MySQL High Availability with Percona XtraDB Cluster 5.7
Hands on tutorial!
Introducing pxc-strict-mode

Krunal Bauskar
krunal.bauskar@percona.com

Alok Pathak
alok.pathak@percona.com

Peter Zaitsev
pz@percona.com

Krunal Bauskar
krunal.bauskar@percona.com
Table of Contents

- Introduction to PXC
- Setting up Environment
- Bootstrapping
- State Transfers
- Certification Errors
- Flow Control
- What's new with PXC-5.7
- Introducing pxc_strict_mode
- Exploring PXC through PFS
- Securing PXC
- Load Balancers
- Percona Monitoring and Management (PMM)
Introduction to PXC

- True Multi-master replication
- Synchronous replication
- Data consistency
- Multi-threaded Slave
- Transparent to Applications
- No complex failover
- Automatic Node Provisioning
Introduction to PXC

Clients

- read/write
- read/write
- read/write

Node 1
wsrep API

Node 2
wsrep API

Node 3
wsrep API

Galera Replication
Are you ready?

Setting Up The Environment
Setting Up The Environment

Fetch a USB Stick, copy the **pxc57.ova** virtualbox image
Open Virtualbox, File -> Import Appliance -> Browse the pxc57.ova file
Setting Up The Environment

Click Next and Import
Setting Up The Environment

Start pxc[1-3] virtual machines one by one
Setting Up The Environment

- Versions

Server version: **5.7.14-8-57-log** Percona XtraDB Cluster (GPL)
ProxySQL version **1.2.3**
Attention - Hands On!

When you see in the right bottom, there is an exercise that you should do!
Testing The Environment

- ssh/putty to:
  - pxc1: `ssh root@localhost -p 8821`
  - pxc2: `ssh root@localhost -p 8822`
  - pxc3: `ssh root@localhost -p 8823`

- root password is `percona password`

- Verify ssh between nodes

- Open 2 ssh sessions to every node
Bootstrapping A Cluster

- Bootstrapping a node gives it permission to form a new cluster
- The bootstrapped node is the source of truth for all nodes going forward
- Bootstrapping should NOT happen automatically without a system with split-brain protection that can coordinate it. Usually this is done manually
Bootstrapping PXC

```bash
# service mysql bootstrap-pxc
#/etc/init.d/mysql bootstrap-pxc
#/etc/init.d/mysql start --wsrep-new-cluster
```

or with `systemd` environments like Centos 7:

```bash
# systemctl start mysql@bootstrap
```

This sets empty gcomm:// address which starts the node as the first of the cluster. Never use it to join an existing cluster.

Today we are using 64-bit Centos 7.

```bash
pxc1# systemctl start mysql@bootstrap
```
Bootstrapping PXC

The node you bootstrapped..

Verify it with:

```bash
[root@pxc1 ~]# mysql
pxc1 mysql> show global status like 'wsrep%';
```

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsrep_local_state</td>
<td>4</td>
</tr>
<tr>
<td>wsrep_local_state_comment</td>
<td>Synced</td>
</tr>
<tr>
<td>wsrep_cluster_size</td>
<td>1</td>
</tr>
<tr>
<td>wsrep_cluster_status</td>
<td>Primary</td>
</tr>
<tr>
<td>wsrep_connected</td>
<td>ON</td>
</tr>
<tr>
<td>wsrep_ready</td>
<td>ON</td>
</tr>
</tbody>
</table>

59 rows in set (0.00 sec)
State Transfers

Recap On IST/SST

- SST: State Snapshot Transfer
  Snapshot the whole database and transfer, using:

- IST: Incremental State Transfer
  Only transfer missing transactions

- One node of the cluster is DONOR
## State Snapshot Transfers (SST)

### State Snapshot Transfer Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>Type</th>
<th>Speed</th>
<th>Blocks Donor</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysqldump</td>
<td>Logical</td>
<td>Slow</td>
<td>Blocks</td>
</tr>
<tr>
<td>rsync</td>
<td>Physical</td>
<td>Fastest</td>
<td>Blocks</td>
</tr>
<tr>
<td>xtrabackup-v2</td>
<td>Physical</td>
<td>Fast</td>
<td>Briefly</td>
</tr>
</tbody>
</table>

The recommended SST method is xtrabackup-v2 because it locks donor for least amount of time.

```sql
wsrep_sst_method = xtrabackup-v2
wsrep_sst_auth = sst:s3cret
wsrep_sst_donor=<donor_list>
```
Adding new nodes to the Cluster

pxc2# systemctl start mysql
pxc3# systemctl start mysql

More verification

[root@pxc3 ~]# mysql
pxc3 mysql> show global status like 'wsrep%';
+---------------------------------------------+--------+
| Variable_name                              | Value  |
+---------------------------------------------+--------+
| ...                                         | ...    |
| wsrep_local_state                          | 4      |
| wsrep_local_state_comment                  | Synced |
| ...                                         | ...    |
| wsrep_cluster_size                         | 3      |
| ...                                         | ...    |
| wsrep_cluster_status                       | Primary|
| wsrep_connected                            | ON     |
| ...                                         | ...    |
+---------------------------------------------+--------+
59 rows in set (0.00 sec)
**myq_status**

