48th ACM SIGMOD/PODS International Conference on Management of Data

Main memory database recovery strategies

Arlino Magalhães
Angelo Brayner
José Maria Monteiro

1

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

3

3

1.
Introduction and motivation

Let's start with the first set of slides!

4

Introduction and motivation

Main Memory Database - MMDB (or In-Memory Database - IMDB)

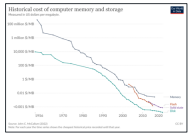
- An efficient alternative to:
  - real-time view of operational data (OLTP-style)
  - high-performance mission-critical situations.
- Can provide:
  - very high transaction throughput rates, and
  - low latency.
- Data resides in main memory.
- Designed to optimize access to main memory.

5

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.



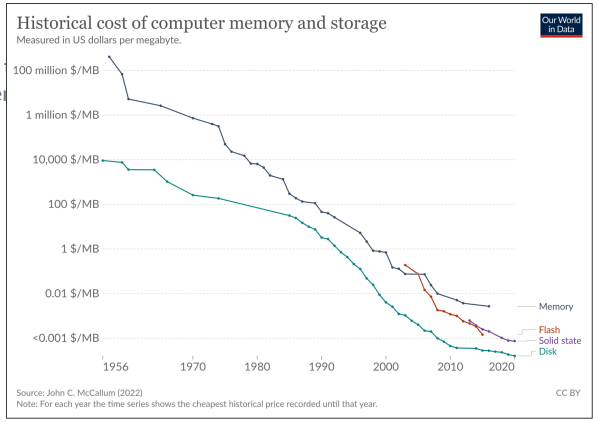
6

6

Introduction and motivation

Factos

- Me
  -



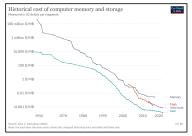
7

7

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,



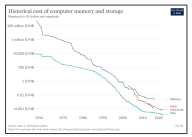
8

8

Introduction and motivation

Factors that boosted th

- Memory storage ca
  - growing at a ra
  - costs falling by



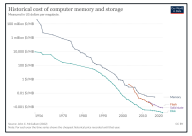
9

9

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.



10

10

Brief historical overview

1980s	1990s	Today?
▷ 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.	▷ 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.	Let's see now!

11

11

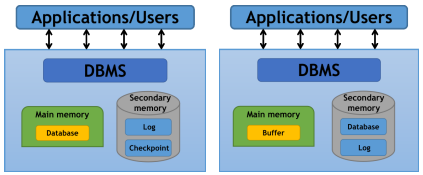
How to implement a main memory database?

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

12

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

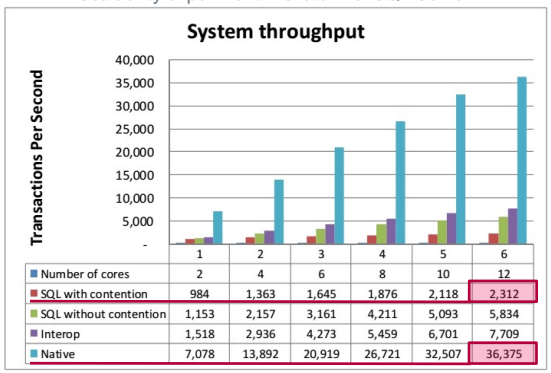
- Database resides in secondary storage.
- Improves access to secondary memory.
- Implements buffer manager.
- Dependent on disk logical layout (e.g. page).
- In-place updating.
- Lock Based Concurrency Control.

13

13

Hekaton vs. SQL Server

Scalability experiment: Hekaton vs. SQL Server



	1	2	3	4	5	6
Number of cores	2	4	6	8	10	12
SQL with contention	984	1,363	1,645	1,876	2,118	2,312
SQL without contention	1,153	2,157	3,161	4,211	5,093	5,834
Interop	1,518	2,936	4,273	5,459	6,701	7,709
Native	7,078	13,892	20,919	26,721	32,507	36,375

Source:
Diaconu, Cristian, et al., 2013

14

14

MMDB problem:

Main memory valatility!

MMDBs are more vulnerable to system failures.

E.g.: the database is lost in a power cut.

15

15

2.
Main memory databases overview

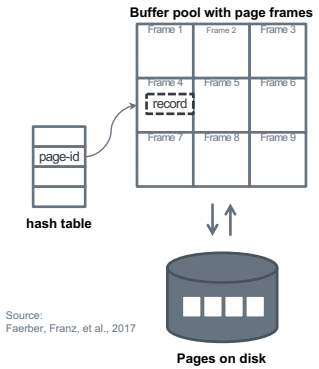
MMDB basic concepts. New approaches!

