InnoDB to MyRocks migration in main MySQL database at Facebook

Yoshinori Matsunobu
Production Engineer, Facebook
May/2017
Who am I

- Was a MySQL consultant at MySQL (Sun, Oracle) for 4 years
- Joined Facebook in March 2012
  - MySQL 5.1 -> 5.6 upgrade
  - Fast master failover without losing data
  - Partly joined HBase Production Engineering team in 2014.H1
  - Started a research project to integrate RocksDB and MySQL from 2014.H2, with Database Engineering Team
- Started MyRocks deployment since 2016.H2
Agenda

- MySQL at Facebook
- Issues in InnoDB
- RocksDB and MyRocks overview
- Production Deployment
“Main MySQL Database” at Facebook

- Storing Social Graph
- Massively Sharded
- Petabytes scale
- Low latency
- Automated Operations
- Pure Flash Storage (Constrained by space, not by CPU/IOPS)
InnoDB Issue (1) -- Write Amplification

- 1 Byte Modification results in one page write (4~16KB)
- InnoDB “Doublewrite” doubles write volume
InnoDB Issue (2) -- B+Tree Fragmentation

INSERT INTO message_table (user_id) VALUES (31)
InnoDB Issue (3) -- Compression

Uncompressed 16KB page

Compressed to 5KB

Using 8KB space on storage

0~4KB => 4KB
4~8KB => 8KB
8~16KB => 16KB
RocksDB

- http://rocksdb.org/
- Forked from LevelDB
  - Key-Value LSM (Log Structured Merge) persistent store
  -Embedded
  - Data stored locally
  - Optimized for fast storage

- LevelDB was created by Google
- Facebook forked and developed RocksDB
- Used at many backend services at Facebook, and many external large services
- Needs to write C++ or Java code to access RocksDB
How LSM/RocksDB works

INSERT INTO message (user_id) VALUES (31);
INSERT INTO message (user_id) VALUES (99999);
INSERT INTO message (user_id) VALUES (10000);

WAL/ MemTable

existing SSTs

Compaction

Writing new SSTs sequentially

N random row modifications => A few sequential reads & writes
What is MyRocks

- MySQL on top of RocksDB (RocksDB storage engine)
- Open Source, distributed from MariaDB and Percona as well

http://myrocks.io/
MyRocks Goals

InnoDB in main database

CPU: 20%
IO: 15%
Space: 90%

MyRocks in main database

CPU: 18%
IO: 12%
Space: 45%
MyRocks goals (more details)

- Smaller space usage
  - 50% compared to compressed InnoDB at FB
- Better write amplification
  - Can use more affordable flash storage
- Fast, and small enough CPU usage with general purpose workloads
  - Large enough data that don’t fit in RAM
  - Point lookup, range scan, full index/table scan
  - Insert, update, delete by point/range, and occasional bulk inserts
  - Same or even smaller CPU usage compared to InnoDB at FB
  - Make it possible to consolidate 2x more instances per machine
MyRocks features

- Clustered Index (same as InnoDB)
- Bloom Filter and Column Family
- Transactions, including consistency between binlog and RocksDB
- Faster data loading, deletes and replication
- Dynamic Options
- TTL
- Online logical and binary backup
Facebook MySQL related teams

- Production Engineering (SRE)
  - MySQL Production Engineering
  - Data Performance
- Software Engineering
  - RocksDB Engineering
  - MySQL Engineering
    - MyRocks
    - Others (Replication, Client, InnoDB, etc)
Relationship between MySQL PE and SWE

- SWE does MySQL server upgrade
  - Upgrading MySQL revision with several FB patches
  - Hot fixes

- SWE participates in PE oncall
  - Investigating issues that might have been caused by MySQL server codebase
  - Core files analysis

- Can submit diffs each other

- “Hackaweek” to work on shorter term tasks for a week
MyRocks migration -- Technical Challenges

- Initial Migration
  - Creating MyRocks instances without downtime
  - Loading into MyRocks tables within reasonable time
  - Verifying data consistency between InnoDB and MyRocks

- Continuous Monitoring
  - Resource Usage like space, iops, cpu and memory
  - Query plan outliers
  - Stalls and crashes
MyRocks migration -- Technical Challenges (2)

- When running MyRocks on master
  - RBR (Row based binary logging)
  - Removing queries relying on InnoDB Gap Lock
  - Robust XA support (binlog and RocksDB)
Creating first MyRocks instance without downtime

- Picking one of the InnoDB slave instances, then starting logical dump and restore
  - Stopping one slave does not affect services
Faster Data Loading

Normal Write Path in MyRocks/RocksDB

- Write Requests
- WAL
- MemTable
  - Level 0 SST
  - Level 1 SST
  - ....
  - Level max SST

Faster Write Path

- Write Requests
- Level max SST

“SET SESSION rocksdb_bulk_load=1;”
Original data must be sorted by primary key
Data migration steps

