CCTS: Project Report

* T. Rohan; CS20BTECH11064
* S. Kotesh Satvik; CS20BTECH11048

**Algorithm – 1**

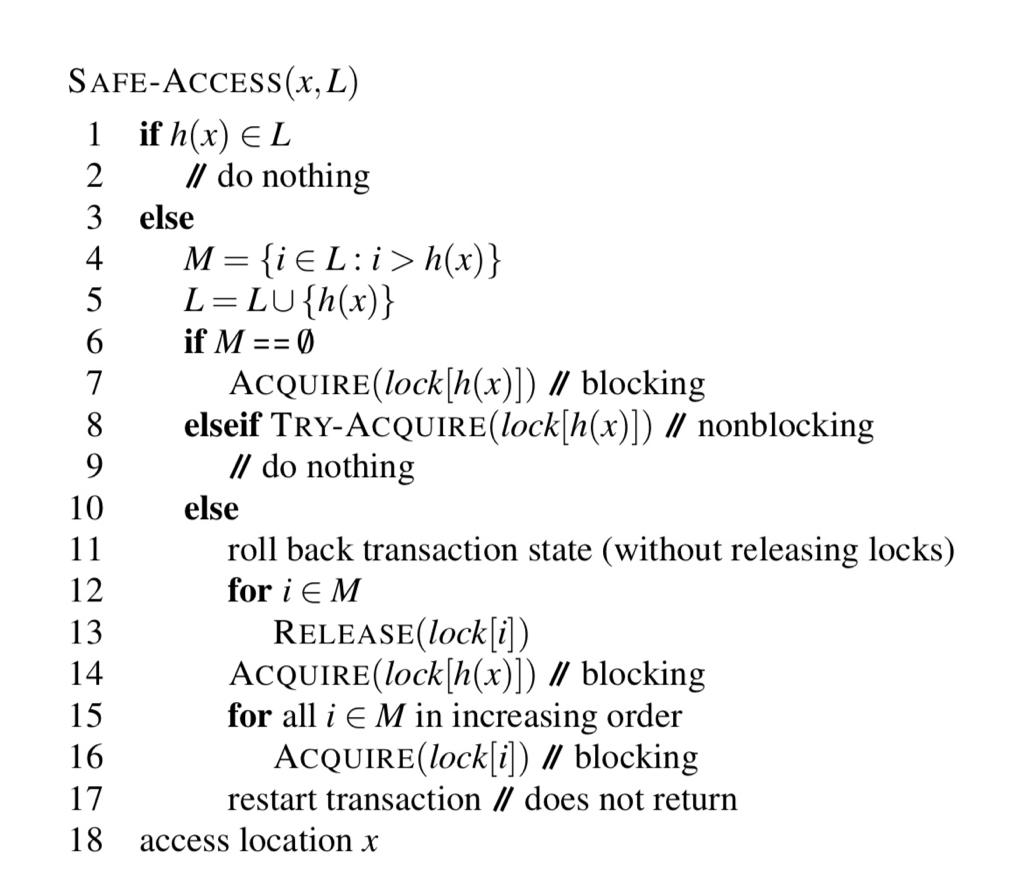
**The L Algorithm –** A deterministic Algorithm for Guaranteeing the Forward Progress of Transactions

**The Algorithm:**

* The algorithm employs a finite ownership array of n locks, which is a global array accessible by all the transactions.
* An arbitrary mant-to-one owner funtion ‘h’ maps the set of all shared-memory locations to one of the n slots in the ownership array.
* The finite ownership array introduces the possibilty of a false conflict, where two transactions accessing different locations conflict by requiring the same lock.
* Before accessing a shared-memory location x, a transaction must acquire the lock in the ownership array associated with x.
* Each transaction maintains its own local set L of lock indexes, which starts out as an empty set.

**Test Application**

* A shared map of int to dataItem is used as a shared memory for the transactions.
* The number of concurrent transactions are the number of threads being used for the transactions.
* Each transaction operates on random data items.
* Size of shared map – numVar
* Size of ownership array – numLocks
* Number of transaction - numTrans

