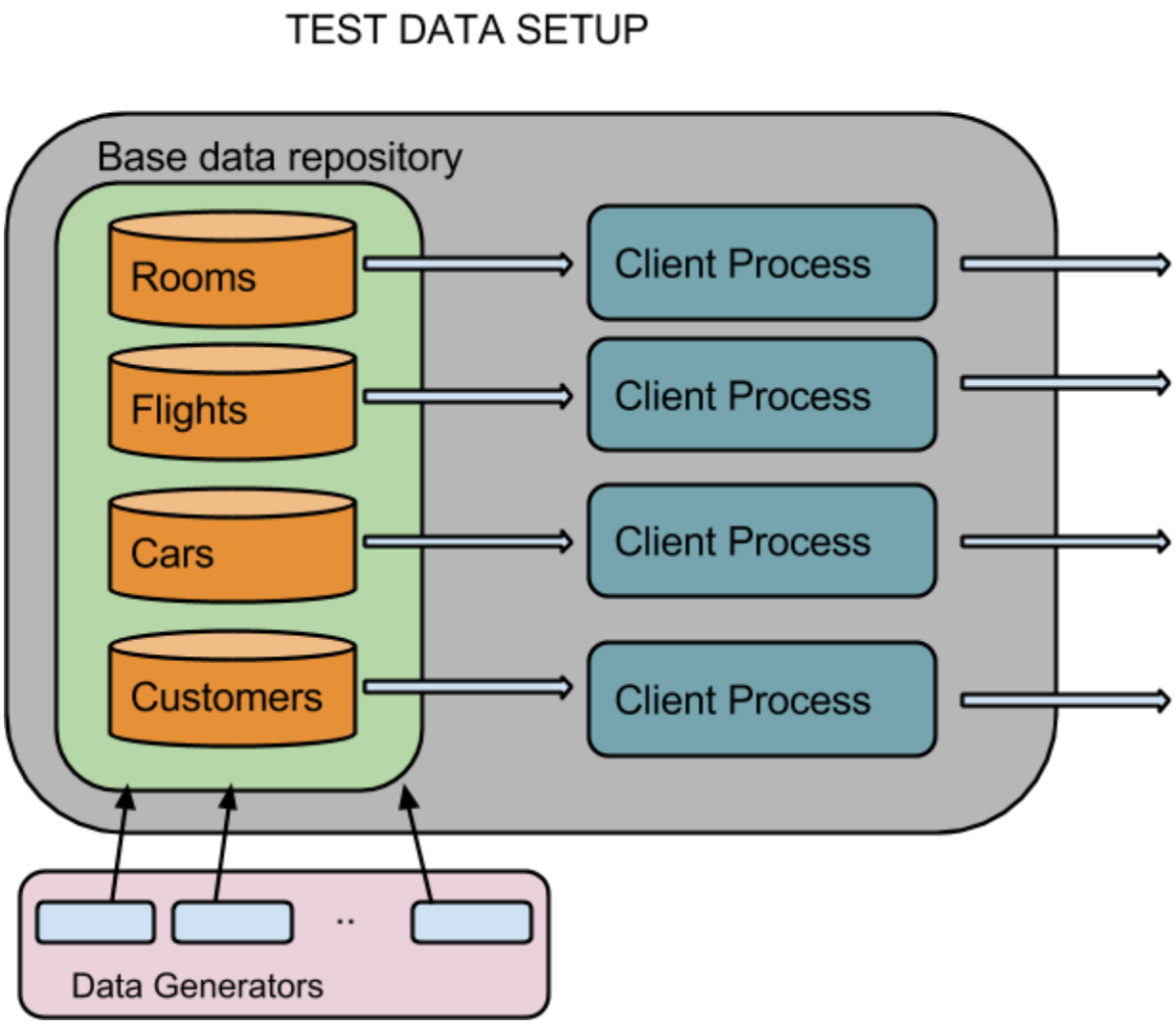
The infrastructure for parallel test setup is as described below



The host finder can be used to locate hosts in the network and their load averages to pick suitable hosts for parallel testing environment.

