12-logging

- Database Recovery
  - Database recovery algorithms are techniques to ensure database **consistency**, transaction **atomicity**, and **durability** despite failures.
  - Recovery Algorithms have two parts
    - 1. Actions during normal transaction processing to ensure that the DBMS can recover from a failure
    - 2. Actions after a failure to recover the database to a state that ensures atomicity, consistency, and dura- bility
  - Logging Schemes
    - Approach #1 - Physical Logging
      - Record the changes made to a specific record in the database.
      - Example: Store the original value and the after value for an attribute that is changed by a query.
        - 比如low level，如果改变很多tuples，那么对每一个tuple都要log
    - Approach #2 - Logical Logging
      - Record the high-level operations executed by transactions.
      - Example: The UPDATE, DELETE and INSERT queries invoked by a transaction.
        -
        - 在红的执行的时候又个context switch，导致第二个update执行
        - Log entry包含的信息不足以恢复原来的状态
      - 没有commercialized system用logical logging
        - Logical logging writes less data in each log record than physical logging. Difficult to implement in a system with logical logging if you have concurrent transactions. Its hard to determine which parts of the database may have been modified by a query before crash and it takes longer to recover because you have to re- execute every transaction over again.
- Disk-Oriented Logging and Recovery
  - The "gold standard" for physical logging and recovery in a disk-oriented DBMS is ARIES (Algorithms for Recovery and Isolation Exploiting Semantics)
    - ARIES was invented by IBM Research in the early 1990s for IBM DB2. It relies on STEAL and NO-FORCE buffer pool management policies.
    - Reference
      - C. Mohan, D. Haderle, B. Lindsey, H. Pirahesh, and P. Schwarz. Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write ahead logging. ACM TODS 17(1): 94-162, Sept. 1972 (TR-54), March 1992. URL: https://dl.acm.org/citation.cfm?id=128765.
  - Main Ideas
    - Write-Ahead Logging:
      - Any change is recorded in log on stable storage before the database change is written to disk.
    - Repeating History During Redo:
      - On restart, retrace actions and restore database to exact state before crash.
    - Logging Changes During Undo:
      - Record undo actions to log to ensure action is not repeated in the event of repeated failures.
  - Recovery Phases
    - Phase 1: Analysis
      - Read the WAL to identify dirty pages in the buffer pool and active transactions at the time of the crash.
    - Phase 2: Redo:
      - Repeat all actions starting from an appropriate point in the log.
      - Log redo operations in case of crash during recovery.
    - Phase 3: Undo:
      - Reverse the actions of transactions that did not commit before the crash.
  - Log Sequence Numbers
    - LSN
    - Each log record is assigned a unique identifier (**LSN**).
    - Every log record has a globally unique (**log sequence number (LSN)**) that is used to determine the serial order of those records.
    - The DBMS keeps track of various LSNs for both volatile and non-volatile storage to determine the order of almost **everything** in the system…
    -
    -
      -
    - OBSERVATION
      - Flush log记录是最大的overhead
      - Often the slowest part of the txn is waiting for the DBMS to flush the log records to disk.
      - 要等所有record都写入
      - Have to wait until the records are safely written before the DBMS can return the acknowledgement to the client.
  - Optimizations
    - Group Commit
      - Batch together log records from multiple transactions and flush them together with a single fsync.
        Amortizes the cost of I/O over several transactions.
      - Logs are flushed either after a timeout or when the buffer gets full.
      - Originally developed in **IBM IMS Fastpath** in the 1980s.
    - Early Lock Release:
      - 内部可以进行调换作（按顺）assume changes可提前commit，直到所有changes都完成才expose给外部。
      - A transaction's locks can be released before its commit record is written to disk as long as it does not return results to the client before becoming durable.
      - Other transactions can read data updated by a **pre-committed** transaction but cannot commit until its and also have to wait for their predecessor's log records to reach disk.
- In-Memory Database Recovery
  - Recovery in an in-memory system is slightly easier than in a disk-oriented system because the DBMS does not have to worry about tracking dirty pages in case of a crash during recovery
  - The DBMS also does not need to store undo records.
  - It is still stymied by the slow sync time of non-volatile storage.
    - This is why early papers (1980s) on recovery for in-memory DBMSs assume that there is non-volatile memory
      - Reference
        - T. J. Lehman and M. J. Carey. A recovery algorithm for a high-performance memory-resident database system. In SIGMOD '87 Proceedings of the 1987 ACM SIGMOD international conference on Manage- ment of data, pages 104-117, Dec 1987. URL: https://dl.acm.org/citation.cfm?id=38740.
    - This hardware is still not widely available so we want to use existing SSD/HDDs.
- Silo/ - Physical Logging
  - Silo/ is an extended version of the Silo DBMS that supports logging and checkpoints
    - W. Zheng, S. Tu, E. Kohler, and B. Liskov. Fast databases with fast durability and recovery through multicore parallelism. In OSDI'14 Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation, pages 465-477, Oct 2014. URL: https://dl.acm.org/citation.cfm?id=2685085
  - It uses the epoch based OCC that we discussed previously
    - 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. URL: https://dl.acm.org/citation.cfm?id=2522713.