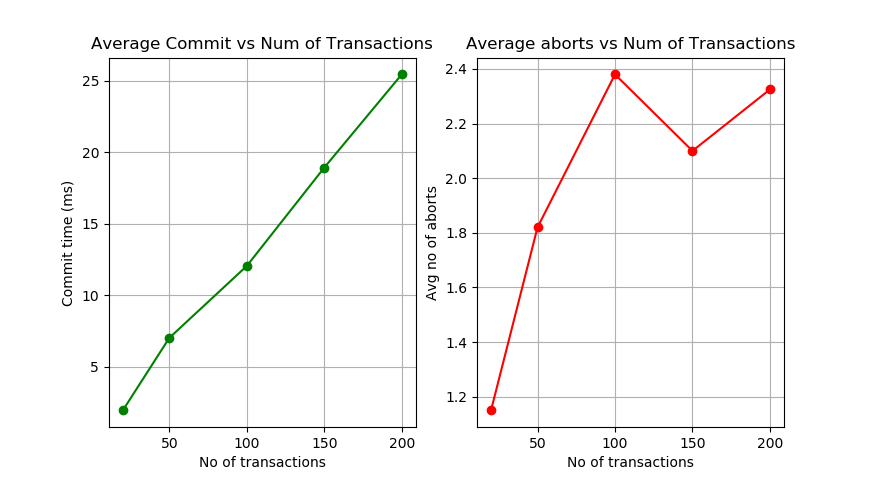


**Pseudo-code**

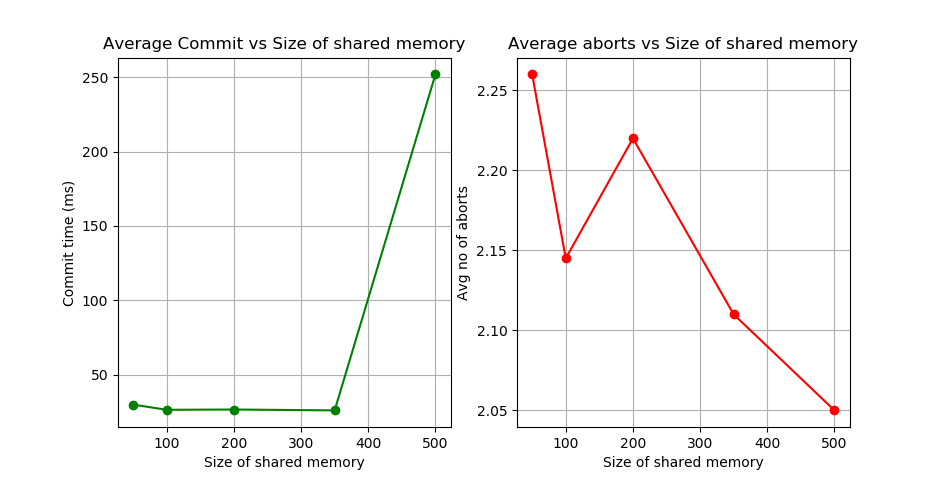
**Advantages**

* It is a deterministic (not probabilistic) contention management algorithm for guaranteeing the forward progress of transaction
* The contention manager ensures that the system does not deadlock, where transactions are caught in a cycle of waiting and cannot progress.
* The contention manager ensures that the system does not livelock, where transactions are repeatedly aborted and restarted without making progress.

**Graphs 1 and 2**: The following graphs represents Avg. commit time in microsecs and average no. of aborts on y-axis respectively and no. of transactions on x-axis.



**Graphs 3 and 4:** The following graphs represents Avg. commit time in microsecs and average no. of aborts on y-axis respectively and size of shared map on x-axis.



**Algorithm – 2**

**Virtual World Consistency Protocol**

**Virtual World Consistency:** Virtual world consistency is a condition that requires

* All committed transactions to be serializable
* Each aborted transaction (reduced to a read prefix) reads values that are consistent with its respect to its causal past, in other words, if there is a legal linear extension of the partial order past.

The formal definition of virtual world consistency is based on a total order on the committed transactions and a partial order on the whole set of transactions (where each aborted transaction is reduced to a read prefix).

**VWC – Protocol:** This section presents a protocol, called VWC-Protocol, that implements the virtual world consistency condition. The implementation of key features of the protocol are as follows:

