Main memory database recovery strategies

Arlino Magalhães
Angelo Brayner
José Maria Monteiro

Agenda

1. Introduction and motivation
2. Main memory databases overview
3. Database recovery
4. Main memory databases recovery
5. Main memory databases recovery strategies
6. Main challenges and future directions
Introduction and motivation

Let's start with the first set of slides!

Main Memory Database - MMDB (or In-Memory Database - IMDB)
▷ An efficient alternative to:
  ○ real-time view of operational data (OLTP-style)
  ○ high-performance mission-critical situations.
▷ Can provide:
  ○ very high transaction throughput rates, and
  ○ low latency.
▷ Data resides in main memory.
▷ Designed to optimize access to main memory.
Introduction and motivation

Factors that boosted the reinterest in MMDB research:

- Memory storage capacity and bandwidth:
  - Costs falling by a factor of 10 every five years.
Introduction and motivation

Factors that boosted the reinterest MMDB research

- Memory storage capacity and bandwidth:
  - costs falling by a factor of 10 every five years.
  - growing at a rate of 100% every three years.
Introduction and motivation

Factors that boosted the reinterest MMDB research

▷ Memory storage capacity and bandwidth:
  ○ growing at a rate of 100% every three years,
  ○ costs falling by a factor of 10 every five years.

▷ Recent hardware/architecture improvements:
  ○ NUMA architecture,
  ○ SIMD instructions,
  ○ RDMA networking,
  ○ hardware transactional memory, and
  ○ non-volatile memory.

Brief historical overview

1980s
▷ The researches have focused on:
  ○ improving the performance of disk-based databases, or
  ○ fitting the database in memory.
▷ Some products developed:
  ○ IMS/Fast Path, MARS MMDB, System M, TPK, OBE, and HALO.
▷ An infeasible solution due to high price and limited capacity of RAM.

1990s
▷ Advances in hardware/architecture technology re-generated interests in MMDB.
▷ Commercial systems targeting specialized workloads (e.g., telecom).
▷ Some systems: Dali/DataBlitz, ClustRa, TimesTen, and P*/Time.

Today?
Let's see now!
How to implement a main memory database?

Increase the main memory size of a disk-based database system to fit the entire database in the cache?

Main memory database
- Database resides in main memory.
- Improves access to main memory.
- Direct pointer to memory.
- Stores data in any format to achieve high throughput.
- Multi-versioning.
- MVCC, serial execution, embedded metadata.

Disk-resident database
- Database resides in secondary storage.
- Improves access to secondary memory.
- Implements buffer manager.
- Dependent on disk logical layout (e.g., page).
- In-place updating.
- Lock Based Concurrency Control.
**Hekaton vs. SQL Server**

Scalability experiment: Hekaton vs. SQL Server

<table>
<thead>
<tr>
<th># Number of cores</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td># SQL with contention</td>
<td>984</td>
<td>1,169</td>
<td>1,645</td>
<td>1,876</td>
<td>2,118</td>
<td>2,312</td>
</tr>
<tr>
<td># SQL without contention</td>
<td>515</td>
<td>2,157</td>
<td>3,161</td>
<td>4,311</td>
<td>5,933</td>
<td>6,834</td>
</tr>
<tr>
<td># Incomp</td>
<td>5,518</td>
<td>2,936</td>
<td>4,273</td>
<td>5,459</td>
<td>6,095</td>
<td>7,269</td>
</tr>
<tr>
<td># Nuke</td>
<td>7,978</td>
<td>13,849</td>
<td>20,912</td>
<td>24,721</td>
<td>32,527</td>
<td>38,857</td>
</tr>
</tbody>
</table>

Source: Diaconu, Cristian, et al., 2013

---

**MMDB problem:**

**Main memory volatility!**

MMDBs are more vulnerable to system failures.

E.g.: the database is lost in a power cut.
2. Main memory databases overview

MMDB basic concepts. New approaches!

Data storage -- Concurrency control -- Indexing -- Query processing
Durability and recovery -- Core technologies for MMDBs

Data storage

Disk-based database systems
- Buffer Manager: swaps pages from disk to memory, and vice versa.
- If page is in memory, it yet performs an indirect action to access a record:
  1. accessing the page in the main memory, and
  2. calculating the offset within the page to reach the record.
- Elegant solution to minimize disk access.
- Too much overhead for in-memory systems.

