Brainstorming: Cross-Engine Transactions

Global transaction metadata storage

Marko Mäkelä, Lead Developer InnoDB

InnoDB Concepts

Some terms that an Advanced DBA should be familiar with

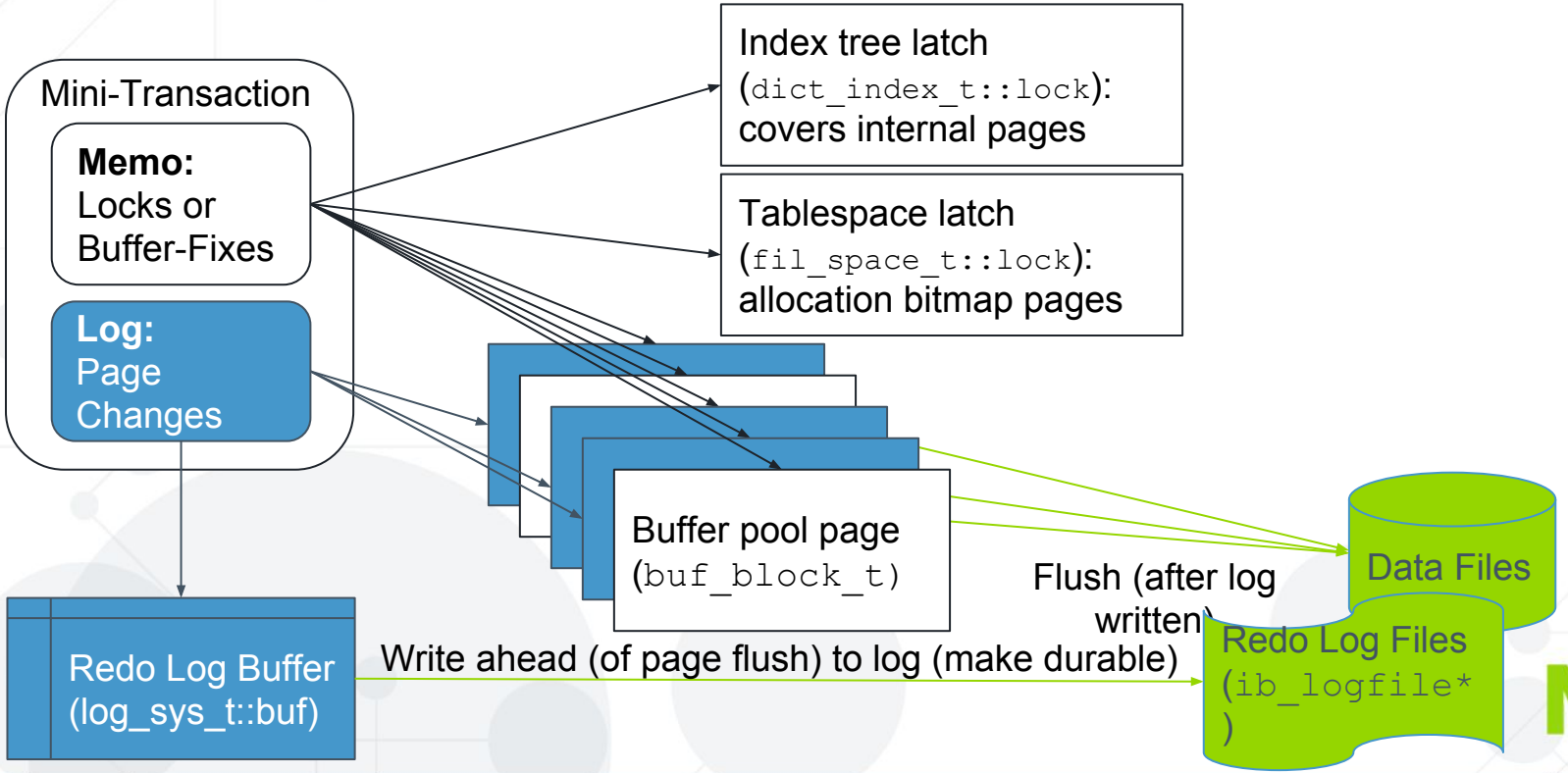
A **mini-transaction** is an atomic set of page reads or writes, with write-ahead **redo log**.

A **transaction** writes **undo log** before modifying **indexes**.

The **read view** of a transaction may access the undo logs of newer transactions to retrieve old versions.

Purge may remove old undo logs and **delete-marked records** once no read view needs them.

Mini-Transactions: RW-Locks and Redo Logs



Mini-transactions and Durability

Depending on when the redo log buffer was written to the redo log files, recovery may miss some latest mini-transaction commits. The most important mini-transaction commit is that of a

User transaction commit:

1. Update undo log directory and header pages.
2. Commit the mini-transaction (write to the redo log buffer).
3. Obtain the last written LSN (redo log sequence number) in the buffer.
4. Optionally, ensure that all redo log buffer is written at least up to the LSN.

One parameter related to the transaction durability is `innodb_flush_log_at_trx_commit`.

Mini-transaction Operations for Transactions

- *Start transaction:*
  - Assign `trx->id = trx_sys.max_trx_id++` (also known as `DB_TRX_ID`)
  - Create undo log header, update undo log directory (also known as “rollback segment page”).
  - Done on the first write of a transaction.
  - Never done for read-only transactions.
- *Write undo log record:*
  - Append to last undo page, or allocate a new undo page and write the record there.
  - InnoDB writes undo log in advance, before updating each index affected by the row operation.
- *Commit transaction:*
  - Assign end id (`trx->no`) from the same increasing sequence as `trx->id`.
  - Update undo log header and undo log directory (`trx->no` is unnecessarily written).
  - Change main-memory data structures: release locks, resume waiting transactions (and purge).