- **Dst** Create table ... ENGINE=ROCKSDB; (creating MyRocks tables with proper column families)
- **Dst** ALTER TABLE DROP INDEX; (dropping secondary keys)
- **Src** STOP SLAVE;
- **mysqldump** --host=innodb-host --order-by-primary | **mysql** --host=myrocks-host --init-command="set sql_log_bin=0; SET rocksdb_bulk_load=1"
- **Dst** ALTER TABLE ADD INDEX; (adding secondary keys)
- **Src, Dst** START SLAVE;
Data Verification

- MyRocks/RocksDB is relatively new database technology
- Might have more bugs than robust InnoDB
- Ensuring data consistency helps avoid showing conflicting results
Verification tests

- Index count check between primary key and secondary keys
  - If any index is broken, it can be detected
  - `SELECT 'PRIMARY', COUNT(*) FROM t FORCE INDEX (PRIMARY)
    UNION SELECT 'idx1', COUNT(*) FROM t FORCE INDEX (idx1)`
  - Can’t be used if there is no secondary key

- Index stats check
  - Checking if “rows” show `SHOW TABLE STATUS` is not far different from actual row count

- Checksum tests w/ InnoDB
  - Comparing between InnoDB instance and MyRocks instance
  - Creating a transaction consistent snapshot at the same GTID position, scan, then compare checksum

- Shadow correctness check
  - Capturing read traffics
Shadow traffic tests

- We have a shadow test framework
  - MySQL audit plugin to capture read/write queries from production instances
  - Replaying them into shadow master instances

- Shadow master tests
  - Client errors
  - Rewriting queries relying on Gap Lock
    - gap_lock_raise_error=1, gap_lock_write_log=1
Creating second MyRocks instance without downtime

Master (InnoDB)

Slave1 (InnoDB)  Slave2 (InnoDB)  Slave3 (MyRocks)  Slave4 (MyRocks)

myrocks_hotbackup (Online binary backup)
Promoting MyRocks as a master

- Master (MyRocks)
- Slave1 (InnoDB)
- Slave2 (InnoDB)
- Slave3 (InnoDB)
- Slave4 (MyRocks)
Crash Safety

- Crash Safety makes operations much easier
  - Just restarting failed instances can restart replication
  - No need to rebuild entire instances, as long as data is there

- Crash Safe Slave
- Crash Safe Master
- 2PC (binlog and WAL)
# Master crash safety settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>sync-binlog</th>
<th>rocksdb-flush-log-at-trx-commit</th>
<th>rocksdb-enable-2pc</th>
<th>rocksdb-wal-recovery-mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>No data loss on unplanned machine reboot</td>
<td>1 (default)</td>
<td>1 (default)</td>
<td>1 (default)</td>
<td>1 (default)</td>
</tr>
<tr>
<td>No data loss on mysqld crash &amp; recovery</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>No data loss if always failover</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Notable issues fixed during migration

- Lots of “Snapshot Conflict” errors
  - Because of implementation differences in MyRocks (PostgreSQL Style snapshot isolation)
  - Setting tx-isolation=READ-COMMITTED solved the problem

- Slaves stopped with I/O errors on reads
  - We switched to make MyRocks abort on I/O errors for both reads and writes

- Index statistics bugs
  - Inaccurate row counts/length on bulk loading -> fixed
  - Cardinality was not updated on fast index creation -> fixed

- Crash safety bugs and some crash bugs in MyRocks/RocksDB
Preventing stalls

- Heavy writes cause lots of compactions, which may cause stalls

- Typical write stall cases
  - Online schema change
  - Massive data migration jobs that write lots of data

- Use fast data loading whenever possible
  - InnoDB to MyRocks migration can utilize this technique
  - Can be harder for data migration jobs that write into existing tables
When write stalls happen

- Estimated number of pending compaction bytes exceeded X bytes
  - soft|hard_pending_compaction_bytes, default 64GB
- Number of L0 files exceeded level0_slowdown|stop_writes_trigger (default 10)
- Number of unflushed number of MemTables exceeded max_write_buffer_number (default 4)

- All of these incidents are written to LOG as WARN level
- All of these options apply to each column family
What happens on write stalls

- **Soft stalls**
  - COMMIT takes longer time than usual
  - Total estimated written bytes to MemTable is capped to rocksdb_delayed_write_rate, until slowdown conditions are cleared
    - Default is 16MB/s (previously 2MB/s)

- **Hard stalls**
  - All writes are blocked at COMMIT, until stop conditions are cleared
Mitigating Write Stalls

- Speed up compactions
  - Use faster compression algorithm (LZ4 for higher levels, ZSTD in the bottommost)
  - Increase rocksdb_max_background_compactions
- Reduce total bytes written
  - But avoid using too strong compression algorithm on upper levels
  - Use more write efficient compaction algorithm
    - compaction_pri=kMinOverlappingRatio
- Delete files slowly on Flash
  - Deleting too many large files cause TRIM stalls on Flash
  - MyRocks throttles sst file deletes by 64MB/s by default
  - Binlog file deletions should be slowed down as well
Monitoring