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

16

16

Data storage



Source:
Faerber, Franz, et al., 2017

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.

17

17

Data storage

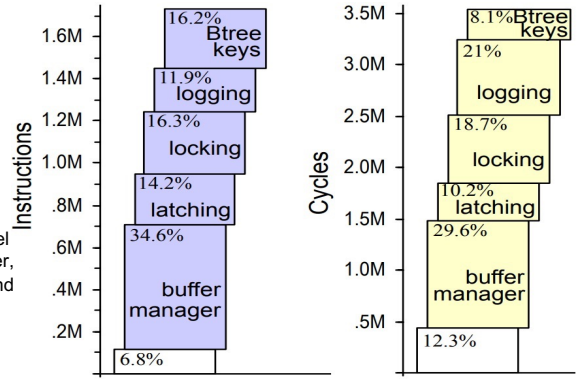
Main memory database systems

- 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) reveled thT buffer manager was responsible for about 40% of the query time execution.

Source: Faerber, Franz, et al., 2017

Data storage

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

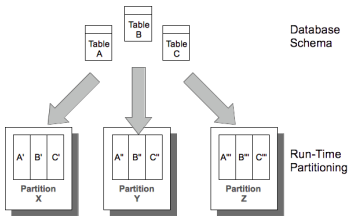
- Organization Choices
  - Data partitioning
  - Multi-Versioning
  - Row/Columnar Layout

20

20

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).



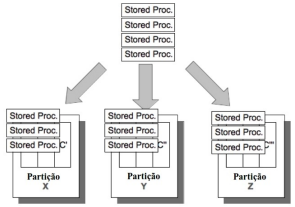
Source: VoltDB Documentation. 2020, 2017

21

21

Data partitioning

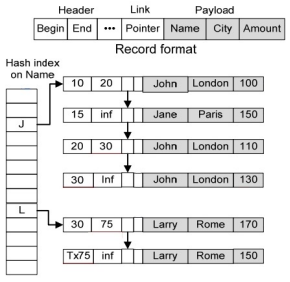
- Disjoint partitioning of the database.
- The database can be partitioned on different machines.
- A single-thread performs a transaction serially in a partition
  - exclusive access to the data and resources (e.g., a CPU core).
- Balancing frequently accessed partitions.
- Some systems:
  - H-Store,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



Source: VoltDB Documentation. 2020, 2017

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.



Source: Diaconu, Cristian, et al., 2017

Row/Columnar Layout

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

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

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

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

24

24

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

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

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

25

25

Concurrency control

- MMDBs avoid implementing a lock manager.
- Aproaches
  - Pessimistic Concurrency Control (PCC)
    - E.g: embed metadata into records
      - "lock bit" (Garcia-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)

26

26

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.

27

27

Partitioned Serial Execution

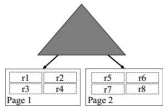
- Whenever a transaction T acquires a grant to access a partition P, only T has access to P.
  - There is no data contention during T's execution on P
  - Very fast for single-partition transactions.
- Transactions can be performed in parallel on different partitions.
- Multi-partition transaction execution
  - can only start if all necessary partitions are available,
  - otherwise, it is aborted and restarted.
- Some systems:
  - H-Store, VoltDB, and Calvin.

28

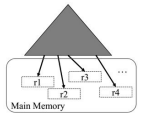
28

Indexing

Disk-based clustered index



MMDB indexing approach



Source: Faerber, Franz, et al., 2016

MMDBs indexing techniques

- Cache efficiency
  - Height Optimized Trie (HOT)
  - Cache Sensitive Search Trees (CSS-Trees)
  - Cache Sensitive B$^+$-Trees (CSB-Trees)
  - Prefetching B$^+$-Trees (pB-Trees)
  - Fast Architecture Sensitive Tree (FAST)
  - Adaptive Radix Tree (ART)
- Multi-core parallelism
  - Multi-rooted B$^+$-Tree (MRBTree)
  - Bw-tree ("Buzz Word Tree")
  - Mass-Tree
- Other techniques:
  - Optimistic Lock Coupling (OLC), T-Tree, Δ-Tree, and BD-Tree.

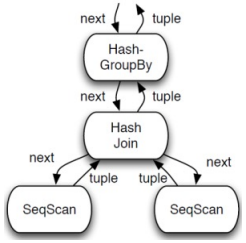
29

29

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.