Monitor Galera metrics using `myq_status`:

```
# myq_status wsrep
mycluster / pxc1 (idx: 0) / Galera 3.17(r447d194)
```

<table>
<thead>
<tr>
<th>Wsrep</th>
<th>Cluster Node</th>
<th>Repl Queue</th>
<th>Ops</th>
<th>Bytes</th>
<th>Conflicting</th>
<th>Gcache</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>P cnf</td>
<td>Stat Laten</td>
<td>Up</td>
<td>Dn</td>
<td>Up</td>
<td>Dn</td>
<td>lcf</td>
</tr>
<tr>
<td>06:31:14</td>
<td>P 3 3 Sync 2.5ms</td>
<td>0 0 22 0 37K</td>
<td>0.0</td>
<td>0 0 14k</td>
<td>163</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>06:31:15</td>
<td>P 3 3 Sync 2.5ms</td>
<td>0 0 12 0 20K</td>
<td>0.0</td>
<td>0 0 14k</td>
<td>177</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>06:31:16</td>
<td>P 3 3 Sync 2.5ms</td>
<td>0 0 22 0 37K</td>
<td>0.0</td>
<td>0 0 14k</td>
<td>205</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>06:31:17</td>
<td>P 3 3 Sync 2.5ms</td>
<td>0 0 22 0 37K</td>
<td>0.0</td>
<td>0 0 14k</td>
<td>47</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

- **P** = Primary: The node is part of a primary component (N = Non-primary)
- **cnf** = cluster config id: This increments whenever the cluster state changes
- **Node Stat - Sync** is normal. Other possible states: Dono, Init
- **Repl Latency**: Average time for a transaction to replicate across the cluster
- **Queue**: Replication queues (Up is outbound, Dn is inbound). Dn queue -> apply queue
- **Ops**: Replication events. Up Ops is data that was written on this node by a client
- **Bytes**: Replication size (same as Ops, but in Bytes)
- **Conflicting**: lcf and bfa are replication conflicts
Bootstrapping PXC
With Stop/Start Of MySQL

When you need to start a new cluster from scratch, you decide which node to start with and you bootstrap it

```bash
# systemctl start mysql@bootstrap
```

That node becomes the cluster source of truth (SSTs for all new nodes)
Bootstrapping PXC
Without Restarting MySQL

When a cluster is already partitioned and you want to bring it up again.

- 1 or more nodes need to be in **Non-Primary** state.
- Choose the node that is newest and can be enabled (to work with application)
- To bootstrap online:

```
mysql> set global wsrep_provider_options="pc.bootstrap=true";
```
- be sure there is **NO OTHER PRIMARY** partition or there will be a split brain!!
Bootstrapping PXC
Without Restarting MySQL

Use Case:

- Only 1 of the 3 nodes is available and the other 2 nodes crashed, causing node 1 to go **Non-Primary**.
- In Multi Datacenter environments:
  - DC1 has 2 nodes, DC2 has 1 node,
  - If DC1 dies, the single node in DC2 will go **Non-Primary**. To activate secondary DC, a bootstrap is necessary
Recover Cleanly Shutdown Cluster

- Run the application (run_app.sh proxysql) on pxc1

```bash
[root@pxc1 ~]# /root/bin/run_app.sh proxysql
```

- One by one, stop mysql on all 3 nodes

```bash
pxc1# systemctl stop mysql@bootstrap
pxc2# systemctl stop mysql
pxc3# systemctl stop mysql
```

How to restart Galera Cluster?
Recover Cleanly Shutdown Cluster

To restart an entire Galera Cluster, complete the following steps:

- Identify the node with the highest `seqno`.
- Start the most advanced node as the first node of the cluster.
- Start the rest of the node as usual.

How to identify most advanced node?
Recover Cleanly Shutdown Cluster

To restart an entire Galera Cluster, complete the following steps:

- Identify the node with the highest `seqno`.
- Start the most advanced node as the first node of the cluster.
- Start the rest of the node as usual.

How to identify most advanced node?

Solution

```
# cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid: e5a514d5-80f2-11e6-8163-ea69a6a66f6e
seqno: 404412
cert_index:
```

Bootstrap node with highest `seqno` and start other nodes.
What if node crashed?

On a crashed node grastate.dat will look like this:

```
cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid:   e5a514d5-80f2-11e6-8163-ea69a6a66f6e
seqno:  -1
cert_index:
```

How can you know the seqno?
What if node crashed?

On a crashed node grastate.dat will look like this:

```bash
cat /var/lib/mysql/grastate.dat
# GALERA saved state
version: 2.1
uuid: e5a514d5-80f2-11e6-8163-ea69a6a66f6e
seqno: -1
cert_index:
```

How can you know the seqno?

Solution

```bash
# mysql_safe --wsrep-recover
Logging to '/var/lib/mysql/error.log'.
Starting mysqld daemon with databases from /var/lib/mysql
WSREP: Running position recovery with --log_error='/var/lib/mysql/wsrep_recovery.Uh8F'
WSREP: Recovered position e5a514d5-80f2-11e6-8163-ea69a6a66f6e:558758
mysqld from pid file /var/lib/mysql/pxc1.pid ended
```
When putting in production unprepared...

