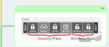
04-occ

- Stored Procedures
  - Disk stalls are (almost) gone when executing transactions in an in-memory DBMS. But there are still other stalls when an application uses **conversational** API to execute queries on DBMS (e.g., JDBC/ODBC, wire protocols)
    -

      维护所有的meta data，保持所有的锁，每待交易中下一个查询
  - Solutions to this problem
    - Prepared Statements
      - Removes query preparation overhead.
      - Provide the DBMS the SQL statement ahead of time and assign it to a name/handle. Can invoke that query just by using that name. This removes SQL parsing, binding, and planning (sometimes).
    - Query Batches
      - Invoke multiple queries per network message.
      - Reduces the number of network roundtrips.
    - Stored Procedures
      - A group of queries that form a logical unit and perform a particular task on behalf of an application directly inside of the DBMS. The application can then invoke the transaction as if it was an RPC.
      - Removes both preparation and network stalls.
  - Stored procedure例子
    -

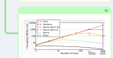
    -

  - Advantages of Stored Procedures
    - Reduce the number of round trips between application and database servers
    - Increased performance because queries are pre-compiled and stored in DBMS
    - Procedure reuse across applications
    - Transparent sever-side transaction restarts on conflicts
  - Disadvantages of Stored Procedures
    - Not as many developers know how to write stored procedure code
    - Outside the scope of the application so it is difficult to manage versions and hard to debug
    - Probably not to be portable to other DBMSs
    - DBAs usually do not give permissions out freely, so it makes it difficult for developers to constantly update their stored procedures.
- Concurrency Control
  - A DBMS' concurrency control protocol to allow transactions to access a database in a multi-programmed fashion while preserving the illusion that each of them is executing alone on a dedicated system. The goal is to have the effect of a group of transactions on the database's state is equivalent to any serial execution of all transactions.
  - Concurrency Control Schemes
    - Two-Phase Locking (Pessimistic)
      - Assume transactions will conflict so they must acquire locks on database objects before they are allowed to access them.
      - 

      - Deadlock
        -

        - Deadlock Detection
          - Each txn maintains a queue of the txns that hold the locks that it waiting for.
          - A separate thread checks these queues for deadlocks.
          - If deadlock found, use a heuristic to decide what txn to kill in order to break deadlock.
        - Deadlock Prevention
          - Check whether another txn already holds a lock when another txn requests it.
          - If lock is not available, the txn will either (1) wait, (2) commit suicide, or (3) kill the other txn.
    - Timestamp Ordering (Optimistic)
      - Assume that conflicts are rare so transactions do not need to first acquire locks on database objects and instead check for conflicts at commit time.
      - 存储：上一次交易读写的时间戳
- Timestamp Ordering Concurrency Control
  - Use timestamps to determine the order of transactions.
  - Basic T/O Protocol
    - Every transaction is assigned a unique timestamp when they arrive in the system.
    - The DBMS maintains separate timestamps in each tuple's header of the last transaction that read that tuple or wrote to it.
    - Each transaction check for conflicts on each read/write by comparing their timestamp with the times-tamp of the tuple they are accessing.
    - The DBMS needs copy a tuple into the transaction's private workspace when reading a tuple to ensure repeatable reads.
  - Optimistic Concurrency Control (OCC)
    - Store all changes in private workspace.
    - Check for conflicts at commit time and then merge.
    - First proposed in 1981 at CMU by H. T. Kung
      - Reference
        - H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. ACM Trans. Datab. Syst. 6(2), June 1981. URL: https://dl.acm.org/citation.cfm?id=319566.
    - Three Phases
      - Read Phase
        - Transaction's copy tuples accessed to private work space to ensure repeatable reads, and keep track of read/write sets.
      - Validation Phase
        - When the transaction invokes **COMMIT**, the DBMS checks if it conflicts with other transactions. Parallel validation means that each transaction must check the read/write set of other transactions that are trying to validate at the same time. Each transaction has to acquire locks for its write set records in some global order. Original OCC uses **serial validation**.
          - Backward Validation
            - Check whether the committing transaction intersects its read/write sets with those of any transactions that have **already** committed.
            -