A tree of physical operators and Volcano-style iteration



Source: Choi, Hyunsik, 2016

30

30

Core technologies for MMDBs

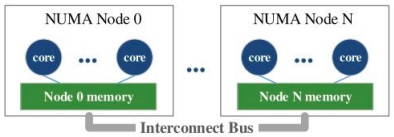
- The emergence of new technologies boosted interests in MMDBs
  - offer promising alternatives for MMDBs to reach their full potential.
- Hardware/architecture solutions:
  - non-uniform memory access (NUMA),
  - hardware transactional memory (HTM),
  - non-volatile memory (NVM),
  - single instruction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

31

31

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.



Source: Zhang, Hao, et al., 2015

32

32

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)

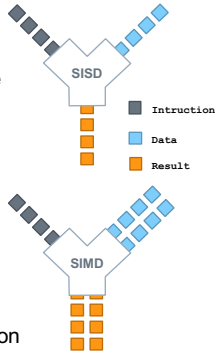
Can remove propagation costs of disk and memory-oriented systems.

33

33

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.



34

34

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,
  - 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.

35

35

3.
Database Recovery

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

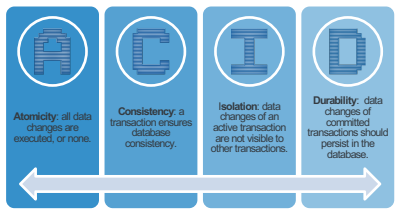
Introduction -- Features of crashes -- Recovery method
ARIES -- Log-structured recovery techniques

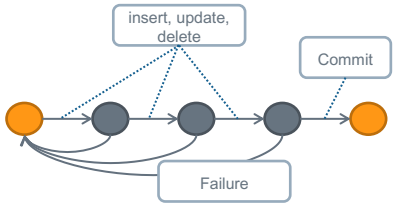
36

36

What is a transaction?

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





37

37

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

38

38

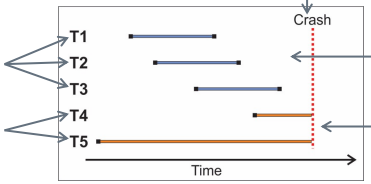
A database crash scenario

E.g., hardware error, operating error, code error, and power cut.

Problens after a failure:

Committed transction may not have been flushed.

Uncommitted transctions 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.

Source: Haerder, Theo, and Andreas Reuter, 1983

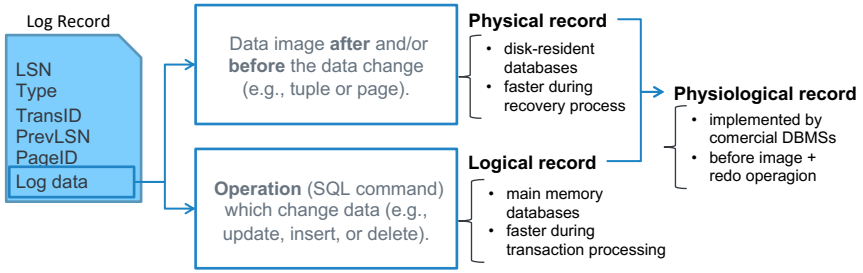
This scenario is possible due to buffer replacement policies:

- No-force, and
- Steal.

39

39

Sequential log file



Problems:

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

40

40

Column	Description
Current LSN	Current Log Sequence Number
Previous LSN	Previous Log Sequence Number
Operation	Describes the operations performed
Context	Context of the operation
Transaction ID	ID of the transaction in the LOG file
Log Record Length	Size of the row in bytes
AllocUnitName	Object name (table or index) against which the operation was performed
Page ID	Table/Index Page ID
SPID	User Session ID
Xact ID	User Transaction ID – logged only in the LOP_BEGIN_XACT and LOP_COMMIT_XACT rows
Begin Time	Transaction Start time - logged only in the LOP_BEGIN_XACT record
End Time	End Time - logged only in the LOP_COMMIT_XACT record
Transaction Name	Typically refers to the type of transaction: INSERT for example - logged only in the LOP_BEGIN_XACT record
Transaction SID	User Security identifier
Parent Transaction ID	If is a child transaction, will contain the ID of its parent transaction
Transaction Begin	The first LSN of the transaction
Number of Locks	Number of locks
Lock Information	Description of the lock
Description	Transaction LOG row description
Log Record	The hexadecimal content of the transaction, an inserted/deleted row or the content of a page i.e.