**Certification Errors**
Certification

What it does:

- Determine if writeset can be applied.
  - Based on unapplied earlier transactions on master
  - Such conflicts must come from other nodes
- Happens on every node, individually
- Deterministic
- Results are not reported to other nodes in the cluster, as every node does certification and is a deterministic process.
  - Pass: enter apply queue (commit success on master)
  - Fail: drop transaction (or return deadlock on master)
- Serialized by GTID
- Cost based on # of keys or # of rows
Certification

node 1

statement (DML)

node 2

node 3
Certification

node 1

statement (DML)

replicate

node 2

node 3
Certification

node 1

node 2

node 3

statement (DML)

replicate

certify
Certification
Certification
Certification
Conflict Detection

- **Local Certification Failure (lcf)**
  - Transaction fails certification
  - Post-replication
  - Deadlock/Transaction Rollback
  - Status Counter: `wsrep_local_cert_failures`
- **Brute Force Abort (bfa)**
  - (Most Common)
  - Deadlock/Transaction rolled back by applier threads
  - Pre-commit
  - Transaction Rollback
  - Status Counter: `wsrep_local_bf_aborts`
Conflict Deadlock/Rollback

Note: Transaction Rollback can be gotten on any statement, including SELECT and COMMIT

Example:

```
pxc1 mysql> commit;
ERROR 1213 (40001): WSREP detected deadlock/conflict and aborted the
transaction. Try restarting the transaction
```
Multi-writer Conflict Types

Brute Force Abort (bfa)

- Transaction rolled back by applier threads
- Pre-commit
- Transaction Rollback can be gotten on any statement, including `SELECT` and `COMMIT`
- Status Counter: `wsrep_local_bf_aborts`
Brute Force Abort (bfa)
Brute ForceAbort (bfa)
Brute Force Abort (bfa)
Brute Force Abort (bfa)
Brute Force Abort (bfa)
Brute Force Abort (bfa)
Brute Force Abort (bfa)
Multi-writer Conflict Types

Local Certification Failure (lcf)

- Transaction fails certification
- Post-replication
- Deadlock on commit
- Status Counter: `wsrep_local_cert_failures`
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Local Certification Failure (lcf)
Reproducing Conflict

On pxc1, create test table:

```sql
pxc1 mysql> CREATE TABLE test.deadlocks (  
  i INT UNSIGNED NOT NULL PRIMARY KEY,  
  j varchar(32),  
  t datetime  
);  
pxc1 mysql> INSERT INTO test.deadlocks VALUES (1, NULL, NULL);
```

Run `myq_status` on pxc1:

```
# myq_status wsrep
mycluster / pxc1 (idx: 1) / Galera 3.17(r447d194)
<table>
<thead>
<tr>
<th>Wsrep</th>
<th>Cluster</th>
<th>Node</th>
<th>Repl</th>
<th>Queue</th>
<th>Ops</th>
<th>Bytes</th>
<th>Conflict</th>
<th>Gcache</th>
<th>Window</th>
<th>Confict</th>
<th>Lcf</th>
<th>bfa</th>
<th>Ist</th>
<th>Idx</th>
<th>Dst</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06:47:51 P 5 3 Sync 2.6ms 0 0 30 0 50K 0.0 0 0 19k 171 19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>06:47:52 P 5 3 Sync 2.6ms 0 0 21 1 35K 8.0b 0 0 19k 120 19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Reproducing Conflict

On pxc1:

```
pxc1 mysql> BEGIN;
pxc1 mysql> UPDATE test.deadlocks SET j='pxc1', t=now() WHERE i=1;
```

Before commit, go to pxc3:

```
pxc3 mysql> BEGIN;
pxc3 mysql> UPDATE test.deadlocks SET j='pxc3', t=now() WHERE i=1;
pxc3 mysql> COMMIT;
```

Now commit the transaction on pxc1:

```
pxc1 mysql> COMMIT;
pxc1 mysql> SELECT * FROM test.deadlocks;
```
Reproducing Conflict

It fails:

```sql
pxc1 mysql> commit;
ERROR 1213 (40001): WSREP detected deadlock/conflict and aborted the
transaction. Try restarting the transaction
```
Reproducing Conflict

It fails:

```sql
pxc1 mysql> commit;
ERROR 1213 (40001): WSREP detected deadlock/conflict and aborted the transaction. Try restarting the transaction
```

- Which commit succeeded?
- Is this a lcf or a bfa?
- How would you diagnose this error?
Reproducing Conflict

- Which commit succeeded? PXC3, first one that got in cluster.
- Is this a lcf or a bfa? **BFA**
- How would you diagnose this error?

```
# myq_status wsrep
mycluster / pxc1 (idx: 1) / Galera 3.17(r447d194)
Wsrep Cluster Node Repl Queue Ops Bytes Conflict Gcache Window
    time  P  cnf  # Stat Laten Up Dn Dn Up Dn lcf bfa
06:48:10 P  5  3  Sync 2.4ms 0 0  15  0  25K  0.0 0  0  19k 179 19
06:48:11 P  5  3  Sync 2.4ms 0 0  15  1  25K  0.7K 0  1  19k 193 19
06:48:12 P  5  3  Sync 2.4ms 0 0  19  1  32K  8.0b 0  0  19k 126 19

