How we managed to scale Percona XtraDB Cluster (PXC)

Krunal Bauskar
PXC Product Lead @ Percona
Agenda

- PXC Performance
- Performance fixes
PXC Performance

Sysbench: dataset 100tables/4M rows(100GB)
innodb_buffer_pool=150GB,innodb_doublewrite=1,
innodb_flush_log_at_trx_commit=1, sync_binlog=1
Box: 28 Cores+HT

Transactions per second, tps

<table>
<thead>
<tr>
<th>Threads</th>
<th>OLTP_RW</th>
<th>UPDATE_KEY</th>
<th>UPDATE_NO_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2,000</td>
<td>7,000</td>
<td>10,000</td>
</tr>
<tr>
<td>8</td>
<td>18,000</td>
<td>23,000</td>
<td>25,000</td>
</tr>
<tr>
<td>16</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>32</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>64</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>128</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>256</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>512</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
<tr>
<td>1024</td>
<td>23,000</td>
<td>23,000</td>
<td>23,000</td>
</tr>
</tbody>
</table>

Percona XtraDB Cluster: 5.7.16 5.7.17
PXC Performance

Sysbench: dataset 100tables/4M rows(100GB)
innodb_buffer_pool=150GB,innodb_doublewrite=1,
innodb_flush_log_at_trx_commit=1, sync_binlog=1
Box: 28 Cores+HT

3x-10X Improvement
PXC Performance

Sysbench: dataset 100tables/4M rows(100GB),
innodb_buffer_pool=150GB,innodb_doublewrite=1,
innnodb_flush_log_at_trx_commit=1,no_binlog
Box: 28 Cores+HT

<table>
<thead>
<tr>
<th>OLTP_RW</th>
<th>UPDATE_KEY</th>
<th>UPDATE_NO_KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Graph OLTP_RW" /></td>
<td><img src="image2" alt="Graph UPDATE_KEY" /></td>
<td><img src="image3" alt="Graph UPDATE_NO_KEY" /></td>
</tr>
</tbody>
</table>

Transactions per second

Percona XtraDB Cluster: 5.7.16, 5.7.17

© 2018 Percona
DML execution in MySQL
DML execution in MySQL

- PARSE
- OPTIMIZE
- REWRITE
- EXECUTE
- PREPARE
- COMMIT
MySQL Group Commit Protocol

- MySQL Group Commit protocol
  - Batch transactions in group and let leader perform the action while followers wait.
  - Batching actions like flush, sync, commit (including redo flush) help reduce number of fsync and achieve better throughput.

1. FLUSH BINARY LOG (also flushed REDO log)
2. SYNC BINARY LOG
3. COMMIT TRANSACTION in Storage Engine
PXC Commit Monitor

- **PXC follow strict commit ordering**
  - Transaction that commits first to the group channel should be first to get committed too.
  - Each Transaction is assigned a unique identifier when it replicates on group channel.
Commit Protocol

- **Binary Log enabled**
  - Old Commit Protocol
  - New Commit Protocol
- **Binary Log disabled**
  - Old Commit Protocol
  - New Commit Protocol
Old-Commit Protocol - Binary log enabled
Old-Commit Protocol - Binary log enabled

- Binary Logging is **enabled**

REPLICATE to group-channel

Enter CommitMonitor

Update UNDO log to reflect PREPARE state

Flush/Sync REDO Log

Flush BINARY Log

Exit CommitMonitor

Commit InnoDB transaction in memory

Sync BINARY Log

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.

pxc action
mysql action
prepare phase
commit phase

© 2018 Percona
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

MySQL GROUP COMMIT starts with enqueuing of transaction

- REPLICATE to group-channel
- Enter CommitMonitor
- Update UNDO log to reflect PREPARE state
- Flush/Sync REDO Log
- Flush BINARY Log
- Commit InnoDB transaction in memory
- Sync BINARY Log
- Exit CommitMonitor

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Old-Commit Protocol - Binary log enabled

- Binary Logging is **enabled**

**REPLICATE to group-channel** → **Enter CommitMonitor** → **Update UNDO log to reflect PREPARE state** → **Flush/Sync REDO Log** → **Flush BINARY Log**

**Exit CommitMonitor** → **Commit InnoDB transaction in memory** → **Sync BINARY Log**

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.

pxc action
mysql action
prepare phase
commit phase

© 2018 Percona
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

REPLICATE to group-channel → Enter CommitMonitor → Update UNDO log to reflect PREPARE state → Flush/Sync REDO Log → Flush BINARY Log → Exit CommitMonitor → Commit InnoDB transaction in memory → Sync BINARY Log

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

1. **REPLICATE to group-channel**
2. **Enter CommitMonitor**
3. **Update UNDO log to reflect PREPARE state**
4. **Flush/Sync REDO Log**
5. **Flush BINARY Log**
6. **Commit InnoDB transaction in memory**
7. **Sync BINARY Log**
8. **Exit CommitMonitor**

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Binary Logging is **enabled**

- **REPLICATE to** group-channel
- **Enter CommitMonitor**
- **Update UNDO log to reflect PREPARE state**
- **Flush/Sync REDO Log**
- **Flush BINARY Log**
- **Exit CommitMonitor**
- **Commit InnoDB transaction in memory**
- **Sync BINARY Log**

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

REPLICATE to group-channel
Enter CommitMonitor
Update UNDO log to reflect PREPARE state
Flush/Sync REDO Log
Flush BINARY Log
Sync BINARY Log
Commit InnoDB transaction in memory
Exit CommitMonitor

pxc action
mysql action
prepare phase
commit phase

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

- {REPLICATE to group-channel} Enter CommitMonitor
- Update UNDO log to reflect PREPARE state
- Flush/Sync REDO Log
- Flush BINARY Log
- Exit CommitMonitor
- Commit InnoDB transaction in memory
- Sync BINARY Log