- MyRocks files
- SHOW ENGINE ROCKSDB STATUS
- SHOW ENGINE ROCKSDB TRANSACTION STATUS
- LOG files
- information_schema tables
- sst_dump tool
- ldb tool
SHOW ENGINE ROCKSDB STATUS

- Column Family Statistics, including size, read and write amp per level
- Memory usage

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Status</th>
<th>Level</th>
<th>Files</th>
<th>Size(MB)</th>
<th>Score</th>
<th>Read(GB)</th>
<th>Rn(GB)</th>
<th>Rpnl(GB)</th>
<th>Write(GB)</th>
<th>Wnew(GB)</th>
<th>Moved(GB)</th>
<th>W-Amp</th>
<th>Rd(MB/s)</th>
<th>Wr(MB/s)</th>
<th>Comp(sec)</th>
<th>Comp(cnt)</th>
<th>Avg(sec)</th>
<th>KeyIn</th>
<th>KeyDrop</th>
</tr>
</thead>
<tbody>
<tr>
<td>L0</td>
<td>CF_COMPACTION</td>
<td>default</td>
<td>** Compaction Stats [default]**</td>
<td>L0</td>
<td>2/0</td>
<td>51.58</td>
<td>0.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>40.3</td>
<td>7</td>
<td>10</td>
<td>0.669</td>
<td>0</td>
</tr>
<tr>
<td>L3</td>
<td></td>
<td></td>
<td></td>
<td>6/0</td>
<td>109.36</td>
<td>0.9</td>
<td>0.7</td>
<td>0.7</td>
<td>0.0</td>
<td>0.6</td>
<td>0.6</td>
<td>0.0</td>
<td>0.9</td>
<td>43.8</td>
<td>40.7</td>
<td>16</td>
<td>3</td>
<td>5.172</td>
<td>7494K</td>
<td>297K</td>
</tr>
<tr>
<td>L4</td>
<td></td>
<td></td>
<td></td>
<td>61/0</td>
<td>1247.31</td>
<td>1.0</td>
<td>2.0</td>
<td>0.3</td>
<td>1.7</td>
<td>2.0</td>
<td>0.2</td>
<td>0.0</td>
<td>6.9</td>
<td>49.7</td>
<td>48.5</td>
<td>41</td>
<td>9</td>
<td>4.593</td>
<td>15M</td>
<td>176K</td>
</tr>
<tr>
<td>L5</td>
<td></td>
<td></td>
<td></td>
<td>989/0</td>
<td>12592.86</td>
<td>1.0</td>
<td>2.0</td>
<td>0.3</td>
<td>1.8</td>
<td>1.9</td>
<td>0.1</td>
<td>0.0</td>
<td>7.4</td>
<td>8.1</td>
<td>7.4</td>
<td>258</td>
<td>8</td>
<td>32.209</td>
<td>17M</td>
<td>726K</td>
</tr>
<tr>
<td>L6</td>
<td></td>
<td></td>
<td></td>
<td>4271/0</td>
<td>127363.51</td>
<td>1.0</td>
<td>2.0</td>
<td>0.3</td>
<td>1.8</td>
<td>1.9</td>
<td>0.1</td>
<td>0.0</td>
<td>7.4</td>
<td>8.1</td>
<td>7.4</td>
<td>258</td>
<td>8</td>
<td>32.209</td>
<td>17M</td>
<td>726K</td>
</tr>
<tr>
<td>Sum</td>
<td></td>
<td></td>
<td></td>
<td>5329/0</td>
<td>141364.62</td>
<td>0.0</td>
<td>4.7</td>
<td>1.2</td>
<td>3.5</td>
<td>4.7</td>
<td>1.2</td>
<td>0.0</td>
<td>17.9</td>
<td>15.0</td>
<td>15.0</td>
<td>321</td>
<td>30</td>
<td>10.707</td>
<td>41M</td>
<td>1200K</td>
</tr>
</tbody>
</table>
### SHOW GLOBAL STATUS

```sql
mysql> show global status like 'rocksdb%';
```

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rocksdb_rows_deleted</td>
<td>216223</td>
</tr>
<tr>
<td>rocksdb_rows_inserted</td>
<td>1318158</td>
</tr>
<tr>
<td>rocksdb_rows_read</td>
<td>7102838</td>
</tr>
<tr>
<td>rocksdb_rows_updated</td>
<td>1997116</td>
</tr>
<tr>
<td>rocksdb_bloom_filter_prefix_checked</td>
<td>773124</td>
</tr>
<tr>
<td>rocksdb_bloom_filter_prefix_useful</td>
<td>308445</td>
</tr>
<tr>
<td>rocksdb_bloom_filter_useful</td>
<td>10108448</td>
</tr>
</tbody>
</table>
```
```
Summary

- We started migrating from InnoDB to MyRocks in our main database
  - We run both master and slaves in production
- Major motivation was to save space
- Online data correctness check tool helped to find lots of data integrity bugs and prevented from deploying inconsistent instances in production
- Bulk loading greatly reduced compaction stalls
- Transactions and crash safety are supported