# PROJECT REPORT

## DEADLOCK FREE ALGORITHM FOR TRANSACTIONAL MEMORY

Sumanth - ee17btech11041
Srinivas - cs17btech11009

## Introduction

Algorithm L is lock based contention management algorithm, which guarantees forward progress. Advantage over the other lock algorithms studied in class is that transactions in this algorithm will not deadlock.

## Explanation of the algorithm`

Before accessing a shared-memory location x, a transaction must acquire the lock in the **ownership array** associated with x.
An arbitrary many-to-one **owner function (hash function) h** : $U \rightarrow \{0, 1, \ldots, n-1\}$ maps the set U of all shared-memory locations to one of the n slots in the ownership array.
To acquire the lock associated with x, the transaction may perform one of two operations :

- ACQUIRE (lock[h(x)]), which blocks on the lock acquisition until the lock becomes free.
- TRY-ACQUIRE (lock[h(x)]), which either successfully acquires the lock and returns the Boolean TRUE , or fails and returns FALSE.

Each transaction maintains its own local set L of lock indexes, which starts out as the empty set.
Whenever the transaction wants to access a new memory location x in the shared memory it attempts to acquire to lock[h(x)] and add h(x) to L as follows

- If h(x) < largest value in L, then the transaction aborts if the lock[h(x)] is held by another transaction.
- If h(x) > largest value in L, the transaction blocks if lock[h(x)] is taken. Once the transaction acquires the lock, it performs the access of x.

If an abort occurs the transaction performs rollback, and then releases all the locks in L with index > h(x), and waits till it acquires lock[h(x)], then it reacquires the locks it previously held in order.Then the transaction is restarted.

## Test application -

The test application developed to test this algorithm is as follows :

- A shared map is used as a shared memory for multiple transactions.

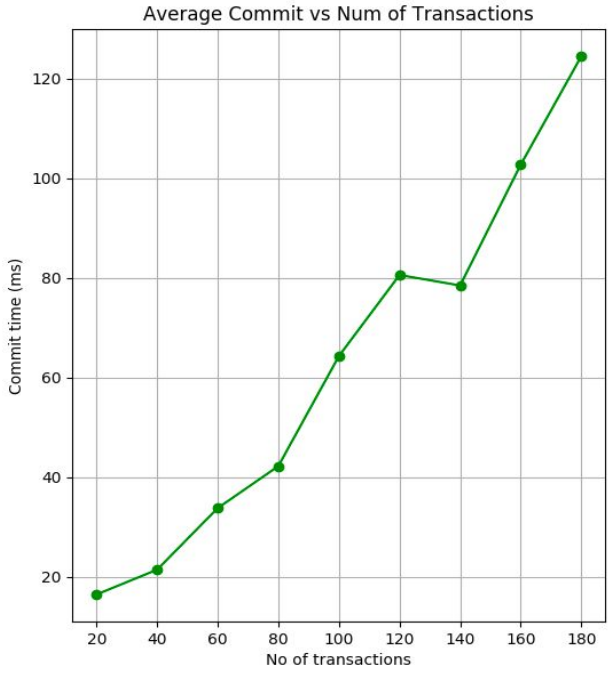
- There are m threads -> m concurrent transactions.
- Each transaction has some random number of operations which include reads and writes to the map.
- The size of the shared map - n
- Size of ownership array - p