41

41

Operation types in SQL Server log record

Operation	Description
LOP_BEGIN_XACT	Begin Transaction
LOP_COMMIT_XACT	End Transaction
LOP_FORMAT_PAGE	Page Modified
LOP_INSERT_ROWS	Row inserted
LOP_DELETE_ROWS	Row deleted
LOP_LOCK_XACT	Lock
LOP_MODIFY_ROW	Row Modified
LOP_MODIFY_COLUMNS	Column Modified
LOP_XACT_CKPT	Checkpoint
LOP_BEGIN_CKPT	Checkpoint start
LOP_END_CKPT	Checkpoint end
LOP_MARK_SAVEPOINT	Save point

42

42

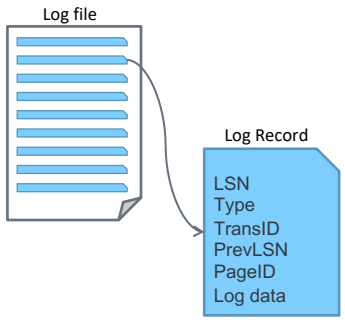
Logging techniques

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



43

43

Checkpoint

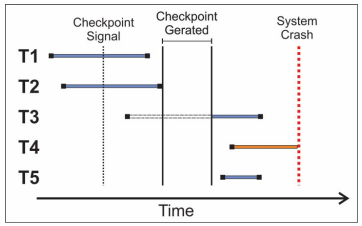
Mechanism to reduce redo operations in a recovery process
- Identifies a star point for the redo phase.

- Main techniques:
  - Transaction Consistent Checkpoint (TCC),
  - Fuzzy Checkpoint.

44

44

A TCC scenario



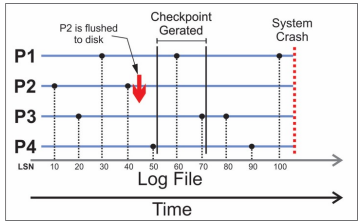
Source: Haerder, Theo, and Andreas Reuter, 1983

- 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.

45

45

A Fuzzy checkpoint scenario



- Advantages
  - Does not stop transaction performing.
  - Does not degrade the system preformance.
- Disadvantages
  - Makes the recovery more expensive.
- Actions after crash:
  - redo P1, P3, and P4,
  - redo active transactions, and
  - nothing to do for P2.

47

47

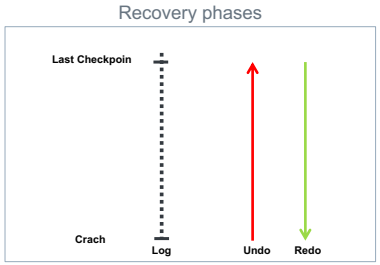
Undo/Redo algorithm

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



48

48

ARIES algorithm

Algorithms for Recovery and Isolation Exploiting Semantics

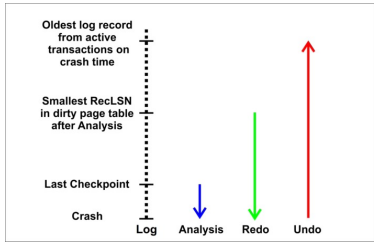
Supports:

- 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

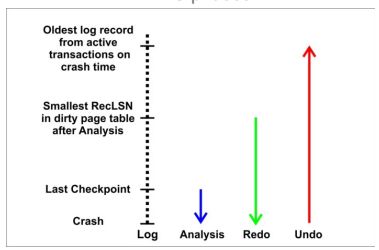


Source: Ramakrishnan, Raghu, Johannes Gehrke, and Johannes Gehrke, 2003

ARIES algorithm

Algorithms for Recovery and Isolation Exploiting Semantics

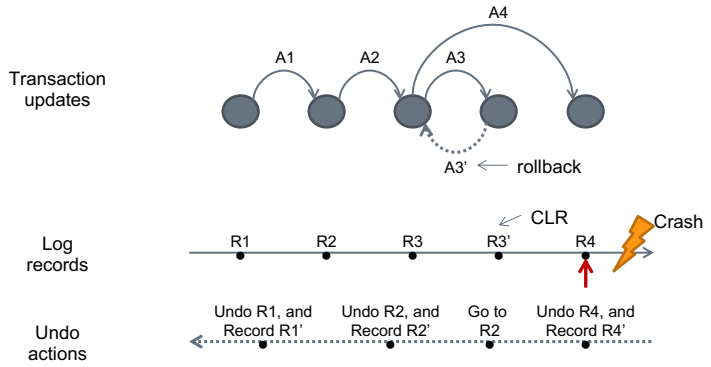
ARIES phases



Source: Ramakrishnan, Raghu, Johannes Gehrke, and Johannes Gehrke, 2003

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



Source: Haerder, Theo, and Andreas Reuter, 1983

51

51

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.