Source: Faerber, Franz, et al., 2017
Main memory database systems
- Data lives in main memory; no need to access page from disk.
- Avoid record indirection in buffer pool: (page id, offset).
- Common practice → pointers for direct access to the record in memory:
  - can save space for large values,
  - improves access in order of magnitude, and
  - requires fewer CPU cycles to access data.
- Experimental evaluation with IBM Starburst main-memory project (Lehman, et al., 1992) revealed that the buffer manager was responsible for about 40% of the query time execution.

Source: Faerber, Forz, et al., 2017

Harizopoulos, Stavros and Abadi, Daniel J. and Madden, Samuel and Stonebraker, M. OLTP Through the Looking Glass, and What We Found There. ACM SIGMOD, 2008
Data storage

Main memory database systems

- Organization Choices
  - Data partitioning
  - Multi-Versioning
  - Row/Columnar Layout

Data partitioning

- Disjoint partitioning of the database.
- The database can be partitioned on different machines.
- A single-thread performs a transaction serially in a partition:
  - exclusive access to data and resources (e.g., a CPU core).

Source: VoltDB Documentation. 2020, 2017
Data partitioning

- Disjoint partitioning of the database.
- The database can be partitioned on different machines.
- A single-thread performs a transaction serially in a partition
  - exclusive access to the data and resources (e.g., a CPU core).
- Balancing frequently accessed partitions.
- Some systems:
  - H-Store, VoltDB, and Calvin.

Multi-Versioning

- The updated and previous versions of data are written into different locations.
- Allows high concurrency degree
  - easier implementation of non-blocking concurrency control protocols.
- Enable snapshot generation.
- Needs a garbage collector.
- Some systems:
  - Hekaton, HyPer, and SAP HANA.
Row/Columnar Layout

N-ary Storage Model
▷ Row-oriented
▷ Attribute values for a single tuple are stored contiguously.
▷ OLTP workloads

<table>
<thead>
<tr>
<th>ID</th>
<th>IMAGE ID</th>
<th>NAME</th>
<th>PRICE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>201</td>
<td>ITEM-101</td>
<td>10</td>
<td>DATA-101</td>
</tr>
<tr>
<td>102</td>
<td>202</td>
<td>ITEM-102</td>
<td>20</td>
<td>DATA-102</td>
</tr>
<tr>
<td>103</td>
<td>203</td>
<td>ITEM-103</td>
<td>30</td>
<td>DATA-103</td>
</tr>
<tr>
<td>104</td>
<td>204</td>
<td>ITEM-104</td>
<td>40</td>
<td>DATA-104</td>
</tr>
</tbody>
</table>

Decomposition Storage Model
▷ Column-oriented
▷ Tuples' values for a single attribute are stored contiguously.
▷ OLAP workloads

<table>
<thead>
<tr>
<th>ID</th>
<th>IMAGE ID</th>
<th>NAME</th>
<th>PRICE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>201</td>
<td>ITEM-101</td>
<td>10</td>
<td>DATA-101</td>
</tr>
<tr>
<td>102</td>
<td>202</td>
<td>ITEM-102</td>
<td>20</td>
<td>DATA-102</td>
</tr>
<tr>
<td>103</td>
<td>203</td>
<td>ITEM-103</td>
<td>30</td>
<td>DATA-103</td>
</tr>
<tr>
<td>104</td>
<td>204</td>
<td>ITEM-104</td>
<td>40</td>
<td>DATA-104</td>
</tr>
</tbody>
</table>

Source: Arulraj, Joy, Andrew Pavlo, and Prashanth Menon, 2016

Row/Columnar Layout

Flexible Storage Model
▷ Generalizes the NSM and DSM models.
▷ Systems that need to transform the most up-to-date data into critical insights.
▷ HTAP workloads

<table>
<thead>
<tr>
<th>ID</th>
<th>IMAGE ID</th>
<th>NAME</th>
<th>PRICE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>201</td>
<td>ITEM-101</td>
<td>10</td>
<td>DATA-101</td>
</tr>
<tr>
<td>102</td>
<td>202</td>
<td>ITEM-102</td>
<td>20</td>
<td>DATA-102</td>
</tr>
<tr>
<td>103</td>
<td>203</td>
<td>ITEM-103</td>
<td>30</td>
<td>DATA-103</td>
</tr>
<tr>
<td>104</td>
<td>204</td>
<td>ITEM-104</td>
<td>40</td>
<td>DATA-104</td>
</tr>
</tbody>
</table>

