Dynamic Timestamp Vs Travelling Timestamp

Project Report

cs14btech11008, cs15btech11024

Introduction:

- ➔ In Optimistic Concurrency Control (OCC) type of serialization algorithms, transactions are allowed to execute and make changes to local buffer until they reach their commit point and then they are validated (forward/backward) and changes are made transparent to other transactions.
- ➔ So, the execution of a transaction can be divided into three phases.
    1. read phase: read and write to local buffer.
    2. validation phase: validate if the operations performed by the transaction are consistent and following serialization principles
    3. write phase: write the changes to database.
- ➔ OCC protocols are non-blocking and deadlock freedom.
- ➔ If a transaction Ti is serialized before transaction Tj, the following two conditions must be satisfied: [2]
    a. No overwriting. The writes of Ti should not overwrite the writes of Tj.
    b. No read dependency. The writes of Ti should not affect the read phase of T
- ➔ T/O schemes assign a unique, monotonically-increasing timestamp to each transaction.

Problems with conventional OCC:

- ➔ Timestamps are assigned to the transactions statically, according to a fixed sequential order. This eases conflict detection, but limits the concurrency.
- ➔ Timestamp management is the key scalability bottleneck in the concurrency control algorithms.[1]

To handle these bottleneck issues, timestamps are proposed to be allocated dynamically to transactions(occ-da) as well as operations(tictoc).

TICTOC : Time Traveling Optimistic Concurrency Control[1]

- ➔ This paper introduces "*data-driven timestamp management*" : Instead of assigning timestamps to each transaction independently of the data it accesses, TicToc embeds the necessary timestamp information in each tuple to enable each transaction to compute a valid commit timestamp after it has run, right before it commits.[1]
- ➔ Since every transaction commutes it own timestamp, there is no need for centralized timestamp allocation.
- ➔ As timestamps are computed at commit time, this allows the transactions' operations to move forward in time and thereby allowing more serialization.

Protocol:

The dataitems in database are treated as list of tuples where each tuple is of form <value, rts, wts>.

Transactions are treated as a list of tuples where each tuple is of form <pointer to data in database,value, rts, wts>.

rts: read timestamp: timestamp after which reading this version is invalid.

wts: write timestamp: timestamp at which this version is written.

commit_ts: commit timestamp

- ➔ A data version has a valid between its write timestamp (wts) and the read timestamp (rts).
- ➔ A version read by a transaction is valid iff the transaction's commit_ts is in between the version's wts and rts which implies that the version being read before it expired which is valid.
- ➔ A write by a transaction is valid iff the transaction's commit timestamp is greater than the rts of the current version which implies that the transaction is not simply overwriting.
- ➔ For a transaction T to commit, the following should be ensured.

$$\exists\, commit\_ts,$$
$$(\forall v \in \{versions\ read\ by\ T\}, v.wts \leq commit\_ts \leq v.rts)$$
$$\wedge\ (\forall v \in \{versions\ written\ by\ T\}, v.rts < commit\_ts)$$

Algorithm:

1. Read phase : make a local copy of data.
2. Validation phase :
   For each tuple in the read set but not in the write set
   commit_ts should be no less than its wts
   For each tuple in the transaction's write set
   commit_ts needs to be no less than its current rts + 1
   For each tuple in the transaction's write set
   If tuple.rts < commit_ts
   abort
3. Write phase: update value, rts and wts of tuple in the database.

OCC_DA : Optimistic Concurrency Control - Dynamic Adjustment[2]

➔ This protocol identifies the various conflicts between the transaction in the validation phase (Tv) and other live transactions(Tj).

- ★ Read(Tv) - Write(Tj) conflict
  read of Tv should not be affected by write of Tj.
  Follow the serialization order Tv➜Tj, by forward adjusting of Tj.
- ★ Write(Tv) - Read(Tj) conflict
  Tv shouldn't overwrite a version that's being read by Tj.
  Follow the serialization order Tj➜Tv, by backward adjusting of Tj.
- ★ Write(Tv) - Write(Tj) conflict
  Basically a data-race
  Forward adjust (restart) the transaction basing on the priorities.