          - Forward Validation
            - Check whether the committing transaction intersects its read/write sets with any active transactions that have **not** yet committed.
            -

        - Parallel validation means that each txn must check the read/write sets of other txns that are trying to validate at the same time.
          - Each txn has to acquire locks for its write set records in some **global order.**
          - The txn does not need locks for read set records.
      - Write Phase
        - The DBMS propagates the changes in the transactions write set to the database and makes them visible to other transactions' items. As each record is updated, the transaction releases the lock acquired during the Validation Phase
    - Timestamp Allocation
      - X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: an evaluation of concurency control with one thousand cores. In VLDB '14. Proceedings of the VLDB Endowment, volume 8, pages 209-220, November 2014. URL: https://dl.acm.org/citation.cfm?id=2735511.
      - Mutex
        - Worst option. Mutexes are the "Hitler of Concurrency".
      - Atomic Addition
        - Requires cache invalidation on write.
      - Batched Atomic Addition
        - Needs a back-off mechanism to prevent fast burn.
      - Hardware Clock
        - Intel新技术：call rdtscf并且read回一个clock（in sync across all cores）
        - Not sure if it will exist in future CPUs.
      - Hardware Counter
        - Treated as a special register，但是只在论文里提到过，并不存在
        - Not implemented in existing CPUs.
      - 

- Silo OCC
  - The Silo DBMS [1, 3] is an influential in-memory DBMS developed by Harvard and MIT
    - Silo - Multicore In-Memory Storage Engine. https://github.com/stephenehy/silo.
    - S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In SOSP '13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, November 2013. doi: https://dl.acm.org/citation.cfm?id=2522713.
    - It is a single- node OLTP system that uses Serializable OCC with parallel backward validation.
    - All transactions execute as stored procedures.
  - No writes to shared-memory for read txns
  - Batched timestamp allocation using **epochs**.
    - Time is sliced into fixed-length epochs (40ms).
    - All txns that start in the same epoch will be committed together at the end of the epoch.
      - Txns that span an epoch have to refresh themselves to be carried over into the next epoch.
      - 比如大于40ms的交易
    - Worker threads only need to synchronize at the beginning of each epoch.
  - 交易ID
    - Each worker thread generates a unique txn id based on the current epoch number and the next value in its assigned batch.
    -

  - Commit Protocol
    -

    - Step #1: Lock Write Set
      -

    - Step #2: Examine Read Set
      -

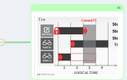
      -

    - Step #3: Install Write Set
      -

  - Garbage Collection
    - Cooperative threads GC.
    - Each worker thread marks a deleted object with a **reclamation epoch.**
      - This is the epoch after which no thread could access the object again, and thus can be safely removed.
      - Object references are maintained in thread local storage to avoid unnecessary data movement.
  - Range Queries
    - DBMS handles phantoms by tracking the txn's scan set (node set) on indexes.
      - Re-execute scans in the validation phase to see whether the index has changed.
      - Have to include "virtual" entries for keys that do not exist in the index.
        - 没有value，但可以阻止其他thread插入
    - ***key.range*** and ***index.gap*** locking
  - 表现
    -

- Tictoc
  - Serializable OCC implemented in DBx1000.
    - Parallel backward validation
    - Stored procedure-only API
  - No global timestamp allocation.
  - Txn timestamps are derived from records
  - Record Timestamps
    - A record is considered valid from W.TS to R.TS
    - Write Timestamp (W.TS)
      - The logical timestamp of the last committed txn that wrote to the record
    - Read Timestamp (R.TS)
      - The logical timestamp of the last txn that read the record.
  - Validation Phase
    - Step #1: Lock Write Set
      -

    - Step #2: Compute CommitTS
      -

    - Step #3: Validate Read Set
      - Case 1: Latest Version
        - 

      - Case 2: Updated Before CommitTS
        -

      - Case 3: Updated After CommitTS
        -

  - 表现
    -