Source: Arulraj, Joy, Andrew Pavlo, and Prashanth Menon, 2016
Concurrency control

- MMDBs avoid implementing a lock manager.
- Approaches
  - Pessimistic Concurrency Control (PCC)
    - E.g: embed metadata into records
      - “lock bit” (García-Molina, et al., 1992), or
      - integer counters (Ren, Kun, et al., 2012).
  - Optimistic Concurrency Control (OCC)
- Commonly used protocols
  - Multi-version Concurrency Control (MVCC)
  - Partitioned Serial Execution (PSE)

Multi-version concurrency control

- Reads do not wait for writes, and do not block writes.
- Overhead of:
  - creating new data versions, and
  - periodically removing obsolete versions.
- Optimistic MVCC: Hekaton and HyPer
  - transactions can perform unlocked until the commitment without context switch,
  - cheaper than handling locks,
  - allows scalability of cores, and
  - good approach when conflict rates are low.
- Pessimistic MVCC: SAP HANA
  - MVCC for read operations + record-level locks for write operations.
**Partitioned Serial Execution**

▷ Whenever a transaction T acquires a grant to access a partition P, only T has access to P.
  o There is no data contention during T’s execution on P
  o Very fast for single-partition transactions.

▷ Transactions can be performed in parallel on different partitions.

▷ Multi-partition transaction execution
  o can only start if all necessary partitions are available,
  o otherwise, it is aborted and restarted.
  o Some systems:
    o H-Store, VoltDB, and Calvin.

---

**Indexing**

**Disk-based clustered index**

**MMDB indexing approach**

**MMDBs indexing techniques**

▷ Cache efficiency
  o Height Optimized Trie (HOT)
  o Cache Sensitive Search Trees (CSS-Trees)
  o Cache Sensitive B*-Trees (CSB-Trees)
  o Prefetching B*-Trees (pB-Trees)
  o Fast Architecture Sensitive Tree (FAST)
  o Adaptive Radix Tree (ART)

▷ Multi-core parallelism
  o Multi-rooted B*-Tree (MRBTree)
  o Bw-tree ("Buzz Word Tree")
  o Mass-Tree

▷ Other techniques:
  o Optimistic Lock Coupling (OLC), T-Tree, ∨-Tree, and BD-Tree.
Query processing

- Avoids the iterator model (Volcano-style processing):
  - generates a huge number of function calls (e.g., open(), next(), and close()).
- MMDBs compile queries directly to machine code:
  - avoids interpretation and parsing overhead
  - can make better use of memory and CPU.

Core technologies for MMDBs

- The emergence of new technologies boosted interests in MMDBs
  - offer promising alternatives for MMDBs to reach their full potential.
- Hardware/architecture solutions:
  - non-uniform memory access (NUMA),
  - hardware transactional memory (HTM),
  - non-volatile memory (NVM),
  - single instruction multiple data (SIMD), and
  - remote direct memory access (RDMA).
NUMA architecture

- CPU speed grew faster than the main memory speed → data starvation.
- NUMA can address data starvation problem in modern CPUs.
  - Each processor can access:
    - local memory → minimal latency,
    - remote memory → longer latency.
  - Benefits:
    - improves memory bandwidth,
    - increases total memory size.

Non-volatile memory

- Promises the best properties from hard disks and DRAM:
  - byte addressability,
  - persistence with high performance, and
  - large storage capacity.
- Disadvantages:
  - limited endurance,
  - write/read asymmetry, and
  - uncertainty of ordering and atomicity.
- It is not clear how best to design a DBMS through NVM.
- Technologies:
  - PCM (Phase-Change Memory),
  - Memristors
  - STTMRAM (Spin-Transfer Torque Magnetic RAM)
SIMD instruction

- Operates multiple data objects in a single instruction.
- Efficient alternative to achieve data-level parallelism.
- Speeds up the processing free from concurrency issues, since each instruction consumes exclusively its own data.
- Disadvantages:
  - limits the maximum parallelism allowed, and
  - has constraints on data structures to operate
- MMDB design should consider data-level parallelism.
- Can speed up expensive database operations:
  - joins and sorts,
  - vector-style computation for Big Data analytics,
  - e.g. SAP HANA database speeds up dictionary decompression during scans by means of SIMD vector schema.