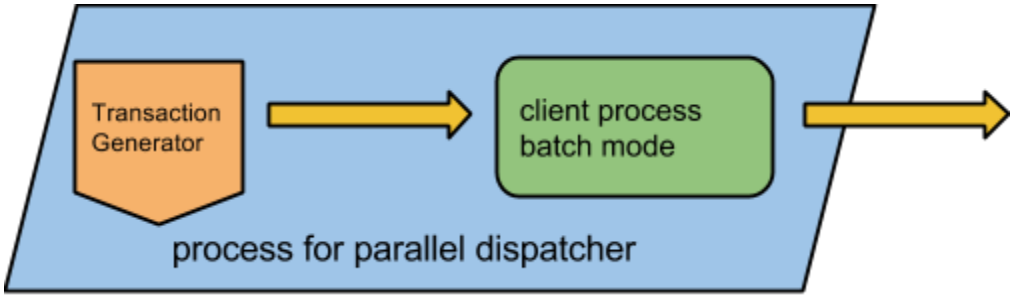


The parallel dispatcher is a program that can execute multiple instances of a script in multiple servers depending on the amount of parallelism required. We make use of this process to have multiple clients launched from different servers.



The base data repository is used to refresh the test environment to a known configuration of data. All test cases will be executed on top of this configuration. This way, we can ensure that all test cases have a known starting point of data configuration.

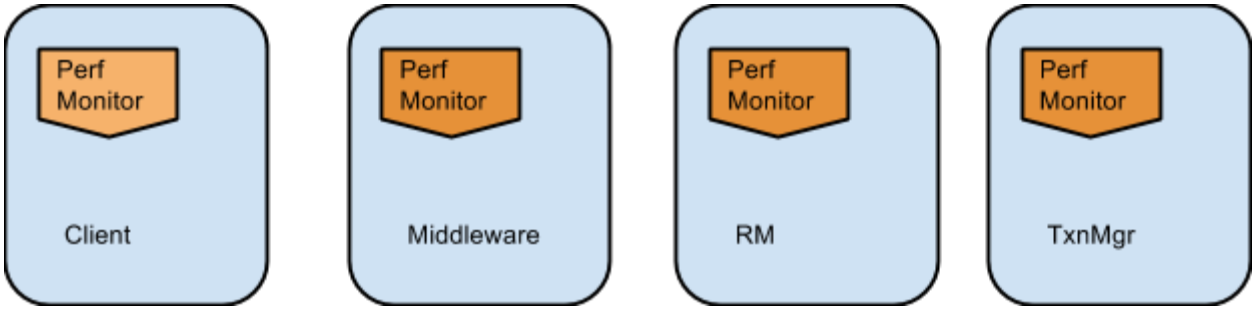
The transaction generator is used to generate transactions that are then fed to the client process, which can be configured to run on batch mode. In this mode it is possible to pass a option to limit the number of transactions that are executed per unit time.



The Performance monitor is a component embedded into all of the elements in the reservation system. It is used to record the starttime and endtime of an operation and print out the response times upon completion of the operation. This information is in CSV format. That can be used by charting utilities like Excel, MATLAB etc.