52

52

Variant ARIES algorithms

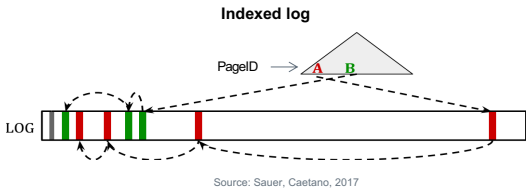
- ARIES/LHS - ARIES for Linear Hashing with Separators
  - handles the concurrency control and recovery with dynamic hash,
  - has not been implemented.
- ARIES/CSA - ARIES for the Client-Server Architecture
  - a client generates log records, and sends them to the server,
  - server:
    - stores the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

53

53

Log-structured recovery techniques

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

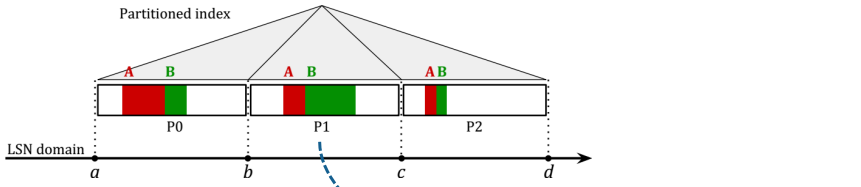


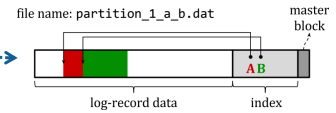
Source: Sauer, Caetano, 2017

54

54

Instant restore





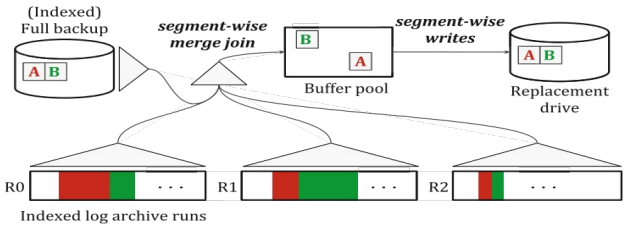
Source: Sauer, Caetano, 2017

55

55

Instant restore

A instant restore scenario



Source: Sauer, Caetano, 2017

56

56

4.
Main memory databases recovery

The time has come! Highlighting techniques.

Introdution -- Features of MMDB crashes -- Logging
Checkpoint -- Restart

57

57

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.

58

58

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.

59

59

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 MMDs 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.

60

60

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.

61

61

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 tha performing logical records.
- MMDB checkpoint avoids Fuzzy checkpoint.
- MMDB checkpoint usually performs an Copy-on-Update (COU) technique.

62

62

Checkpoint in MMDBs

Copy-on-Update (COU) snapshot algorithm

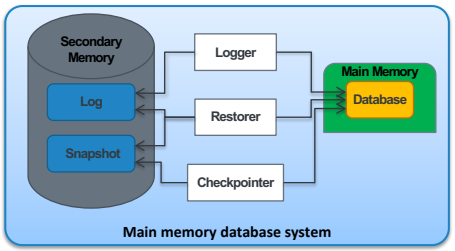
- COU is perfomed at system runtime.
- COU algorithm:
  - copies all database tuples to snapshot since checkpoint beggining;
  - skips inserted tuples;
  - copies updated and deleted tuples to shadow table;
  - copies all tuples from shadow table to snapshot;
  - a thread serializes the snapshot from 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.

63

63

Restart (recovery) in MMDBs

Default MMDB recovery



Source: Araújo, Arlino, 2022

- 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 perfomance *Vs.* transaction processing performance.

64

64

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

65

65

Some modern MMDBs and its main recovery features

MMDB	Concurrence Control	Logging	Checkpoint	Instant Recovery
Hekaton	optimistic MVCC	operation logging	delta and data files	no
VoltDB	serial execution	transaction logging	snapshot	no
HyPer	serial execution	transaction logging	snapshot	no
SAP HANA	MVCC, 2PC	transaction logging	snapshot	no
SiloR	optimistic MVCC	value logging	"fuzzy" snapshot	no
TimesTen	MVCC, 2PL	value logging	"fuzzy" checkpoint, snapshot	no
PACMAN	serial execution	transaction logging	-	no
Adaptive Logging	serial execution	transaction and ARIES logging	snapshot	no
FineLine	-	physiological logging	-	yes
HiEngine		physical logging		yes
MM-Direct	-	operation logging	tuple-level consistent checkpoint	yes

66

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 porposes.