Protocol:

1. Initialization
   Each transaction maintains a Serialization Order Timestamp (SOT) for itself which is initialized to infinity(INF). If this value is not INF, it implies it's being backward adjusted by some other transaction that *was* validated.
   During read phase, set TR(Dp,Tv) to WTS of Dp, where Dp is data item.
   Validation
   ATS(Tv) - set of transactions with SOT <= Tv.SOT

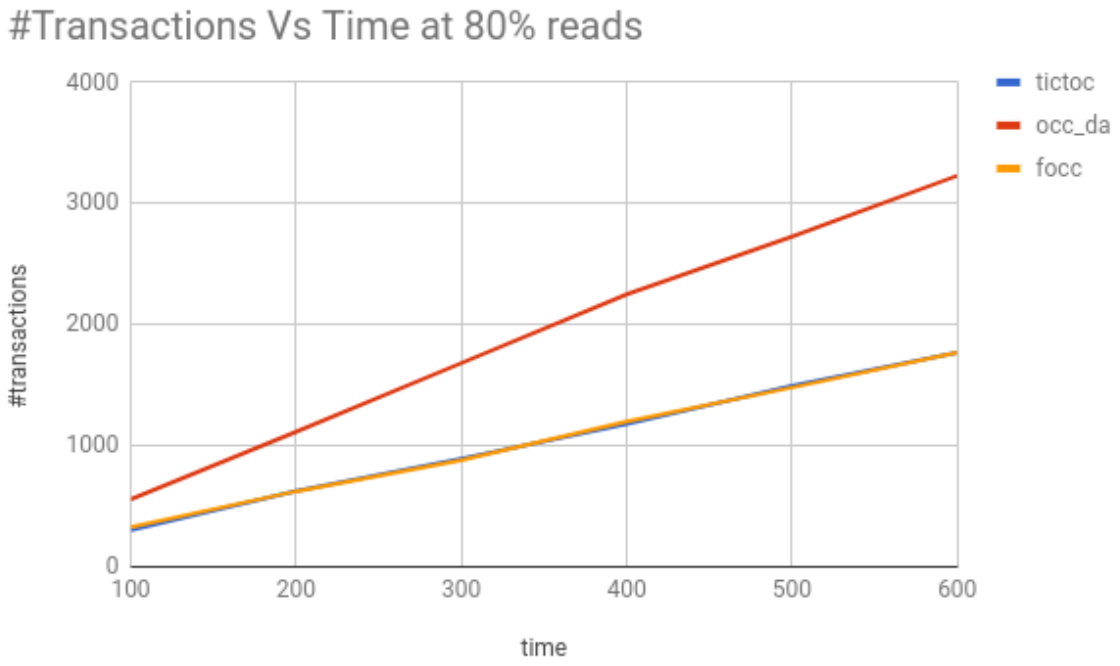
BTS(Tv) - set of transactions with SOT < Tv.SOT

2. Phase1
   If a transaction had been backward adjusted, write and read may not be valid. Upon checking, the transaction needs to restarted or simply aborted.
3. Phase2
   read-write conflicts between the other active transactions and the validating transaction respectively are identified. Conflicting transactions are being marked to adjust them backward.
4. Phase3
   The other conflicts (W-R, W-W) are being identified.
   Either the validating transaction or the conflicting active transaction needs to be restarted basing on the priorities. Due to way other parts of the scheduler is implemented, the current transaction is chosen to be aborted(restarted).
5. Phase4
   Commit the current transaction if it hasn't been aborted.
   Backward adjust the marked transactions from phase2.