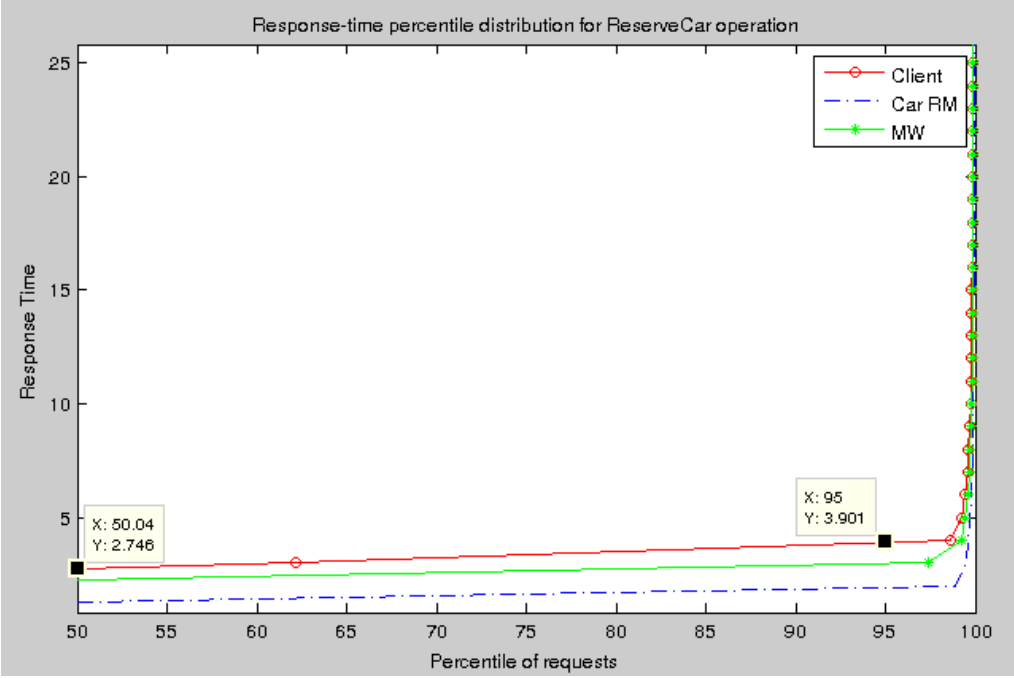
The component can be enabled/disabled externally by setting the appropriate environment variable.

The test suite can be seeded by a test identifier, in which case all the test data generated will be tagged with it. This makes it easier to identify a specific test case with each a performance record is to be identified with.