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.

pxc action
mysql action
prepare phase
commit phase
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

REPLICATE to group-channel

Enter CommitMonitor

Update UNDO log to reflect PREPARE state

Flush/Sync REDO Log

Flush BINARY Log

Commit InnoDB transaction in memory

Sync BINARY Log

Exit CommitMonitor

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.

pxc action
mysql action
prepare phase
commit phase

© 2018 Percona
Old-Commit Protocol - Binary log enabled

- Binary Logging is **enabled**

1. **REPLICATE to group-channel**
2. **Enter CommitMonitor**
3. **Update UNDO log to reflect PREPARE state**
4. **Flush/Sync REDO Log**
5. **Flush BINARY Log**
6. **Exit CommitMonitor**
7. **Commit InnoDB transaction in memory**
8. **Sync BINARY Log**

Commit Monitor will allow only one transaction to enter through it and there-by **SERIALIZING** all the following steps.

Each transaction results in fsync. No batch commit advantage explored.

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
**Old-Commit Protocol - Binary log enabled**

- Binary Logging is **enabled**

**5 steps critical section**

1. **REPLICATE to group-channel**
2. **Enter CommitMonitor**
3. **Update UNDO log to reflect PREPARE state**
4. **Flush/Sync REDO Log**
5. **Flush BINARY Log**

**GOAL is to maintain COMMIT ORDERING**

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
Old-Commit Protocol - Binary log enabled

- Binary Logging is enabled

MySQL already enforces **COMMIT ORDERING** and also takes advantage of batch commit using **GROUP COMMIT PROTOCOL**

5 steps critical section

1. REPLICATE to group-channel
2. Enter CommitMonitor
3. Update UNDO log to reflect PREPARE state
4. Flush/Sync REDO Log
5. Flush BINARY Log

Since binary log is flushed, REDO log capturing commit of transaction can be flushed later.
New-Commit Protocol - Binary log enabled
New-Commit Protocol - Binary log enabled

- Help accelerate both paths (binary log enabled/disabled)

4 steps critical section

1. **REPLICATE to group-channel**
2. **Update UNDO log to reflect PREPARE state**
3. **Enter CommitMonitor**
4. **Flush/Sync REDO Log**
5. **Flush BINARY Log**
6. **Exit CommitMonitor**
7. **Commit InnoDB transaction in memory**
8. **Sync BINARY Log**

**CHANGE-1**
replicate and commit ordering doesn’t need to go together. commit ordering can be enforced after transaction has update UNDO state to prepare.
Help accelerate binary log enabled path

1 steps critical section

REPLICATE to group-channel
Update UNDO log to reflect PREPARE state
Enter CommitMonitor
Interim Commit
Flush/Sync REDO Log

Commit InnoDB transaction in memory
Sync BINARY Log
Flush BINARY Log

CHANGE-2
Grab commit monitor immediately after PREPARE and release it once transaction is added to GROUP COMMIT QUEUE.
New-Commit Protocol - Binary log **enabled**

- Help accelerate binary log **enabled** path

```
<table>
<thead>
<tr>
<th>REPLICARe to group-channel</th>
<th>Update UNDO log to reflect PREPARE state</th>
<th>Enter CommitMonitor</th>
<th>Interim Commit</th>
<th>Flush/Sync REDO Log</th>
</tr>
</thead>
</table>
```

Each transaction is added to commit queue maintained by **MySQL Group Commit** Protocol. Group Commit enforces ordering so **Commit Monitor** is no more needed to enforce the same post this point.
**New-Commit Protocol - Binary log enabled**

- Help accelerate binary log enabled path

We still need to enter and exit **CommitMonitor** to enforce ordering of transaction based on the global seqno assigned them during replicate.
Old-Commit Protocol - Binary log disabled
Old-Commit Protocol - Binary log disabled

- Binary Logging is disabled

4 steps critical section
Old-Commit Protocol - Binary log disabled

- Binary Logging is disabled

- Flushing REDO log twice
- First time: PREPARE stage
- Second time: COMMIT stage

4 steps critical section
New-Commit Protocol - Binary log disabled
New-Commit Protocol - Binary log disabled

- Binary Logging is disabled

**CHANGE-1**
replicate and commit ordering doesn’t need to go together. commit ordering can be enforced after transaction has update UNDO state to prepare. This allow first REDO flush to go in parallel.

2 steps critical section
New-Commit Protocol - Binary log **disabled**

- Binary Logging is **disabled**

**CHANGE-2**
Final FLUSH of REDO log doesn’t need Commit Ordering enforced as the transaction is already committed and REDO log advances sequentially.
Summarize

- **Change-1:** Delay entering into Commit Monitor (enter only when needed)
- **Change-2:** Release Commit Monitor once transaction is queue in Group Commit Protocol Queue
- **Change-3:** Commit ordering is not needed for final REDO flush (log-bin=off).

All this and more in [PERCONA-XTRADB-CLUSTER-5.7.17+](https://www.percona.com/software/xtradb-cluster) (Latest Version: 5.7.21).

Give it a try
Connect back.....

- mail me: krunal.bauskar@percona.com
- PXC forum https://www.percona.com/forums/questions-discussions/percona-xtradb-cluster
- PXC @ JIRA https://jira.percona.com/projects/PXC/issues

Q&A

RATE THE TALK
Database Performance Matters