    - It achieves high performance by parallelizing all aspects of logging, checkpointing, and recovery.
  - Logging Protocol
    - The DBMS assumes that there is one storage device per CPU socket. Each socket has one logger thread dedicated to writing data to that device
    - Worker threads are grouped per CPU socket. As the worker executes a transaction, it creates new log records that contain the values that were written to the database (i.e., REDO).
    - Each logger thread maintains a pool of log buffers that are given to its worker threads When a worker's buffer is full, it gives it back to the logger thread to flush to disk and attempts to acquire a new one
      - If there are no available buffers, then the worker thread stalls.
    - smooth out the process，分配10%内存给logging
  - Log Files
    - The logger threads write buffers out to file
      -
      - After 100 epochs, it creates a new file.
      - The old file is renamed with a marker indicating the max epoch of records that it contains.
    - Log Record Format
      - • Id of the transaction that modified the record (TID).
      - A set of value log triples (Table, Key, Value).
      - The value can be a list of attribute + value pairs.
  - Persistent Epoch
    - A special logger thread keeps track of the current persistent epoch (**epoch**)
      - Special log file that maintains the highest epoch that is durable across all loggers.
    - Txns that executed in epoch **e** can only release their results when the **epoch** is durable to non- volatile storage.
      - 和Groom模型入之图，其他交易不能release change
      -
  - Recovery Protocol
    - Phase 1: Load Last Checkpoint
      - Install the contents of the last checkpoint that was saved into the database.
      - All indexes have to be rebuilt.
        - 很多in-memory都rebuild index
        - 同时也会有一个更加compact的index（比如b+树）
    - Phase 2: Replay Log
      - Process logs in reverse order to reconcile the latest version of each tuple.
      - First Check the persist file to determine the most recent persistent epoch. Any log record from after the **epoch** is ignored.
      - Log files are then processed from newest to oldest:
        - Value logging is able to be replayed in any order.
        - For each log record, the thread checks to see whether the tuple already exists
          - If it does not, then it is created with the value.
          - If it does, then the tuple's value is overwritten only if the log TID is newer than the tuple's TID.
    -
  - 发现
    - The txn ids generated at runtime are enough to determine the serial order on recovery.
    - This is why Silo/ does not need to maintain separate log sequence numbers for each entry.
    - Node failures in OLTP databases are rare.
      - OLTP databases are not that big.
      - OLAP可能要比更charge的cluster整整四
      - They don't need to run on hundreds of machines.
    - It's better to optimize the system for runtime operations rather than failure cases.
  - 评估
    - Comparing Silo performance with and without logging and checkpoints
      - YCSB + TPC-C Benchmarks
        -
        -
        -
          -
    - Hardware
      - Four Intel Xeon E7-4830 CPUs (8 cores per socket)
      - 256 GB of DRAM
      - Three Fusion ioDrive2
      - RAID-5 Disk Array
- VoltDB - Logical Logging
  - VoltDB uses a variant of logical logging called command logging
    - Reference
      - N. Malviya, A. Weisberg, S. Madden, and M. Stonebraker. Rethinking main memory ohp recovery. In 2014 IEEE 30th International Conference on Data Engineering, pages 604-615, March 2014. doi: 10.1109/ICDE.2014.6816685.
    - COMMAND LOGGING
      - Logical logging scheme where the DBMS only records the stored procedure invocation
        - Stored Procedure Name
        - Input Parameters
        - Additional safety checks
      - Command Logging = Transaction Logging
      - DETERMINISTIC CONCURRENCY CONTROL
        - For a given state of the database, the execution of a serial schedule will always put the database in the same new state .
          - The order of txns (or their queries) is defined before they start executing.
          - The txn logic is deterministic
        -
          -
          -
            -
          -
            -
      - Problem
        - The key issue with command logging is that if the log contains multi-node transactions, then if one node goes down and there are no more replicas, then the entire DBMS has to restart.
        -
  - Logging Protocol
    -
      -
    -
      -
    - The DBMS logs the txn command (**before** it starts executing) once a txn has been assigned its serial order.
    - The node with the txn's "base partition" is responsible for writing the log record
      - Remote partitions do not log anything.
      - Replica nodes have to log just like their master.
  - Recovery Protocol
    - The DBMS loads in the last complete checkpoint from disk.
    - Nodes then re-execute all of the transactions in the log that arrived after the checkpoint started (oldest to newest).
      - The amount of time elapsed since the last checkpoint in the log (roughly) determines how long recov- ery will take.
      - Transactions that are aborted first still have to be executed unless there are txns in the log
  - REPLICATION
    - Executing a deterministic transaction on the multiple copies of the same database in the same order provides strongly consistent replicates.
      - DBMS does not need to use an atomic commit protocol (e.g., two-phase commit) to make sure that all of the nodes are in the same state.
      - DBMS does not need to use **Two-Phase Commit**
    -
      -
    -
      -