RDMA networking

- Enables two networked computers to exchange data in main memory without involving the kernel and CPU on the remote side.
  - Server does not coordinate a request, and
  - Clients can access the server’s memory directly.
- Zero CPU overhead compared to Ethernet.
- Disadvantages:
  - limits in synchronizing multiple accesses, and
  - inefficient coordination of access to remote memory, and
  - difficulty connecting directly to traditional Ethernet.
- Some applications:
  - FaRM implements lock-free reads over RDMA.
  - Hyper can generate backup files via RDMA.
3. Database Recovery

Second round! Basic concepts. Let's go ahead!

Introduction -- Features of crashes -- Recovery method
ARIES -- Log-structured recovery techniques

What is a transaction?

A transaction is a sequence of read/write operations on database.

Atomicity: all data changes are executed, or none.
Consistency: a transaction ensures database consistency.
Isolation: data changes of an active transaction are not visible to other transactions.
Durability: data changes of committed transactions should persist in the database.
Failure Types

▷ Transaction failure
  ○ Failure of an active transaction
  ○ Recovery action: UNDO.

▷ System failure
  ○ Main memory data lost.
  ○ Recovery action:
    ■ REDO of committed transactions
    ■ UNDO of uncommitted transactions.

▷ Media failure
  ○ Secondary memory failure.
  ○ Recovery action:
    ○ Database backup + Tape log files + Disk log file

A database crash scenario

Problems after a failure:
- Committed transaction may not have been flushed.
- Uncommitted transactions may have been flushed.

Recovery Manager procedures:
- The effects of T1, T2, and T3 might be redone.
- The effects of T4 and T5 might be undone.

This scenario is possible due to buffer replacement policies:
- No-force, and
- Steal.

Source: Haeder, Theo, and Andreas Reuter, 1983

E.g., hardware error, operating error, code error, and power cut.
Sequential log file

Log Record
- LSN: Log Sequence Number
- Type
- TransID: Transaction ID
- PrevLSN: Previous LSN
- PageID
- Log data

Physical record
- disk-resident databases
- faster during recovery process

Physiological record
- implemented by commercial DBMSs
- before image + redo operation

Logical record
- main memory databases
- faster during transaction processing

Problems:
- The more records, the longer the recovery time.
- Disk has limited space.

---

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current LSN</td>
<td>Current Log Sequence Number</td>
</tr>
<tr>
<td>Previous LSN</td>
<td>Previous Log Sequence Number</td>
</tr>
<tr>
<td>Operation</td>
<td>Describes the operation performed</td>
</tr>
<tr>
<td>Context</td>
<td>Context of the operation</td>
</tr>
<tr>
<td>Transaction ID</td>
<td>ID of the transaction in the LOG file</td>
</tr>
<tr>
<td>Log Record Length</td>
<td>Size of the row in bytes</td>
</tr>
<tr>
<td>Alter/TableName</td>
<td>Object name (table or view) against which the operation was performed</td>
</tr>
<tr>
<td>Page ID</td>
<td>Table/Index/Page ID</td>
</tr>
<tr>
<td>SPID</td>
<td>User Session ID</td>
</tr>
<tr>
<td>Start ID</td>
<td>User Transaction ID - logged only in the LOG.Begin.xact and LOG.Complete.xact records</td>
</tr>
<tr>
<td>Begin Time</td>
<td>Transaction Start Time - logged only in the LOG.Begin.xact record</td>
</tr>
<tr>
<td>End Time</td>
<td>Transaction End Time - logged only in the LOG.Complete.xact record</td>
</tr>
<tr>
<td>Transaction Name</td>
<td>Typically refers to the type of transaction: INSERT for example - logged only in the LOG.Begin.xact record</td>
</tr>
<tr>
<td>Transaction SID</td>
<td>User Security Identifier</td>
</tr>
<tr>
<td>Parent Transaction ID</td>
<td>If it is a child transaction, will contain the ID of its parent transaction</td>
</tr>
<tr>
<td>Transaction Begin</td>
<td>The first LSN of the transaction</td>
</tr>
<tr>
<td>Number of Locks</td>
<td>Number of locks</td>
</tr>
<tr>
<td>Lock Information</td>
<td>Description of the lock</td>
</tr>
<tr>
<td>Description</td>
<td>Transaction (UN)description</td>
</tr>
</tbody>
</table>
| Log Record     | The hexadecimal content of the transaction, an inserted/deleted row or the content of a page in
Operation types in SQL Server log record