show global status like 'wsrep_local_bf%';
show global status like 'wsrep_local_cert%';
```

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsrep_local_bf_aborts</td>
<td>1</td>
</tr>
<tr>
<td>wsrep_local_cert_failures</td>
<td>0</td>
</tr>
</tbody>
</table>
Reproducing Conflict

Log Conflicts

pxc1 mysql> set global wsrep_log_conflicts=on;

** Priority TRANSACTION:
 TRANSACTION 839659, ACTIVE 0 sec starting index read
 mysql tables in use 1, locked 1
 1 lock struct(s), heap size 1136, 0 row lock(s)
 MySQL thread id 2, OS thread handle 13980311456512, query id 648862 System lock

** Victim TRANSACTION:
 TRANSACTION 838894, ACTIVE 18 sec
 2 lock struct(s), heap size 1136, 1 row lock(s), undo log entries 1
 MySQL thread id 26212, OS thread handle 139889637787392, query id 640599 localhost read

** WAITING FOR THIS LOCK TO BE GRANTED:
 RECORD LOCKS space id 26 page no 5 n bits 144 index PRIMARY of table sbtest.sbtetest1
 2016-09-29T06:57:06.854057Z 2 [Note] WSREP: cluster conflict due to high priority abort
 2016-09-29T06:57:06.854065Z 2 [Note] WSREP: Winning thread:
       THD: 2, mode: applier, state: executing, conflict: no conflict, seqno: 415550
       SQL: (null)
 2016-09-29T06:57:06.854068Z 2 [Note] WSREP: Victim thread:
       THD: 26212, mode: local, state: idle, conflict: no conflict, seqno: -1
       SQL: (null)
Reproducing Conflicts

Summary

Conflicts are of a concern when determining if PXC is a fit for the application:

- Long running transactions increase chance of conflicts
- Heavy write workload on multiple nodes
- Large transactions increase chance of conflicts

These issues can usually be resolved by writing to 1 node only.
Hey! Wait for me!

Flow Control
Flow Control

Avoiding nodes drift behind (slave lag?)

- Any node in the cluster can ask the other nodes to pause writes if it lags behind too much.
- Caused by `wsrep_local_recv_queue` exceeding a node’s `gcs.fc_limit`
- Can cause all writes on all nodes in the entire cluster to be stalled.
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control

Diagram:
- Node 1
- Node 2
- Node 3
- Commit
- Synchronous replication
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control
Flow Control

Status Counters

- `wsrep_flow_control_paused_ns`:
  nanoseconds since starts of node did the cluster get **stalled**.

- `wsrep_flow_control_recv`:
  Amount of flow control messages **received** from other nodes

- `wsrep_flow_control_sent`:
  Amount of flow control messages **sent** to other nodes
Flow Control

Observing Flow Control

- Run the application (`run_app.sh`)
- Run `myq_status wsrep` on all nodes.
- LOCK TABLE `sbttest.sbttest1` on `pxc3` and observe its effect on the cluster.
Flow Control

Observing Flow Control

- Run the application (`run_app.sh`)
- Run `myq_status wsrep` on all nodes.
- `LOCK TABLE sbtest.sbstest1` on `pxc3` and observe its effect on the cluster.

pxc1

- `run_app.sh`
- `myq_status wsrep`

all nodes:

pxc3

- `set global pxc_strict_mode = 'DISABLED'`
- `lock tables sbtest.sbstest1 write`
- `wait until flow control kicks in...`
- `unlock tables`
- `set global pxc_strict_mode = 'ENFORCING'`
Flow Control

- Any node in the cluster can ask the other nodes to pause writes if it lags behind too much.
- A node can lag behind more before sending flow control messages, controlled via gcs.fc_limit.
- This can be controlled per node.

Is there another alternative?
Flow Control

DESYNC mode

- It's possible to let a node going behind the flow control limit.
- This can be performed by setting `wsrep_desync=ON`
Flow Control

DESYNC mode

- It's possible to let a node going behind the flow control limit.
- This can be performed by setting `wsrep_desync=ON`