67

67

Hekaton

- Logging
  - logical,
  - read-only,
  - log records
    - versions created, and
    - keys of deleted versions.

    They are generated at the commit time!
  - group commit.
- Checkpoint
  - two type of files
    - data file = inserted and updated versions
      - append-only,
      - read-only.
    - delta file = IDs of deleted versions
      - append-only,
  - data files can degrade recovery when the number of delta files increases,
    - adjacent data files must be merged.

68

68

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 falie
     - data/delta units can be loaded in parallel,
       - differents cores can load pair in parallel.
  2. replays log actions from checkpoint record.

69

69

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.

70

70

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.

71

71

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.

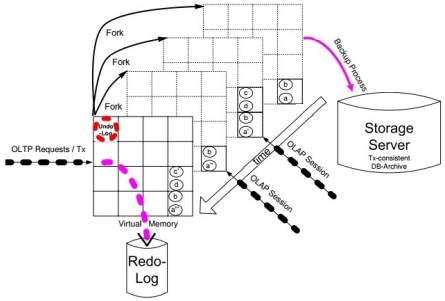
72

72

HyPer

Hybrid OLTP&OLAP High Performance

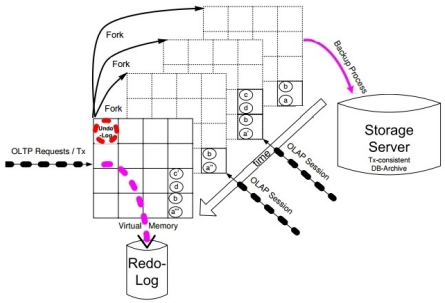
- ▷ It can handle both OLTP and OLAP workloads on the same database.
- ▷ OLAP queries can run on the most recent database state.
- ▷ It can generate a memory overhead from OLAP sessions.
- ▷ OLAP queries can access secondary storage when memory is low.



Source: Funke, Florian, et al., 2014

HyPer

- ▷ 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.

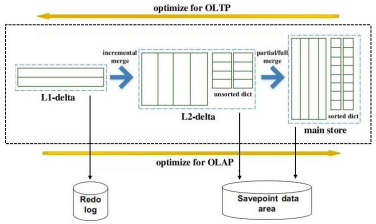


Source: Funke, Florian, et al., 2014

SAP HANA

High-performance ANalytic Appliance

- It can has multiple data processing engines:
  - the classical relational processing (both transactional and analytical),
  - text processing, and
  - graph processing.
- It implements a unified table structure:
  - optimized storage for OLTP transactions, and
  - highly compressed structure for OLAP queries.
- Three representations for tuples in a table:
  - L1-delta,
  - L2-delta,



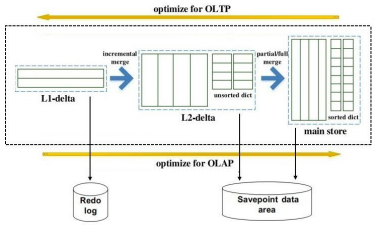
Source: Sikka, Vishal, et al., 2012

75

75

SAP HANA

- L1-delta
  - row logical format,
  - fast insertion, update, deletion, and record projection.
- L2-delta
  - column-store format (intermediate),
  - unsorted dictionary encoding,
  - bulk load operations.
- Main store
  - columnar store format,
  - sorted dictionary highest comprised



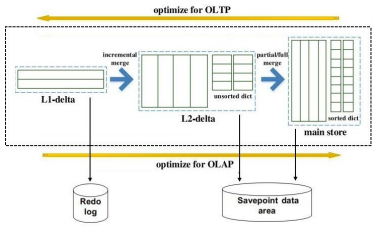
Source: Sikka, Vishal, et al., 2012

76

76

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.



Source: Sikka, Vishal, et al., 2012

77

77

PACMAN

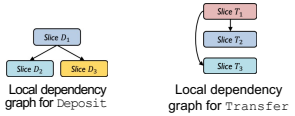
- It is a recovery mechanism that tries to:
  - parallelize the recovery, and
  - minimizes the runtime overhead for transaction processing.
- It was designed based on two prerequisites:
  1. the DBMS must utilize command logging,
  2. the DBMS must replay re-executing transactions to recover the database.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

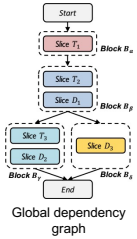
83

83

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.