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG_BEGIN_XACT</td>
<td>Begin Transaction</td>
</tr>
<tr>
<td>LOG_COMMIT_XACT</td>
<td>End Transaction</td>
</tr>
<tr>
<td>LOG_FORMAT_PAGE</td>
<td>Page Modified</td>
</tr>
<tr>
<td>LOG_INSERT_ROWS</td>
<td>Raw Inserted</td>
</tr>
<tr>
<td>LOG_DELETE_ROWS</td>
<td>Raw Deleted</td>
</tr>
<tr>
<td>LOG_LOCK_XACT</td>
<td>Lock</td>
</tr>
<tr>
<td>LOG_MODIFY_ROW</td>
<td>Raw Modified</td>
</tr>
<tr>
<td>LOG_MODIFY_COLUMNS</td>
<td>Column Modified</td>
</tr>
<tr>
<td>LOG_TRUNCATE_PAGE</td>
<td>Truncate Page</td>
</tr>
<tr>
<td>LOG.createUser</td>
<td>CreateUser</td>
</tr>
<tr>
<td>LOG_CREATE_INDEX</td>
<td>Create Index</td>
</tr>
<tr>
<td>LOG_TRUNCATE_INDEX</td>
<td>Truncate Index</td>
</tr>
<tr>
<td>LOG_END_COMMIT</td>
<td>Checkpoint End</td>
</tr>
<tr>
<td>LOG_MARK_SAVEPOINT</td>
<td>Checkpoint End</td>
</tr>
</tbody>
</table>

Logging techniques

**Write-Ahead Log (WAL) protocol**
- Writes log record to log file before flushing data pages to secondary memory

**Force-Log-At-Commit protocol**
- Writes log record to log during the execution of commit
Checkpoint

Mechanism to reduce redo operations in a recovery process
○ Identifies a star point for the redo phase.
▷ Main techniques:
○ Transaction Consistent Checkpoint (TCC),
○ Fuzzy Checkpoint.

A TCC scenario

- Advantages
  ○ Starts the recovery from the last checkpoint record.

- Disadvantages
  ○ Stops the transaction processing.
  ○ Degrades the system performance.

- Actions after crash:
  ○ redo T3 and T5,
  ○ roll back T4, and
  ○ nothing to do for T1 and T2.

Source: Haerder, Theo, and Andreas Reuter, 1983
A Fuzzy checkpoint scenario

Advantages
○ Does not stop transaction performing.
○ Does not degrade the system performance.

Disadvantages
○ Makes the recovery more expensive.

Actions after crash:
○ redo P1, P3, and P4,
○ redo active transactions, and
○ nothing to do for P2.

Undo/Redo algorithm

Phases:
1. Analysis
2. Undo
   ○ scans backward log, undoing the actions of loser transactions.
3. Redo
   ○ scans forward log, redoing the effects of committed transactions that did not flush.

Coarse-granularity
Blocking granularity ≥ Log granularity
ARIES algorithm

Supports:
▷ WAL protocol,
▷ steal and no-force buffer protocols,
▷ fuzzy checkpoint,
▷ partial rollbacks, and
▷ fine-granularity,
  ○ blocking granularity ≤ log granularity

Addition to earlier recovery methods:
▷ CLR (Compensation Log Record)

ARIES phases

1. Analysis identifies dirty pages, active transactions, and start of Redo.
2. Redo repeats transaction updates (repeating history paradigm).
3. Undo undoes updates of unfinished transactions.
CLR with partial rollback

Transaction updates

Log records

Undo actions

Variant ARIES algorithms

- **ARIES/NT** - ARIES for Nested Transactions
  - implements nested transactions,
  - uses backward chain tree.
- **ARIES-RRH** - ARIES with Restricted Repeating of History
  - handles the Redo phase more efficiently,
  - combines the repeating history paradigm with the selective redo paradigm,
  - can reduce the number of I/Os and the amount of CPU processing during the recovery.
- **ARIES/IM** - ARIES for Index Management
  - handles concurrency control and recovery with B-trees indexes,
  - ARIES/KVL was a previous work.
Variant ARIES algorithms

▷ ARIES/LHS - ARIES for Linear Hashing with Separators
  ○ handles the concurrency control and recovery with dynamic hash,
  ○ has not been implemented.
▷ ARIES/CSA - ARIES for the Client-Server Architecture
  ○ a client generates log records, and sends them to the server,
  ○ server:
    ○ stores the records in a single log file,
    ○ takes care of global locking, and
    ○ recovery.
  ○ has not been implemented.