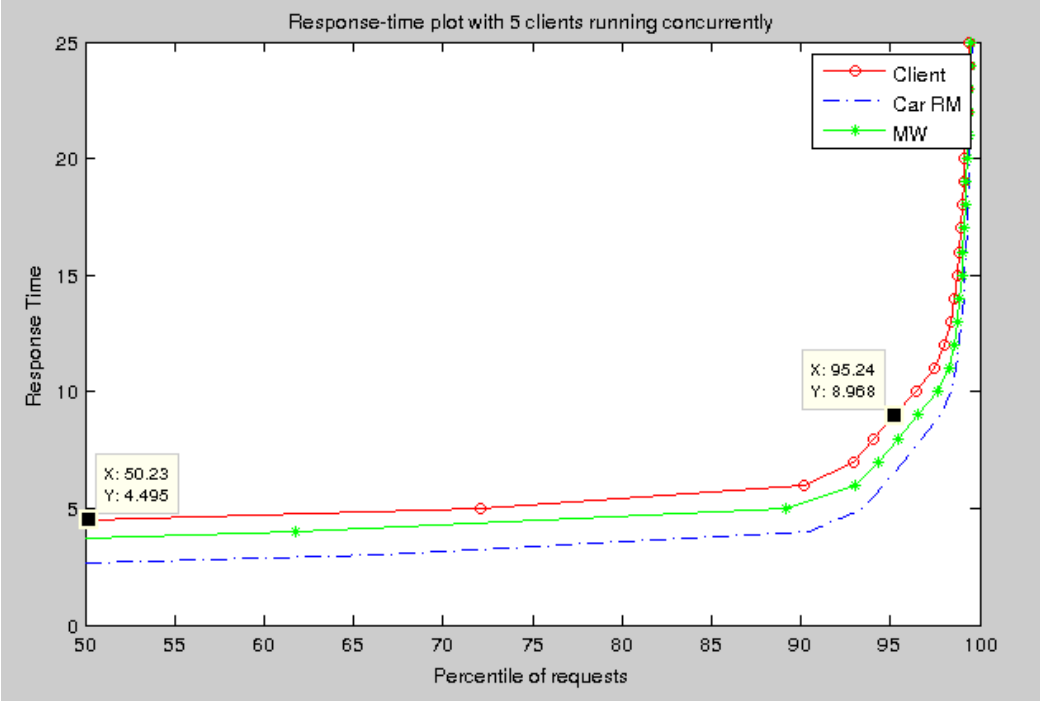


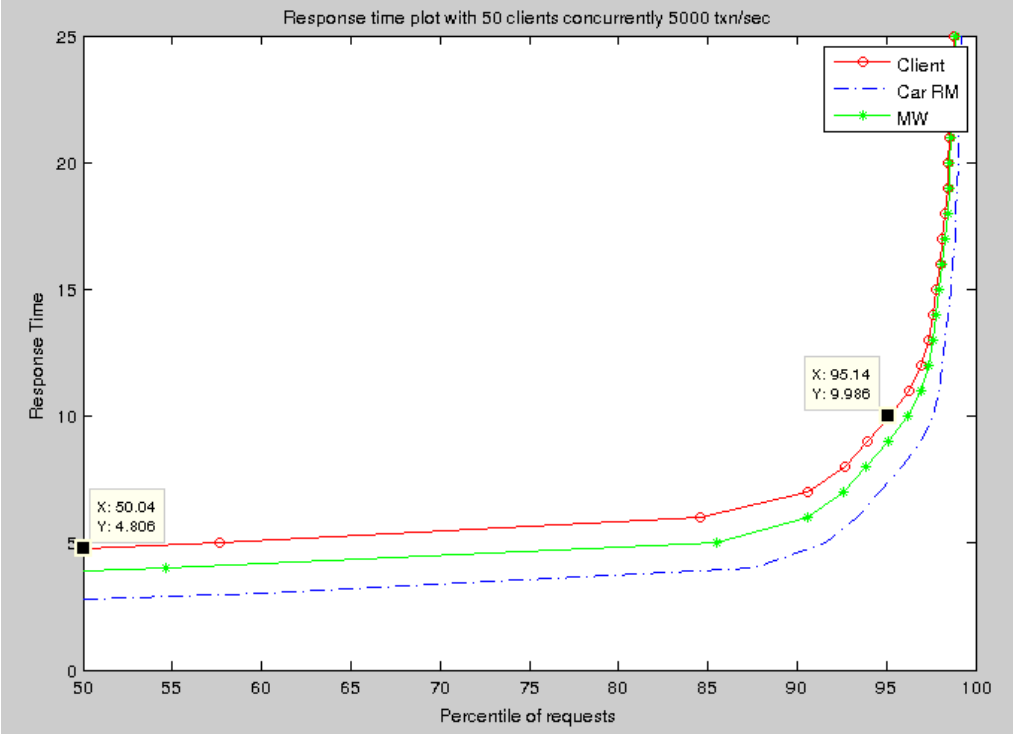
Test Execution

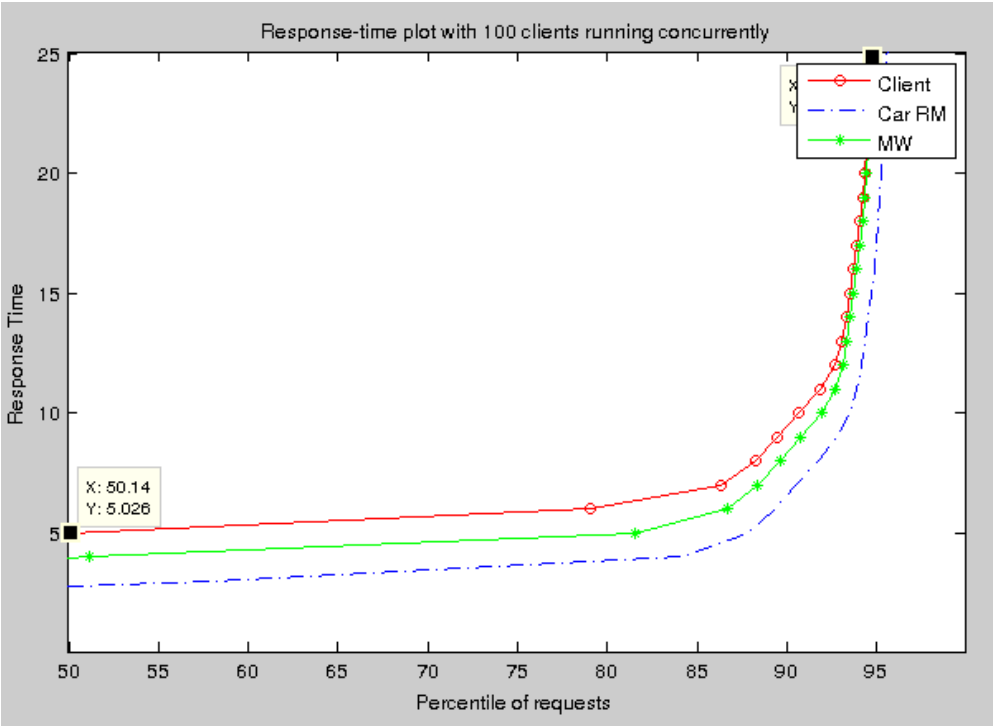
By executing a single client without any wait period between transactions, it was noticed that 50% of the requests completed under 2.75ms and 95% under 3.9ms



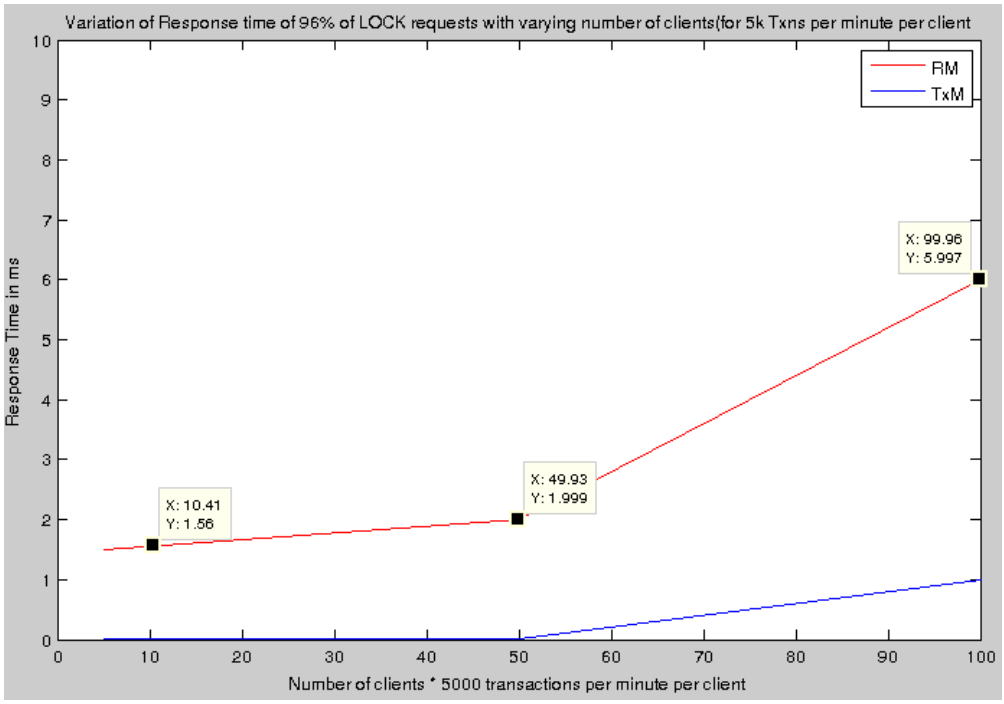
Execution of 5000 txn / Sec







We notice that as the number of clients increased from 5 to 100, the 50% of queries still give good performance, however we notice the outliers at 95% increases. While for 5 to 50 the changes are not significant, the effects are more noticeable as we increase the clients to 100.



We can notice the proportional increase in lock request time in the system, pointing to lock contention and deadlocks occurring in the system at higher concurrency.