```
pxc3 mysql> set global wsrep_desync=on;
```
What's new in PXC 5.7?

- Save your workload from experimental features:
  - Introducing pxc-strict-mode.

- Extended support for PFS.
  - Now monitor Galera and wsrep instrument through PFS.

- Support for encrypted tablespaces in Multi-Master Topology.
  - PXC can wire encrypted tablespace to new booting node.

- Proxy-SQL compatible PXC
  - PXC is ProxySQL compatible with new single step configuration.
What's new in PXC 5.7?

- PMM enabled monitoring for PXC
  - Effectively monitor PXC using PMM.

- MySQL/PS-5.7.14 and galera-3.17 compatibility
  - Bug fixes, Improved logging and lot more.

- Simplified packaging for PXC
  - PXC packages now takes care of galera installation too.

- Support to use latest Xtrabackup with enhanced security checks.
  - Feel power of improved and latest XB with PXC.
Why pxc_strict_mode?

PXC/Galera limitations:

- Supports only Transactional Storage Engine.
- Explicit locks don’t fit in multi-master topology.
- Need of primary key for certification data generation.
- XA-transaction are not supported.
Understanding `pxc_strict_mode`

- **ENFORCING (DEFAULT/RECOMMENDED)**
  - Use of experimental features raises error. (during startup server refuse to start and runtime operation is blocked. error is logged)

- **DISABLED**
  - PXC-5.6 compatible. Allows experimental feature. No error. No warning

- **PERMISSIVE**
  - Use of experimental features results in a warning at startup and runtime. Server continues to accept the setting and operate.

- **MASTER**
  - Same as ENFORCING for all experimental features except explicit-table-locking validation checks are not performed under this mode.
Understanding `pxc_strict_mode`

- `pxc_strict_mode` is local to given node and if user plan to toggle it, it should be done on all the nodes for cluster consistency and correctness.

- Toggling `pxc_strict_mode` from less strict mode to more strict mode
  - For example: DISABLED -> ENFORCING .... need to ensure ENFORCING characteristics are met.
  - Like
    - `wsrep_replicate_myisam=OFF`
    - `binlog_format=ROW`
    - `log_output=FILE/NONE`
    - `tx_isolation=SERIALIZABLE`.
pxc_strict_mode blocks

- Semantics blocked:
  - DML and DDL operations (except CREATE/DROP) are not permitted on non-transactional Storage Engine.
  - Table can be converted from non-transactional SE to transactional SE using ALTER.
  - Trying to enable MyISAM replication is blocked.
  - binlog-format has to be ROW.
pxc_strict_mode blocks

- Semantics blocked:
  - DML to tables without primary-key is not allowed.
  - log-output has to be directed to FILE or DISABLED (NONE).
  - Explicit TABLE locking feature (LOCK table, GET_LOCK, FLUSH TABLE WITH READ LOCK, Setting SERIALIZABLE transaction level) is blocked.
  - auto-increment mode has to be INTERLEAVED.
  - Combining Schema and DML changes in single statement like CTAS is not permitted.
pxc\_strict\_mode hands on

Let's try some simple command with pxc-strict-mode.

```sql
pxc3 mysql> select @@pxc\_strict\_mode;
+-------------------------+
<table>
<thead>
<tr>
<th>@@pxc_strict_mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENFORCING</td>
</tr>
</tbody>
</table>
+-------------------------+

pxc3 mysql> create table t1 (i int, primary key pk(i)) engine=myisam;
pxc3 mysql> insert into t1 values (1);
ERROR 1105 (HY000): Percona-XtraDB-Cluster prohibits use of DML command on a table (test.t1) that resides in non-transactional storage engine with pxc\_strict\_mode = ENFORCING or MASTER

pxc3 mysql> set wsrep_replicate_myisam=1;
ERROR 1105 (HY000): Percona-XtraDB-Cluster prohibits use of MyISAM table replication feature with pxc\_strict\_mode = ENFORCING or MASTER

pxc3 mysql> alter table t1 engine=innodb;
pxc3 mysql> insert into t1 values (1);
pxc3 mysql> lock table t1 read;
ERROR 1105 (HY000): Percona-XtraDB-Cluster prohibits use of LOCK TABLE/FLUSH TABLE <table> WITH READ LOCK with pxc\_strict\_mode = ENFORCING or MASTER
pxc3 mysql> drop table t1;
```
Let's have a small test now?

```
pxc3 mysql> set global pxc_strict_mode='DISABLED';
pxc3 mysql> create table myisam (i int, primary key pk(i)) engine=myisam;
pxc3 mysql> set global wsrep_replicate_myisam=1;
pxc3 mysql> insert into myisam values (1), (2), (3);
pxc3 mysql> alter table myisam add column j int;
pxc3 mysql> update myisam set j = 100;
pxc3 mysql> select * from myisam;
pxc3 mysql> drop table myisam;
```

pxc3 mysql> set global pxc_strict_mode='ENFORCING';

Will it PASS? FAIL? WARNING?
**Restore pxc\_strict\_mode**

Remember to restore the mode back to ENFORCING before we proceed with next set of experiment.

```bash
pxc3 mysql> set global wsrep_replicate_myisam=0;
pxc3 mysql> set global pxc_strict_mode='ENFORCING';
pxc3 mysql> select @@pxc_strict_mode;
```
Monitoring PXC through Performance Schema

- Traditional method to monitor PXC or MySQL through log file is time consuming and may need special tool even to detect occurrence of an event.
- Performance Schema is effective way and has become de facto standard for monitoring different elements of MySQL.
- Till 5.6 PXC had limited support for performance_schema where-in only wsrep related instruments were exposed through performance schema that too in limited fashion.
- Starting PXC-5.7 we have taken big-step further enabling monitoring of galera instruments and other wsrep-instruments as part of performance schema.
Monitoring PXC through Performance Schema

- Instruments that are monitored:
  - THREADS: applier, rollback, service_thd, gcomm conn, receiver, sst/ist threads, etc...
  - LOCK/COND_VARIABLES: from wsrep and galera library.
  - STAGES: Different stage threads are passing through.

With this information, user should able to track some of the most important instruments that can help get some insight as to where server is really spending time or out-of-path flow like rollback of transactions.
Experimenting P_S threads

Let's see what all different threads are active:

```sql
pxc3 mysql> select thread_id, name from performance_schema.threads
where name like '%galera%' or name like '%wsrep';
+-+---------------------------------------------------+
| 3 | thread/galera/THREAD_galera_service_thd    |
| 4 | thread/galera/THREAD_galera_gcommconn      |
| 5 | thread/galera/THREAD_galera_receiver      |
| 6 | thread/sql/THREAD_wsrep_applier            |
| 7 | thread/sql/THREAD_wsrep_rollbacker         |
+-+---------------------------------------------------+
5 rows in set (0.03 sec)
```

