TicToc: Time Traveling Optimistic Concurrency Control

Xiangyao Yu\textsuperscript{1}, Andrew Pavlo\textsuperscript{2}, Daniel Sanchez\textsuperscript{1}, Srinivas Devadas\textsuperscript{1}

Massachusetts Institute of Technology\textsuperscript{1}
Carnegie Mellon University\textsuperscript{2}
PHYSICAL VS LOGICAL TIME

Two-Phase Locking (2PL)

- Tuple A: Read Lock
- Tuple A: Write Lock
- Tuple B: Write Lock

Timestamp Ordering (T/O)

- Tuple A: Write Lock
- Tuple B: Write Lock

T1 @ ts = 1
T2 @ ts = 2
Timestamp Allocation
(txns have unique timestamps)

- Centralized Allocator
  - Timestamp allocation is a scalability bottleneck

  \[ ts = \text{sync\_fetch\_and\_add}(&\text{glob\_ts}, 1) \]

- Synchronized Clock
  - Clock skew causes unnecessary aborts

**Histogram:**
- Throughput (Million txn/s)
- Thread Count

**Lines:**
- T/O
- 2PL
Timestamp Allocation
(txns have unique timestamps)

Static Timestamp Assignment

- Txns abort due to bad timestamp assignment
DATA DRIVEN TIMESTAMP MANAGEMENT

Traditional T/O

1. Acquire timestamp (TS)
2. Determine tuple visibility using TS

- Timestamp Allocation
- Static Timestamp Assignment

TicToc

1. Access tuples and remember their timestamp info.
2. Compute commit timestamp (CommitTS)

- No Timestamp Allocation
- Dynamic Timestamp Assignment
OPTIMISTIC CONCURRENCY CONTROL (OCC)

Tuple Format

Data

wts (Write Timestamp)

rts (Read Timestamp)
READ PHASE

Write to local write set

Read data and timestamps atomically

Logical Time

1 2 3 4
VALIDATION PHASE

Step 1 Lock Write Set

TXN
WRITE(A)
READ(B)
READ(C)

Logical Time
VALIDATION PHASE

Step 1: Lock Write Set
Step 2: Compute CommitTS

TXN
- WRITE(A)
- READ(B)
- READ(C)

Logical Time
- CommitTS
- 1 2 3 4
VALIDATION PHASE

Step 1: Lock Write Set
Step 2: Compute CommitTS
Step 3: Validate Read Set

Case 1: latest version
VALIDATION PHASE

Step 1: Lock Write Set
Step 2: Compute CommitTS
Step 3: Validate Read Set

Case 1: latest version
Case 2: New version at/before CommitTS
VALIDATION PHASE

Case 1: latest version

Case 2: New version at/before CommitTS

Case 3: New version after CommitTS

Step 1: Lock Write Set

Step 2: Compute CommitTS

Step 3: Validate Read Set
TIME TRAVELING

Txn 1
BEGIN
READ(A)
COMMIT

Txn 2
BEGIN
WRITE(A)
COMMIT

Tuple A
Txn 1
CommitTS

Txn 2
CommitTS

Logical Time

Time
TICTOC OPTMIZATIONS

- No-wait locking in validation phase
- Preemptive aborts
- Timestamp history
- Lower isolation levels
EXPERIMENTAL SETUP

- 4-socket, 40-core Machine (80 threads w/ hyper-threading)
- Main Memory DBx1000
  - No logging
  - No B-tree (Hash indexing)
- Concurrency Control Algorithms
  - MVCC: HEKATON
  - OCC: SILO
  - 2PL: DL_DETECT, NO_WAIT
YSCB High Contention

Throughput

Abort Rate

Throughput (Milliontxn/s)

Abort Rate

Thread Count

Thread Count

TICTOC  HEKATON  DL_DETECT  NO_WAIT  SILO
Xiangyao Yu  
yxy@mit.edu

Andy Pavlo  
pavlo@cs.cmu.edu

Daniel Sanchez  
sanchez@mit.edu

Srinivas Devadas  
devadas@mit.edu

DBx1000 (https://github.com/yxymit/DBx1000)