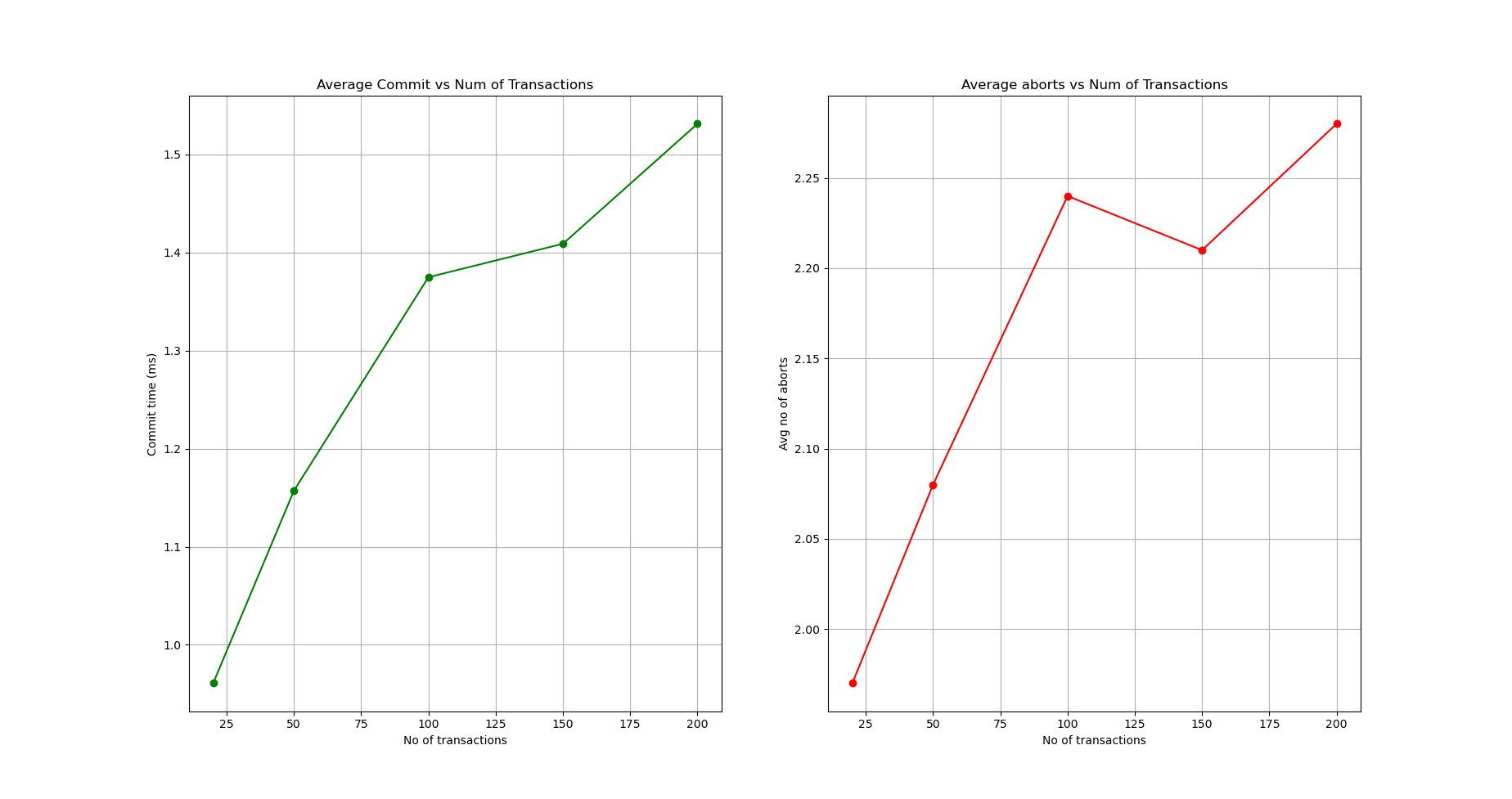
* **Struct dataItem:** Each data item is made up of id of the object, value of the object, its read list, its write list, its mutex lock and a map that tracks its value dependencies. If a sequence number can be seen as a logical date associated with an object, X.depend[X] is the sequence number of current value of X, while X.depend[Y] (Y ̸= X) is the sequence number of the value of Y on which the current value of X depends.
* **Class transaction:** A transaction is made up of its id, status(live, committed or aborted) its read set, its write set, time values indicating its start and end, and since a transaction has to work with object values that are not older than the ones used by the previous transactions, it maintains a local map *depend[1…m]* such that *depend[X]* contains the sequence number of the last value of X that is known by the transaction. It also maintains local copies for each data item.
* **Map – Shared:** A map acting as the shared array.
* **Begin\_trans():** It creates a transaction object with an updated unique transaction id
* **Read():** It takes in as parameters the id of the transaction that wants to read it, the id of the data item it wants to read, and the location to which the value has to be read. If there exists a local copy of the data item in the transaction, it returns that. If it was the first read of that data item by the transaction, it builds a local copy and updates its read sets and depend maps accordingly. However, suppose if Y was an object that has been previously read by the transaction, then the sequence number of the value of Y read is kept in depend[Y]. If the value of X just read by the txn depends on a more recent value of Y, the values of X and Y are mutually inconsistent. Hence the transaction aborts.
* **Write():** If the local value of the data item exists, it updates the value, else, includes the data id in the write set of the txn.
* **tryC():** The transaction first locks all the objects it has accessed. If the transaction was a read-only transaction and if the values it has read have not been overwritten, then the transaction is committed, and if not, then it is aborted. If the transaction was write-only, it updates their dependencies in the shared memory and commits.

If neither, it first checks if all the reads are consistent the same way it does in read function and if found true, continues with updating the dependencies and writes the objects and commits, else it aborts.

**Test application:** Since the applications of the L-algorithm and VWC protocols are quite different, it would not be possible to compare them using the same test application. Hence we use the updtmem() function used for programming assignments to test the applicability of the VWC-Protocol.

Given below are two graphs indicating how the commit times and no. of aborts vary with increasing number of transactions and with increasing size of shared memory respectively.

**Graphs 1 and 2**: The following graphs represents Avg. commit time in microsecs and average no. of aborts on y-axis respectively and no. of transactions on x-axis.



**Graphs 3 and 4**: The following graphs represents Avg. commit time in microsecs and average no. of aborts on y-axis respectively and size of shared map on x-axis.