```sql
pxc3 mysql> set global wsrep_slave_threads=10;
pxc3 mysql> select count(*) from performance_schema.threads
where name like '%THREAD_wsrep_applier';
```

```sql
pxc3 mysql> set global wsrep_slave_threads=1;
pxc3 mysql> select count(*) from performance_schema.threads
where name like '%THREAD_wsrep_applier';
```

pxc1# /root/bin/run_app.sh [Stop app after 2 seconds]

```sql
pxc3 mysql> select count(*) from performance_schema.threads
where name like '%THREAD_wsrep_applier';
```
Experimenting P_S files

- Let's now checkout files

```sql
pxc3 mysql> select * from performance_schema.file_instances
where file_name like '%galera%';
+---------------------+-----------------+-------------+
| FILE_NAME           | EVENT_NAME       | OPEN_COUNT  |
| /var/lib/mysql/galera.cache | wait/io/file/galera/FILE_galera_ringbuffer |           1 |
+---------------------+-----------------+-------------+
```

1 row in set (0.02 sec)

```sql
pxc3 mysql> select * from performance_schema.file_instances
where file_name like '%gcache%';
+---------------------+-----------------+-------------+
| FILE_NAME            | EVENT_NAME       | OPEN_COUNT  |
| /var/lib/mysql/dn1/gcache.page.000000 | wait/io/file/galera/FILE_galera_gcache_page |           1 |
+---------------------+-----------------+-------------+
```
Experimenting P_S files

- Some other important points related to files:
  - Galera uses mmap for most of these files and so we avoid updating read/write stats for file available through file_summary_by_instance/file_summary_by_event_name.*
  - Only dynamically changing caching write-set are cached. grastate/gvwstate files are not tracked.
Experimenting P_S mutex/cond_var

Enable instruments:

```
pxc3 mysql> use performance_schema;
pxc3 mysql> update setup_consumers set enabled='YES'
         where name like '%events_waits%';
pxc3 mysql> update setup_consumers set enabled='YES'
         where name like '%events_stages%';
pxc3 mysql> update setup_instruments set enabled='YES', timed='YES'
         where name like '%wsrep%' or name like '%galera%';
```

```
pxc3 mysql> use test; create table t (i int, primary key pk(i)) engine=innodb;
pxc3 mysql> begin; insert into t values (1);
pxc3 mysql> use test; begin; insert into t values (1); commit;
pxc3 mysql> commit;
```

```
pxc3 mysql> select thread_id, EVENT_NAME, TIMER_WAIT from
              performance_schema.events_waits_history where event_name like '%rollback%';
pxc3 mysql> select thread_id, EVENT_NAME, TIMER_WAIT from
              performance_schema.events_waits_current where event_name like '%rollback%';
```

```
pxc3 mysql> select thread_id, event_id, EVENT_NAME, TIMER_WAIT from
              performance_schema.events_waits_history where event_name like '%monitor%';
(Check the apply and commit monitor being invoked by applier thread)
```
Experimenting P_S stages

Find out precisely what the threads are doing

```
pxc3 mysql> select * from performance_schema.events_stages_current;
[What are thread currently executing.]

pxc3 mysql> select * from performance_schema.events_stages_history;
[What action was replicated earlier. UPDATE, INSERT, DELETE. Was it finally committed]
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>stage/wsrep/wsrep: aborter active</th>
<th>wsrep</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>10</td>
<td>18</td>
<td>stage/wsrep/wsrep: applying write set</td>
<td>wsrep</td>
</tr>
<tr>
<td>7</td>
<td>152</td>
<td>154</td>
<td>stage/wsrep/wsrep: writing rows</td>
<td>log_e</td>
</tr>
<tr>
<td>7</td>
<td>156</td>
<td>164</td>
<td>stage/wsrep/wsrep: applying idle</td>
<td>wsrep</td>
</tr>
<tr>
<td>7</td>
<td>166</td>
<td>167</td>
<td>stage/wsrep/wsrep: committing</td>
<td>wsrep</td>
</tr>
<tr>
<td>7</td>
<td>168</td>
<td>171</td>
<td>stage/wsrep/wsrep: committing</td>
<td>wsrep</td>
</tr>
</tbody>
</table>

```
pxc3 mysql> use performance_schema;
pxc3 mysql> update setup_consumers set enabled='NO'
where name like '%events_waits%';
pxc3 mysql> update setup_consumers set enabled='NO'
where name like '%events_stages%';
pxc3 mysql> update setup_instruments set enabled='NO', timed='NO'
where name like '%wsrep%' or name like '%galera%';
```
Securing PXC

There are 2 main components to secure.

- Traffic (SST/IST inter-node traffic). [5.6 feature]
- Tablespace. [5.7 feature]

Securing inter-node traffic is achieved using SSL and support is present since PXC-5.6.

- For SST using xtrabackup-v2 or mysqldump can help achieve this. Recommended approach is xtrabackup due to added flexibility in locking database while servicing SST.
- For IST galera in-build SSL support ensure that replication channel is secured.
Securing PXC

- PXC-5.7 extended support for MySQL introduced encrypted tablespace. This means using XB/mysqldump these encrypted tablespaces will be moved to new booting node and re-encrypted with new node configuration making them auto-available to end-user in secured fashion.