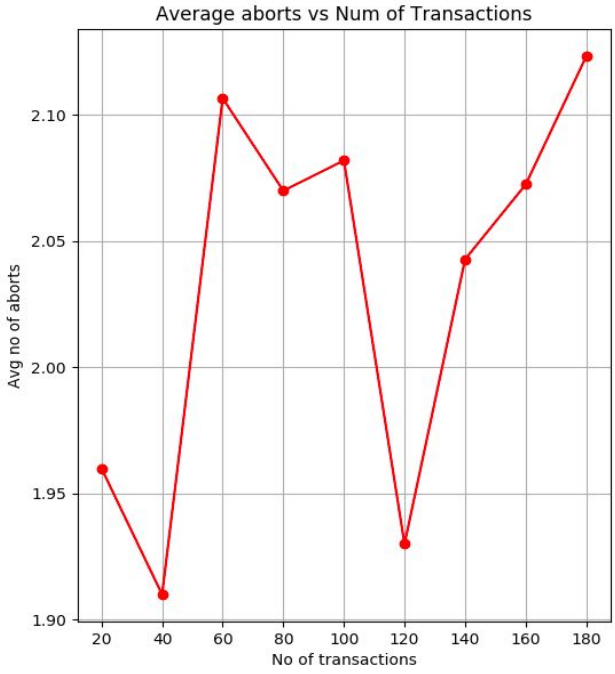
## Observations

The average commit time and the number of restarts per transaction are taken as the parameters to evaluate the algorithm.

- Average commit delay and abort counts with varying number of Transactions

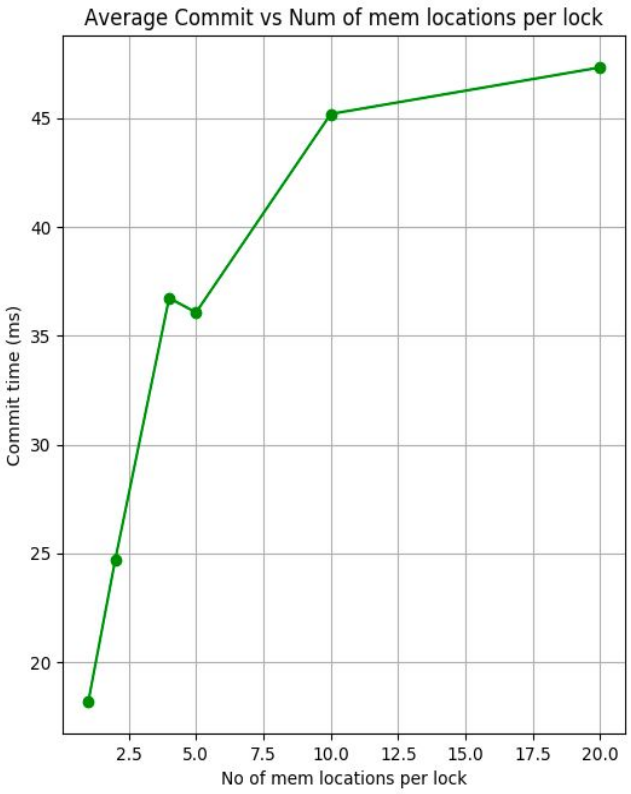
  Shared Memory Size = 100
  Ownership array size = 50
  Lambda = 100 (used in generating random distribution)

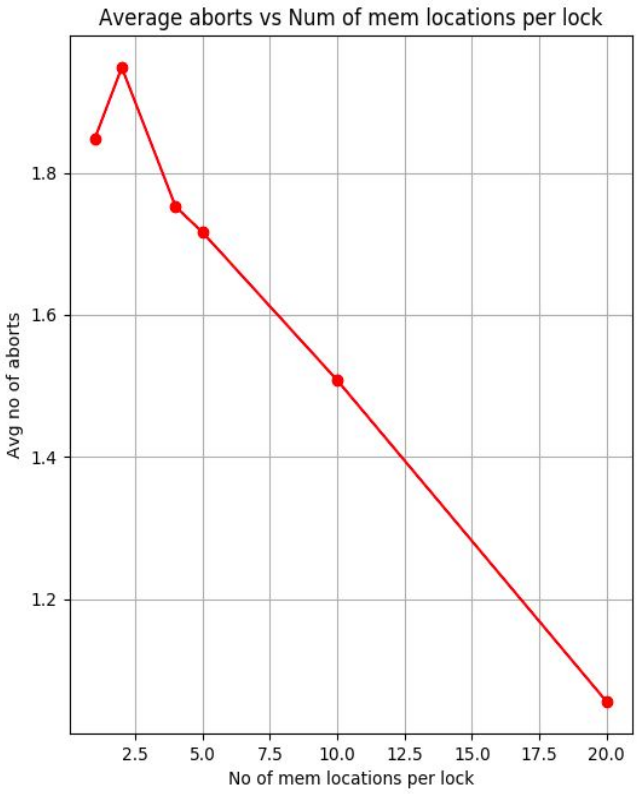


1. The commit delay increases with the number of concurrent Transactions as many Transactions compete to acquire locks.
2. Average Abort count remains nearly same with increase in number of Transactions.