Log-structured recovery techniques

▷ Log file structured as an index structure.
▷ Techniques:
  ○ Single-page repair,
  ○ Single-pass restore,
  ○ Instant restart, and
  ○ Instant restore.
Instant restore

A instant restore scenario

Source: Sauer, Caetano, 2017
4. Main memory databases recovery

The time has come! Highlighting techniques.

Introduction

- MMDB recovery activities: logging, checkpoint, and restart.
  - The only reason to access secondary memory.
  - The only way to recover an MMDB.
- Systems can keep database copies for higher availability.
  - High-availability infrastructures are not immune to failures.
  - High-availability infrastructures can lead to a significant costs.
- Recovery techniques are necessary to:
  - survive to failures, and
  - repair crashed databases as quickly as possible.
- MMDBs avoid ARIES-style recovery for performance reasons.
MMDB failures

▷ Transaction failure
   Failure of an active transaction.
   Recovery action:
   ○ in-place update storage → transaction rollback,
   ○ multi-versing storage → discard data versions.

▷ System failure
   ○ Symptoms similar to media failures in disk-resident databases
   ○ Database disappears.
   ○ Database applications stop running.
   ○ Recovery action:
     1. reload the last snapshot,
     2. replay logged actions,
     ○ Only redo operations → undo action is not necessary.

Loggin in MMDBs

▷ Log file stores transaction update records on secondary to support database recovery.
▷ Logging is negatively impacted by disk latency.
   ○ It increases transaction response time.
   ○ Transactions have to wait for disk write operations to commit.
   ○ It threatens MMDBs performance.
▷ MMBD logging tries to reduces the amount of write operations into disk:
   ○ Logical logging → fewer log data than physical logging.
   ○ Redo-only log (a local undo log only for transaction rollback).
   ○ It avoids logging indexes.
▷ SSD can be a good solution to store log files.
**Loggin in MMDBs**

- MMBD logging tries to reduce log traffic:
  - Command logging (or transaction logging)
    - It records transaction's logic rather than transaction's operations.
    - Each transaction must be a predefined stored procedure.
    - It only records the stored procedure identifier and its parameters.
    - It is very lightweight during transaction processing.
    - It can slow down the recovery process.
  - Group commit
    - It tries to flush records of multiple transactions in a single I/O.
    - It reduces the number I/Os.
  - Pre-commit
    - Transactions do not wait for log records to be written.
    - A transaction can escape from the log record write overhead.
    - However, a transaction can lose durability guarantees.

**Checkpoint in MMDBs**

- Log file tends to grow a lot:
  - the more records, the slower the recovery,
  - the disk has limited space,
  - e.g.: in OLTP environments, several log records update the same data item.
- Checkpoint approaches:
  - materializing logical log operations to an archive, and
  - Periodical database backup.
- Checkpoint reduces recovery time:
  - reduces the number of log records to be processed, and
  - recovering by loading physical data is faster than performing logical records.
- MMDB checkpoint avoids Fuzzy checkpoint.
- MMDB checkpoint usually performs an Copy-on-Update (COU) technique.
Checkpoint in MMDBs

**Copy-on-Update (COU)** snapshot algorithm
- COU is performed at system runtime.
- COU algorithm:
  - copies all database tuples to snapshot since checkpoint beginning;
  - skips inserted tuples;
  - copies updated and deleted tuples to shadow table;
  - copies all tuples from shadow table to snapshot;
  - a thread serializes the snapshot from memory to disk.
- COU can generate a memory usage overhead:
  - It may potentially need twice the database memory size.
- Other proposed snapshot algorithms:
  - Naive, Zigzag, PingPong, Hourglass, and Piggyback.

Restart (recovery) in MMDBs

**Default MMDB recovery**

- Ordered reload algorithm:
  - reloads the data in the same order in which they were written physically;
  - can provide the fastest database reload, and
  - must reload the entire database before replaing log actions or starting up.
- Some MMDBs try to parallelize recovery:
  - Hekaton → multiple log devices,
  - PACMAN → dependency graph, and
  - RAMCloud → log-structured approach.
- Trade-off:
  - recovery performance Vs. transaction processing performance.

Source: Araújo, Arlino, 2022
5. Main memory databases recovery strategies

A representative sample of MMDB recovery strategies.