This help completes the loop of ensuring everything is secured in PXC.

--early-plugin-load
[keyring_file_data]
Spread the load

**Load Balancers**
Load Balancers

With PXC a Load Balancer is commonly used:

- Lot's of choice
- HAProxy (most-common)
- **ProxySQL**
- MariaDB MaxScale
- ScaleArc (proprietary)
- MySQL Router
Load Balancers

ProxySQL

- High-performance Proxy for MySQL and forks
- Query routing and read/write split
- On-the-fly rewrite of queries
- Caching reads outside the database server
- Supports failover
- High Availability
- Runtime reconfiguration
Load Balancers

ProxySQL

- All backends are grouped into hostgroups
- Hostgroups have logical functionalities
Load Balancers

ProxySQL

- All backends are grouped into hostgroups
- Hostgroups have logical functionalities

HG10: Write masters
Load Balancers

ProxySQL

- All backends are grouped into hostgroups
- Hostgroups have logical functionalities

HG10: Write masters
HF11: Read Slaves
Load Balancers

ProxySQL

- All backends are grouped into hostgroups
- Hostgroups have logical functionalities
  
  HG10: Write masters
  HF11: Read Slaves

- Read/Write Split
Load Balancers
ProxySQL - Scheduler

- Scheduler is a cron-like solution embedded directly inside ProxySQL
- Can run any external scripts
- Support for Percona XtraDB Cluster
Load Balancers

ProxySQL Admin

- Tool for configuring Percona XtraDB Cluster nodes with ProxySQL
- `--enable` option to automatically configure a PXC into ProxySQL
- Add Percona XtraDB Cluster node into the ProxySQL database
- Add the monitoring scripts into the ProxySQL scheduler table
- Create two new users, one user for monitoring, other one for communicating with the cluster
- `--disable` option to remove PXC configuration from ProxySQL
Load Balancers

ProxySQL
Load Balancers

ProxySQL Admin

```bash
mysql --host=127.0.0.1 --user=admin --password=admin --port=6032
...
Your MySQL connection id is 20795
Server version: 5.5.30 (ProxySQL Admin Module)
```

ProxySQL Admin(proxysql-admin) to configure PXC nodes into ProxySQL.

```bash
/bin/proxysql-admin --proxysql-user=admin --proxysql-password=admin \
--proxysql-port=6032 --proxysql-host=127.0.0.1 \
--cluster-user=test --cluster-password=test --cluster-port=3306 \
--cluster-host=127.0.0.1 \
--galera-check-interval=1000 --enable --mode='loadbal' --adduser
```
Load Balancers

ProxySQL - Setup

```
pxc1 mysql> select hostgroup_id, hostname, port, status from mysql_servers;
+-----------------+-----------------+-----------------+---------+
| hostgroup_id | hostname | port | status |
+-----------------+-----------------+-----------------+---------+
| 10             | pxc1        | 3306 | ONLINE |
| 10             | pxc2        | 3306 | ONLINE |
| 10             | pxc3        | 3306 | ONLINE |
+-----------------+-----------------+-----------------+---------+
3 rows in set (0.00 sec)
```

```
pxc1 mysql> select username, password, active, default_hostgroup from mysql_users;
+-----------------+----------------+----------------+-------------+
| username | password      | active | default_hostgroup |
+-----------------+----------------+----------------+-------------+
| test        | *94BDCCEBE19083CE2A1F95... | 1     | 10          |
+-----------------+----------------+----------------+-------------+
1 row in set (0.01 sec)
```
Load Balancers

ProxySQL - Setup

Usage: /bin/proxysql_galera_checker <hostgroup_id write> [hostgroup_id read] [number writers] [writers are readers 0|1] [log_file]

pxc1 mysql> select id, active, interval_ms, filename, arg1, arg2, arg3, arg4, arg5 from scheduler

* * * * * * * * * * * * * * * * * * * * * * * * * *
1. row * * * * * * * * * * * * * * * * * * * * * * * * * *
id: 10
active: 1
interval_ms: 1000
filename: /bin/proxysql_galera_checker
arg1: 10
arg2: 10
arg3: 3
arg4: 1
arg5: /var/lib/proxysql/proxysql_galera_check.log

* * * * * * * * * * * * * * * * * * * * * * * * * *
2. row * * * * * * * * * * * * * * * * * * * * * * * * * *
id: 11
active: 1
interval_ms: 5000
filename: /bin/proxysql_node_monitor
arg1: 10
arg2: /var/lib/proxysql/proxysql_node_monitor.log
...