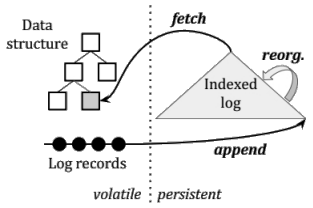




Source: Wu, Yingjun, et al., 2017

FineLine

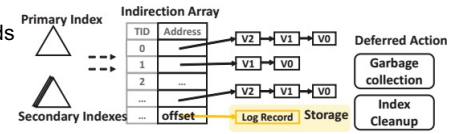
- It uses a structured log (partitioned B-tree) as the only way of persistence.
- It implements the same recovery strategy as Instant Restore.
  - Recovers tuples incrementally and on-demand.
  - It implements only a log file (physical and read-only).
  - After a system failure, the system performs an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



Source: Sauer, Caetano et al., 2018

HiEngine

- It is an MMDB developed by Huawei.
- It uses a sequential log whose records contains:
  - 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.

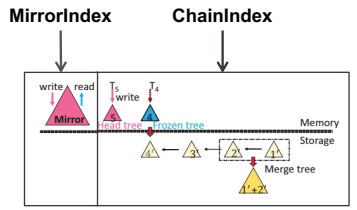


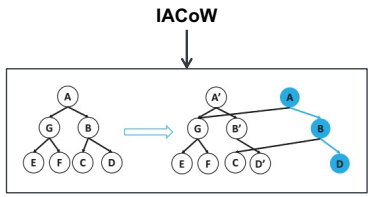
Source: Sauer, Caetano et al., 2018

HiEngine

Wait-free index checkpoint techniques:

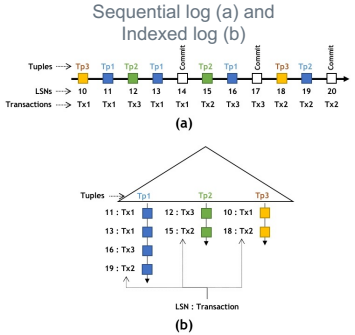
- ChainIndex
  - "frozens" the tree and regards it as an incremental snapshot.
- MirrorIndex
  - an extra mirror tree for index reads.
- IACoW
  - copy-on-write for tree structures to generate snapshots.





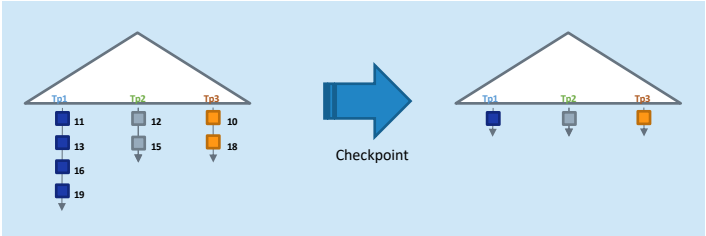
MM-Direct

- **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.

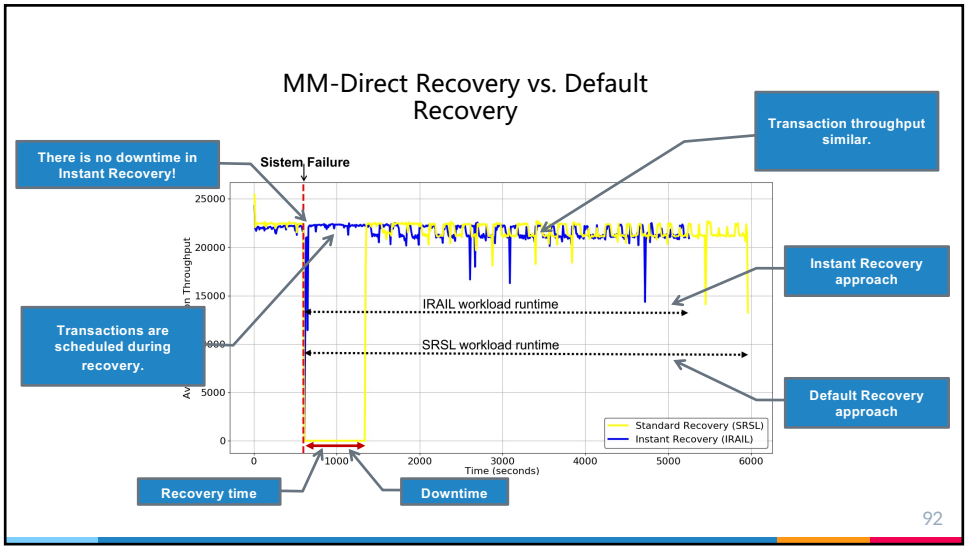


MM-Direct

- Tuple Consistent Checkpoint - TuCC



- Tuple Consistent Checkpoint for MFU tuples - TuCC-MFU



92

6.