InnoDB Transaction Layer

The InnoDB transaction layer relies on atomically updated data pages forming persistent data structures:

- Page allocation bitmap, index trees, undo log directories, undo logs
- Undo logs are the glue that make the indexes of a table look consistent.
- On startup, any pending transactions (with implicit locks) will be recovered from undo logs.
  - Locks will be kept until incomplete transactions are rolled back, or
  - until explicit `XA COMMIT` or `XA ROLLBACK`. (These can be issued by binlog recovery too.)
- On startup, InnoDB must know which transactions are committed.
  - This is currently stored in the undo log header or undo log directory.
  - Proposal: `CREATE TABLE sys.transactions` for persistently storing transaction metadata.

Persistent Table for Transaction Metadata

- No `ROLLBACK`, undo logging, purge or MVCC for this table!
- Only 1 key: `start_time BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY`
  - Corresponds to the InnoDB DB_TRX_ID and PAGE_MAX_TRX_ID (48 bits)
  - `end_time` (`COMMIT/XA PREPARE`) uses the same underlying sequence
  - The `start_time`, `end_time` can also be used for temporal tables (“as of”)
- GTID, XID and other metadata would be part of the record
  - Uniqueness of GTID, XID is enforced by caching all transactions in RAM.
- Can be stored in any crash-safe engine (InnoDB, Aria, MyRocks)
  - **Unless and until we have a common crash recovery log for all engines, this must be partitioned by storage engine**
- Can be partitioned further for improved commit concurrency
  - Analogy: 128 InnoDB rollback segment header pages (undo log directories)

Cross-Engine R/W Transactions, XID, GTID

```
CREATE TABLE sys.transactions_of_$engine(
  start_id SERIAL COMMENT 'logical start time, like DB_TRX_ID',
  xid CHAR(128) NULL /*UNIQUE*/ COMMENT 'XA transaction ID',
  gtid CHAR(16) NOT NULL /*UNIQUE*/,
  state ENUM('active','prepared','pre_commit','committed') NOT NULL
) NO_ROLLBACK; -- any crash-safe $engine: Aria, InnoDB, MyRocks, ...
```

MariaDB

Removal of Transaction History

- Storage engines maintain private state needed for MVCC, locking, and `COMMIT`, `ROLLBACK`, `XA ROLLBACK`, `XA COMMIT`.
  - The `sys.transactions` record only tracks the transaction state.
- Referential integrity considerations (in particular, after crash recovery):
  - Engines must know all their pending `sys.transactions.start_id`
  - Must be able to `ROLLBACK` (or `COMMIT`) or `XA COMMIT` or `XA ROLLBACK`
- On completion, `ROLLBACK` can `DELETE FROM sys.transactions`.
- For GTID, we must preserve the latest sequence from each source domain
  - If no GTID, `COMMIT` can `DELETE FROM sys.transactions`.
- `XA PREPARE` transactions must remain until `XA COMMIT` or `XA ROLLBACK`

Crash Recovery with the `sys.transactions`

- Writes to the partitions of `sys.transactions` will be persisted to the recovery log of the underlying storage engine(s).
  - A transaction state change (`COMMIT`, `XA PREPARE`, `XA COMMIT`, `XA ROLLBACK`) is durable when the write to `sys.transactions` becomes durable.
- On startup, engines may `SELECT * FROM sys.transactions WHERE …;`
  - `MyRocks` would need this at least to determine XA transaction state.
  - `InnoDB` will do this for all `start_id` found in its undo logs.
- InnoDB could move undo logs into *`filename`*`.ibu` files
  - Easier to manage undo logs and to see how much space they take
  - Combined with some changes to `.ibd` files, we could
    - Remove or make optional the InnoDB system tablespace (MDEV-11633) and do
    - Support instant import of InnoDB tables by “simply copying files” (MDEV-11658)

Cross-Engine Commit Without Binlog

- Commit must be Atomic and Durable across storage engines
  - If we had a cross-engine write-ahead log, this would be trivial.
  - Some kind of 2-phase commit is needed for engines that use a private log.
- Possible idea to allow cross-engine commit without enabling binlog:
  - For each participating engine:
    - `UPDATE sys.transactions_of_$engine SET state='pre_commit';`
    - Flush the recovery log.
  - For each participating engine:
    - `UPDATE sys.transactions_of_$engine SET state='committed';`
  - Flush at least one recovery log.

Cross-Engine Transaction Recovery without Binlog

Read and merge the transaction metadata records into RAM.

- If state='`committed`' in any record for the same `start_id`:
  - Other records must have `state IN ('prepared','pre_commit')`.
  - `UPDATE…SET state='committed'` in all records.
  - Recover the transaction as committed.
- If `state='prepared'` in all records: recover as `XA PREPARE`.
  - The transaction state will remain until `XA ROLLBACK` or `XA COMMIT`.
- Otherwise: treat the transaction as incomplete, and initiate `ROLLBACK`:
  - Upon completion, each storage engine can `DELETE` the records.

Thank You

These slides were quickly composed and edited from 2 of my other presentations. This version was edited after the presentation for clarity

Thanks to Monty for pointing out that we do not really need to write a transaction end ID anywhere. (InnoDB only needs it for multi-versioning read views, including the purge read view.)

Thanks to Andrei Elkin and Monty for clarifying what exactly we need to preserve for GTID.

Hopefully, this idea could replace the GTID table, pave the way for removing the InnoDB system tablespace, and allow to have cross-engine transactions without enabling the binlog.