Load Balancers
ProxySQL - Load balancing

ProxySQL is configured to listen on port 6033 for applications

```bash
while true;
do
  mysql -h pxc1 -P 6033 -u test -ptest -e "select @@wsrep_node_name;" 2>/dev/null;
sleep 1;
done
```

```
+-----------------------------+-----------------------------+
| @@wsrep_node_name           | pxc1                        |
+-----------------------------+-----------------------------+
| @@wsrep_node_name           | pxc2                        |
+-----------------------------+-----------------------------+
| @@wsrep_node_name           | pxc3                        |
+-----------------------------+-----------------------------+
```

127
Load Balancers
ProxySQL - R/W Split

ProxySQL
: 6033

192.168.56.2
pxc 1

192.168.56.3
pxc 2

192.168.56.4
pxc 3

Writes
Reads

^Select will go to hostgroup 11
Load Balancers

ProxySQL - Hostgroups

All backends are grouped into hostgroups

[root@pxc1 ~]# mysql -h 127.0.0.1 -P 6032 -uadmin -padmin
Server version: 5.5.30 (ProxySQL Admin Module)

pxc1 mysql> select hostgroup_id, hostname, port, status, weight from mysql_servers;

<table>
<thead>
<tr>
<th>hostgroup_id</th>
<th>hostname</th>
<th>port</th>
<th>status</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>pxc1</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>pxc2</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>pxc3</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc1</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc2</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc3</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
</tbody>
</table>

6 rows in set (0.00 sec)
## Load Balancers

### ProxySQL - R/W split

```sql
pxc1 mysql> select active, match_pattern, destination_hostgroup, apply
from mysql_query_rules;
+----------+-----------------+-----------------+-----+
<table>
<thead>
<tr>
<th>active</th>
<th>match_pattern</th>
<th>destination_hostgroup</th>
<th>apply</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>^SELECT.*</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>^SELECT.*FOR UPDATE</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
+----------+-----------------+-----------------+-----+
2 rows in set (0.00 sec)
```

### Application User:

```sql
pxc1 mysql> select username, password, active, default_hostgroup from mysql_users;
+----------+-------------+-----+-----------------+
<table>
<thead>
<tr>
<th>username</th>
<th>password</th>
<th>active</th>
<th>default_hostgroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>*94BDCBE8190B3CE2A1F959FD...</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
+----------+-------------+-------+-----------------+
1 row in set (0.00 sec)
```
Load Balancers
ProxySQL - Scheduler

Usage: /bin/proxysql_galera_checker
[hostgroup_id write] [hostgroup_id read] [number writers]
[writers are readers 0|1] [log_file]

pxc1 mysql> select id, active, interval_ms, filename, arg1, arg2, arg3,
arg4, arg5 from scheduler\G
*************** 1. row **********************
  id: 10
  active: 1
  interval_ms: 3000
  filename: /bin/proxysql_galera_checker
  arg1: 10
  arg2: 11
  arg3: 1
  arg4: 0
  arg5: /var/lib/proxysql/proxysql_galera_check.log
1 row in set (0.00 sec)
Load Balancers

ProxySQL - Scheduler

pxc1 mysql>
LOAD MYSQL SERVERS TO RUNTIME;
SAVE MYSQL SERVERS TO DISK;
LOAD MYSQL USERS TO RUNTIME;
SAVE MYSQL USERS TO DISK;
LOAD MYSQL QUERY RULES TO RUNTIME;
SAVE MYSQL QUERY RULES TO DISK;
LOAD SCHEDULER TO RUNTIME;
SAVE SCHEDULER TO DISK;
LOAD MYSQL VARIABLES TO RUNTIME;
SAVE MYSQL VARIABLES TO DISK;
Load Balancers

ProxySQL - R/W split

```sql
pxc1 mysql> select hostgroup_id, hostname, port, status, weight from mysql_servers;
+----------------+----------------+-----+---------+-----+
<table>
<thead>
<tr>
<th>hostgroup_id</th>
<th>hostname</th>
<th>port</th>
<th>status</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>pxc1</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>pxc2</td>
<td>3306</td>
<td>OFFLINE_SOFT</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>pxc3</td>
<td>3306</td>
<td>OFFLINE_SOFT</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc1</td>
<td>3306</td>
<td>OFFLINE_SOFT</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc2</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
<tr>
<td>11</td>
<td>pxc3</td>
<td>3306</td>
<td>ONLINE</td>
<td>1000</td>
</tr>
</tbody>
</table>
+----------------+----------------+-----+---------+-----+
6 rows in set (0.00 sec)
```
Load Balancers
ProxySQL - R/W split

ProxySQL is configured to listen on port 6033 for applications

```sql
while true;
do
  mysql -h pxc1 -P 6033 -u test -p test -e "select @@wsrep_node_name;" 2>/dev/null;
sleep 1;
done
```

```
+-----------------+-----------------+
| @@wsrep_node_name | pxc3            |
|-----------------+-----------------+
+-----------------+-----------------+
+-----------------+-----------------+
| @@wsrep_node_name | pxc2            |
|-----------------+-----------------+
+-----------------+-----------------+
```

134
Load Balancers
ProxySQL - R/W split

Let's verify how it balance our writes query:

```sql
while true;
  do
    mysql -h pxc1 -P 6033 -u test -ptest \
    -e "select @@wsrep_node_name,id,k from sbtest.sbtest1 \n    order by rand() limit 1 for update;" 2>/dev/null;
    sleep 1;
  done
```

<table>
<thead>
<tr>
<th>@@wsrep_node_name</th>
<th>id</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxc1</td>
<td>81707</td>
<td>50063</td>
</tr>
<tr>
<td>pxc1</td>
<td>41391</td>
<td>56266</td>
</tr>
</tbody>
</table>
MySQL High Availability with Percona
XtraDB Cluster 5.7

Q&A
Remember to rate our talk on the mobile app!

Peter Zaitsev

Alok Pathak

Krunal Bauskar