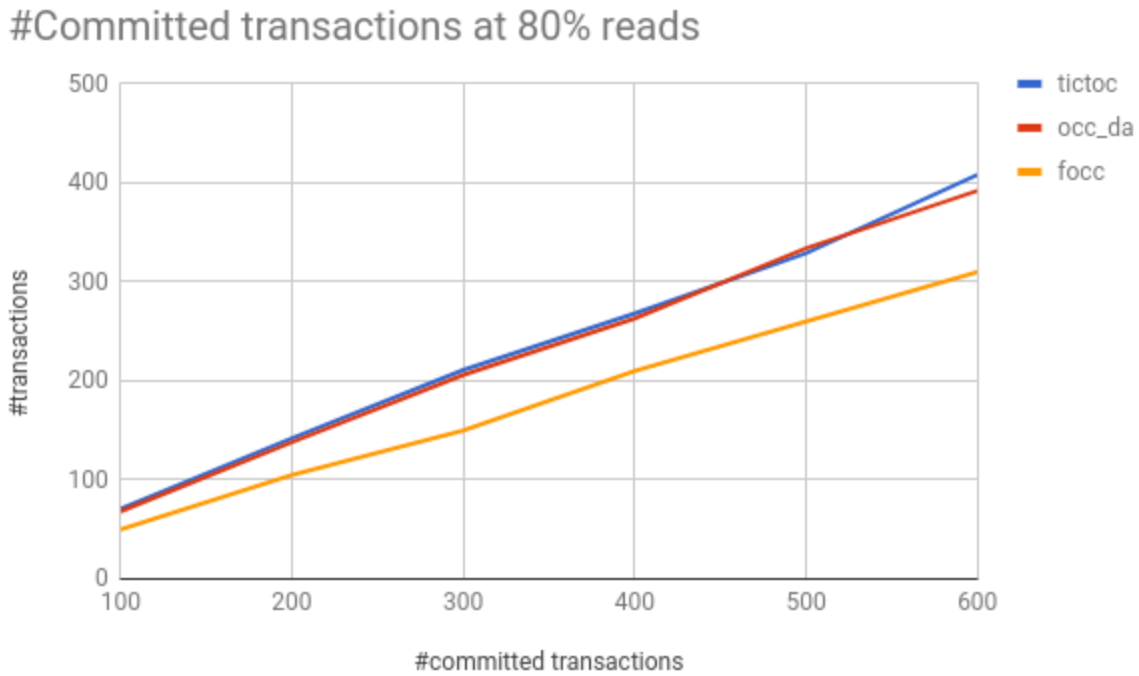
Observation:
Number of threads: 4 (as same as number of hardware threads)
Number of dataitems : 10

The following graphs compare tictoc, occ_da against focc as all of them do forward validation i.e., validation against live transactions.

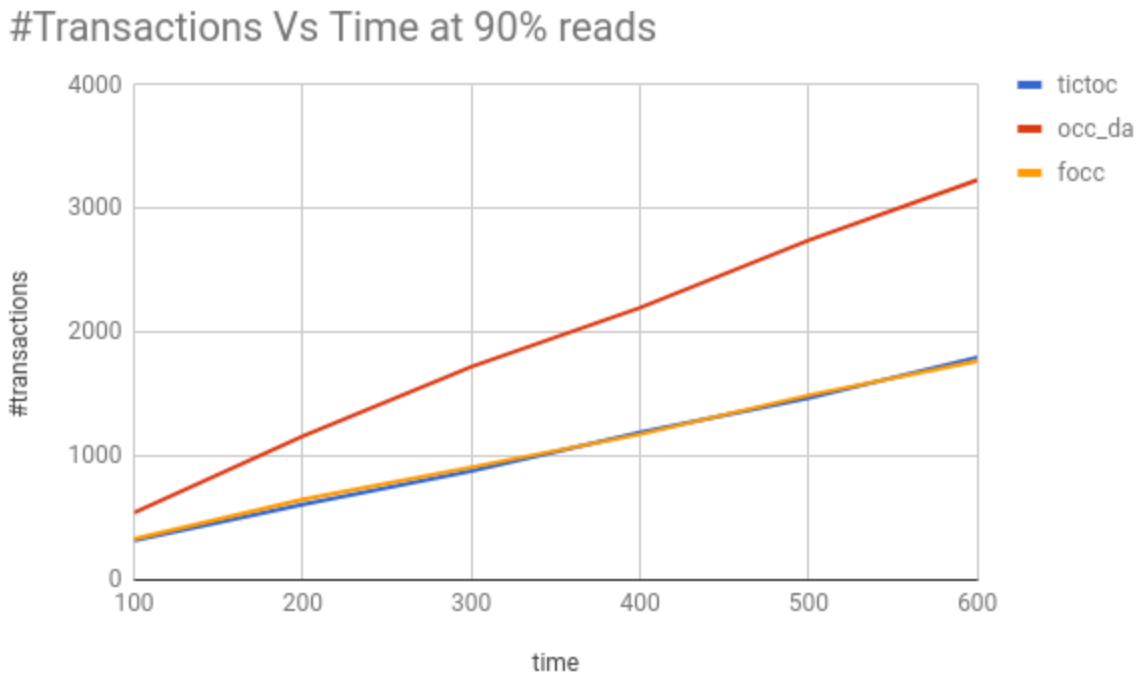
Graph showing the time taken by the schedulers when 80% of operations are reads:



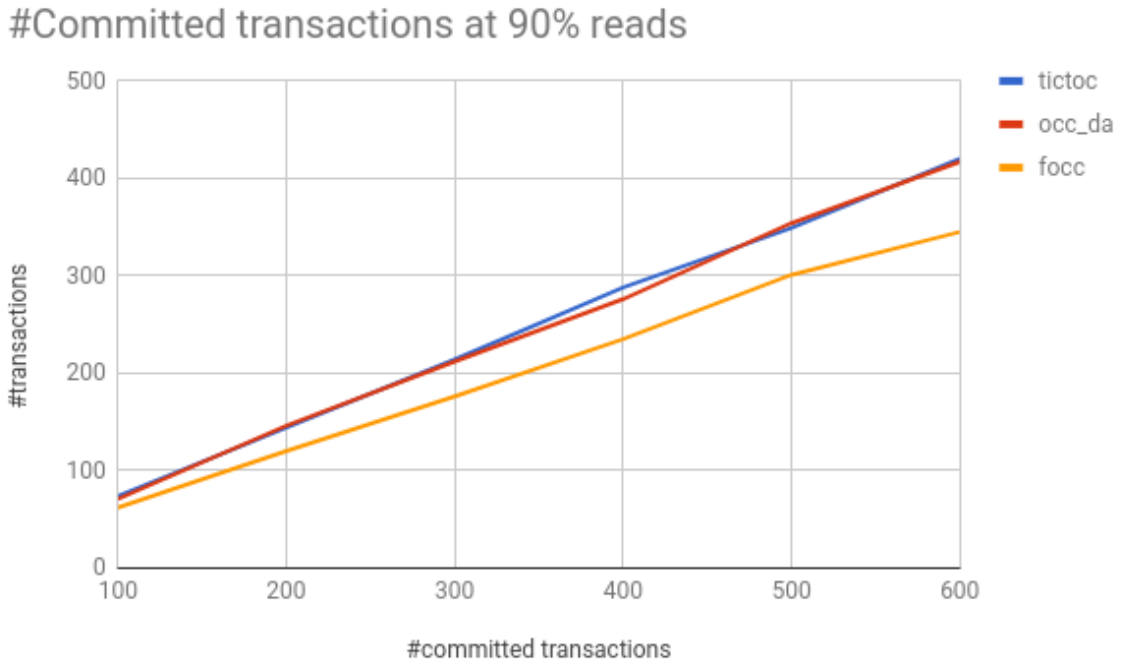
Graph showing the throughput of the schedulers when 80% of operations are reads:



Graph showing the time taken by the schedulers when 90% of operations are reads:



Graph showing the throughput of the schedulers when 90% of operations are reads:



Inferences:

- ➔ Algorithms that follow dynamic timestamp allocation (tictoc,occ_da) allow more transactions to commit than the ones that follow static timestamps (focc).
- ➔ Since these algorithms perform better in reads dominant environment, all of them gave better results at 90% reads than 80% reads.
- ➔ OCC_DA takes more time owing to the multiple checks it performs to detect conflicts (partly due to implementation).

References:

1. https://dl.acm.org/citation.cfm?id=2882935
2. Jan Lindstrom. OCC and Its Variants. Seminar on real-time systems.