Hekaton -- VoltDB -- HyPer -- SAP HANA -- SiloR -- TimesTen
PACMAN -- Adaptive Logging -- FineLine -- HiEngine -- MM-Direct

Some modern MMDBs and its main recovery features

<table>
<thead>
<tr>
<th>MMDB</th>
<th>Concurrence Control</th>
<th>Logging</th>
<th>Checkpoint</th>
<th>Instant Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hekaton</td>
<td>optimistic MVCC</td>
<td>operation logging</td>
<td>delta and data files</td>
<td>no</td>
</tr>
<tr>
<td>VoltDB</td>
<td>serial execution</td>
<td>transaction logging</td>
<td>snapshot</td>
<td>no</td>
</tr>
<tr>
<td>HyPer</td>
<td>serial execution</td>
<td>transaction logging</td>
<td>snapshot</td>
<td>no</td>
</tr>
<tr>
<td>SAP HANA</td>
<td>MVCC, 2PC</td>
<td>transaction logging</td>
<td>snapshot</td>
<td>no</td>
</tr>
<tr>
<td>SiloR</td>
<td>optimistic MVCC</td>
<td>value logging</td>
<td>&quot;fuzzy&quot; snapshot</td>
<td>no</td>
</tr>
<tr>
<td>TimesTen</td>
<td>MVCC, 2PL</td>
<td>value logging</td>
<td>&quot;fuzzy&quot; checkpoint, snapshot</td>
<td>no</td>
</tr>
<tr>
<td>PACMAN</td>
<td>serial execution</td>
<td>transaction logging</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>Adaptive Logging</td>
<td>serial execution</td>
<td>transaction and ARIES logging</td>
<td>snapshot</td>
<td>no</td>
</tr>
<tr>
<td>FineLine</td>
<td>-</td>
<td>physiological logging</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>HiEngine</td>
<td></td>
<td>physical logging</td>
<td>-</td>
<td>yes</td>
</tr>
<tr>
<td>MM-Direct</td>
<td>-</td>
<td>operation logging</td>
<td>tuple-level consistent checkpoint</td>
<td>yes</td>
</tr>
</tbody>
</table>
Hekaton

▷ It is an in-memory engine of Microsoft SQL Server.
▷ SQL Server database tables
  ○ disk-based table (regular table, and
  ○ in-memory table (Hekaton table).
▷ Both in-memory and regular tables are handled by T-SQL.
▷ A transaction can update both in-memory and disk-based tables.
▷ A stored procedure that accesses only in-memory tables is compiled in the machine code.
▷ It uses an optimistic multi-version concurrency control and lock-free (latch-free).
▷ It uses logs and checkpoints for durability and recovery purposes.

Hekaton

▷ Logging
  ○ logical,
  ○ read-only,
  ○ log records
  ○ group commit.
▷ Checkpoint
  ○ two type of files
    ○ data file = inserted and updated versions
    ○ delta file = IDs of deleted versions
  ○ append-only,
  ○ read-only.

They are generated at the commit time!

○ versions created, and
○ keys of deleted versions.

○ data files can degrade recovery when the number of delta files increases,
  ○ adjacent data files must be merged.
Hekaton

- Recovery
  1. loads the last checkpoint file
     - current version are loaded by data/delta files pairs,
       - a delta file filters versions in a data file
     - data/delta units can be loaded in parallel,
       - different cores can load pair in parallel.
  2. replays log actions from checkpoint record.

VoltDB

Volt Active Data

- It was designed from H-Store database (academic version).
- It is partitioned system
  - distributes data across compute nodes (or cores) in shared-nothing configuration,
  - serial execution at partitions: avoids concurrency control.
- A transaction must be a single stored procedure.
- Implements command logging and asynchronous transaction-consistent checkpoint.
VoltDB

Logging
- Single-partition transaction
  - writes records to the log file of its site.
- Distributed transaction
  - only the coordinator site stores log records,
  - messages exchanged are also logged,
  - coordinator replicas also record log records.

Checkpoint
- periodically snapshots are written to devices.