Main challenges and future directions

The final level!

93

93

MMDB Recovery

- **Future directions**
  - Novel log structures.
  - Asynchronous logging processes.
  - New technologies.
- **Challenges**
  - MMDB life-time
    - New technologies
      - SSD with I/O bandwidth similar do DRAM

94

94

References

- Araújo, Arlino Henrique Magalhães de. Main memory database instant recovery. (2022). Availabe in: https://repositorio.ufc.br/handle/riufc/66902.
- Arulraj, Joy, Andrew Pavlo, and Prashanth Menon. **Bridging the archipelago between row-stores and column-stores for hybrid workloads**. Proceedings of the 2016 International Conference on Management of Data. 2016.
- Choi, Hyunsik. **A Survey of Query Execution Engines**. (2016) Available in: https://chatwithengineers.wordpress.com/2016/08/29/a-survey-of-query-execution-engines-from-volcano-to-vectorized-processing. Acesso em: August 4, 2022.
- Diaconu, Cristian, et al. **Hekaton: SQL server's memory-optimized OLTP engine.** Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. 2013.
- Faerber, Franz, et al. **Main memory database systems**. Foundations and Trends® in Databases 8.1-2 (2017): 1-130.
- Funke, Florian, et al. **HyPer beyond software: Exploiting modern hardware for main-memory database systems**. Datenbank-Spektrum 14.3 (2014): 173-181.
- Garcia-Molina, Hector, and Kenneth Salem. **Main memory database systems: An overview**. IEEE Transactions on knowledge and data engineering 4.6 (1992): 509-516.

95

95

References

- Graefe, Goetz, and Harumi Kuno. **Definition, detection, and recovery of single-page failures, a fourth class of database failures**. arXiv preprint arXiv:1203.6404 (2012).
- Haerder, Theo, and Andreas Reuter. **Principles of transaction-oriented database recovery**. ACM computing surveys (CSUR) 15.4 (1983): 287-317.
- Härder, Theo, et al. **Instant recovery with write-ahead logging**. Datenbank-Spektrum 15.3 (2015): 235-239.
- Lehman, Tobin J., Eugene J. Shekita, and L-F. Cabrera. **An evaluation of Starburst's memory resident storage component**. IEEE Transactions on knowledge and data Engineering 4.6 (1992): 555-566.
- Lee, Leon, et al. **Index checkpoints for instant recovery in in-memory database systems**. Proceedings of the VLDB Endowment 15.8 (2022): 1671-1683.
- Mohan, C. **Repeating history beyond ARIES**. VLDB. Vol. 99. 1999.
- Ren, Kun, Alexander Thomson, and Daniel J. Abadi. "**Lightweight locking for main memory database systems**. Proceedings of the VLDB Endowment 6.2 (2012): 145-156.
- Sauer, Caetano. **Modern techniques for transaction-oriented database recovery**. Gesellschaft für Informatik, Bonn, 2017.
- Sauer, Caetano, Goetz Graefe, and Theo Härder. **Instant restore after a media failure**. European Conference on Advances in Databases and Information Systems. Springer, Cham, 2017.

96

References

- Sauer, Caetano, Goetz Graefe, and Theo Härder. **Single-pass restore after a media failure**. Datenbanksysteme für Business, Technologie und Web (BTW 2015) (2015).
- Sikka, Vishal, et al. **Efficient transaction processing in SAP HANA database: the end of a column store myth**. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012.
- Ramakrishnan, Raghu, Johannes Gehrke, and Johannes Gehrke. **Database management systems**. Vol. 3. New York: McGraw-Hill, 2003.
- Sauer, Caetano, Goetz Graefe, and Theo Härder. **FineLine: log-structured transactional storage and recovery**. Proceedings of the VLDB Endowment 11.13 (2018): 2249-2262.
- VoltDB Documentation. Available in: **https://docs.voltdb.com**. Acesso em: December 12, 2020.
- Wu, Yingjun, et al. **Fast failure recovery for main-memory dbmss on multicores**. Proceedings of the 2017 ACM International Conference on Management of Data. 2017.
- Zhang, Hao, et al. "In-memory big data management and processing: A survey." IEEE Transactions on Knowledge and Data Engineering 27.7 (2015): 1920-1948.

97