- Average commit delay and abort count with varying size of Ownership Array (total number of global locks)
  Number of Transactions = 50
  Shared Memory Size = 100
  Lambda = 100
  Size of Ownership Array determines the number of memory locations mapped to each lock.



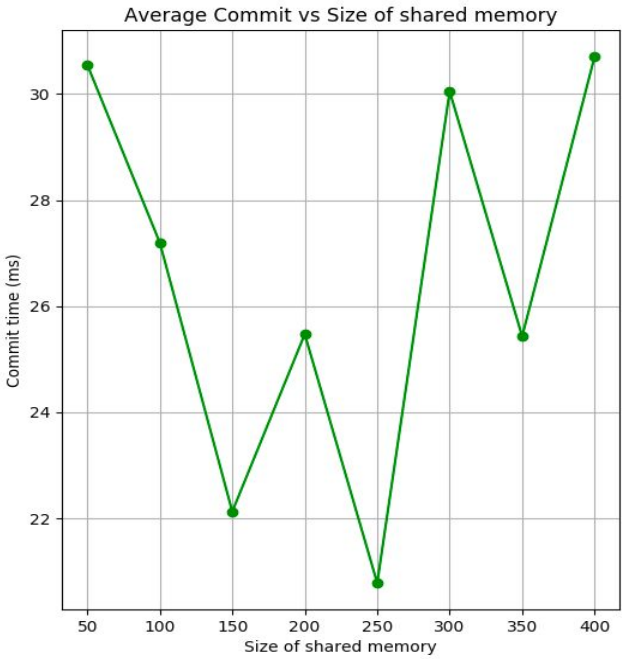



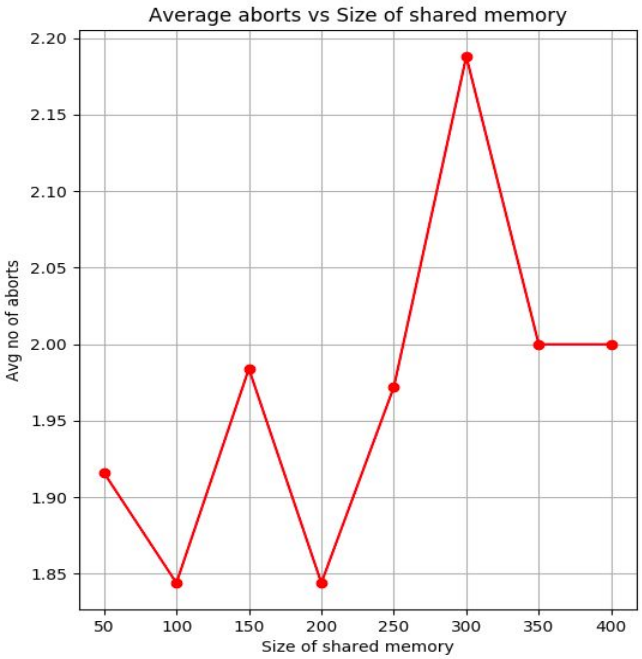

1. The commit delay increases as we decrease the size of OwnershipArray as there will be more collisions in the hash function and different locations map to the same lock.

2. Abort count decreases with decrease in size of Ownership Array as the concurrency decreases.

## Average commit delay and abort counts with increasing size of shared memory

Number of Transactions = 50
Ownership array size = 50
Lambda = 100




1. The performance of the algorithm doesn't get affected by the size of shared memory.

## CONCLUSION

- The algorithm presented in the paper has been implemented and its performance is tested with various metrics.
- The algorithm uses pessimistic or conservative approach opposed to lazy or optimistic approach which is generally the case.
- The size of OwnershipArray can be varied for application to application based on the storage and performance constraints.

## REFERENCES


- A simple deterministic algorithm for guaranteeing the forward progress of transactions by Charles E. Leiserson