VoltDB

Recovery
1. loads the last checkpoint file to memory,
2. replays log actions from checkpoint record,
   1. a thread copies the log content of each node into memory,
   2. node's initiator processes the log entries,
   3. node's initiator dispatches transactions to the appropriate sites.
   This recovery schema can work even if the site topology changes.
   - If a site is removed, the initiator can send the transactions to a new site.
HyPer
Hybrid OLTP&OLAP High Performance
▷ It can handle both OLTP and OLAP workloads on the same database.
▷ OLAP queries can run on the most recent database state.
▷ It can generate a memory overhead from OLAP sessions.
▷ OLAP queries can access secondary storage when memory is low.

Logging
○ employs redo-only command logging.
○ maintains an undo log in memory.

Checkpoint
○ virtual memory snapshots can be used to create database backups.

Recovery
○ loads the last checkpoint and replays log actions.

Source: Funke, Florian, et al., 2014
SAP HANA

High-performance ANalytic Appliance

▷ It can has multiple data processing engines:
  o the classical relational processing (both transactional and analytical),
  o text processing, and
  o graph processing.

▷ It implements a unified table structure:
  o optimized storage for OLTP transactions, and
  o highly compressed structure for OLAP queries.

 o Three representations for tuples in a table:
   o L1-delta,
   o L2-delta,

Source: Sikka, Vishal, et al., 2012

L1-delta
  o row logical format,
  o fast insertion, update, deletion, and record projection.

L2-delta
  o column-store format (intermediate),
  o unsorted dictionary encoding,
  o bulk load operations.

Main store
  o columnar store format,
  o sorted dictionary highest comprised
SAP HANA

- **Logging**
  - logs redo logical operations from L1-delta and L2-delta transactions,

- **Checkpoint**
  - savepoint of L2-delta and main store.

- **Recovery**
  - loads the last savepoint and replays log actions.

PACMAN

- It is a recovery mechanism that tries to:
  - parallelize the recovery, and
  - minimizes the runtime overhead for transaction processing.
- It was designed based on two prerequisites:
  1. the DBMS must utilize command logging,
  2. the DBMS must replay re-executing transactions to recover the database.
- It was implemented in Peloton database.
- It parallels the log replay by two analysis process:
  1. static, and
  2. dynamic.
PACMAN

» Static Analysis
  1. Local dependency graph: identifies opportunities for parallel execution in each store procedures flow dependency, and data dependency.
  2. Global dependency graph: has the constraints and possible paralleling of execution among the pieces of all local dependency graphs.

FineLine

» It uses a structured log (partitioned B-tree) as the only way of persistence.
» It implements the same recovery strategy as Instant Restore.
  - Recovers tuples incrementally and on-demand.
  - It implements only a log file (physical and read-only).
  - After a system failure, the system performs an instant recovery schema.
» Checkpoints are not implemented.
HiEngine

▷ It is an MMDB developed by Huawei.
▷ It uses a sequential log whose records contain:
  ○ type (e.g., insert/update/delete),
  ○ tuple ID, and
  ○ tuple version.
▷ It access tuples by an Adaptive Radix Tree.
▷ It uses an indirection array to map tuple IDs to:
  ○ tuple address, or
  ○ log offsets of the latest tuple version.
▷ During recovery, a tuple version will be loaded into memory on demand.
▷ It needs fast and frequent checkpoints, as its instant recovery technique is to quickly reconstruct its main index structure.

Wait-free index checkpoint techniques:

▷ ChainIndex
  ○ “frozens” the tree and regards it as an incremental snapshot.
▷ MirrorIndex
  ○ an extra mirror tree for index reads.
▷ IACoW
  ○ copy-on-write for tree structures to generate snapshots.
MM-Direct

- **MM-DIRECT** - Main Memory Database Instant Recovery with Tuple Consistent Checkpoint.
- Implemented in Redis database.
- The main idea is to organize the log as an index structure (B+-tree).
- The system recovers the database incrementally.
- Transactions can access tuples as soon as they are restored.
- The system can restore tuples on-demand.

Sequential log (a) and Indexed log (b)

- Tuple Consistent Checkpoint - TuCC
- Tuple Consistent Checkpoint for MFU tuples - TuCC-MFU
There is no downtime in Instant Recovery!

Transactions are scheduled during recovery.

Transaction throughput similar.

Immediate Recovery approach

Default Recovery approach

Recovery time

Downtime

6. Main challenges and future directions

The final level!
**MMDB Recovery**

**Future directions**
- Novel log structures.
- Asynchronous logging processes.
- New technologies.

**Challenges**
- MMDB life-time
  - New technologies
  - SSD with I/O bandwidth similar to DRAM

**References**

References