PERCONA

TRAINING

Percona XtraDB Cluster Tutorial

http://www.percona.com/training/

Table of Contents

XtraDB Cluster Tutorial

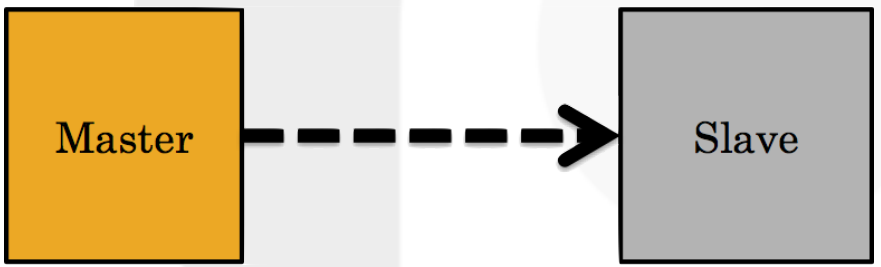
OVERVIEW

Resources

- http://www.percona.com/doc/percona-xtradb-cluster/5.7/
- http://www.percona.com/blog/category/percona-xtradb-cluster/
- http://galeracluster.com/documentation-webpages/reference.html

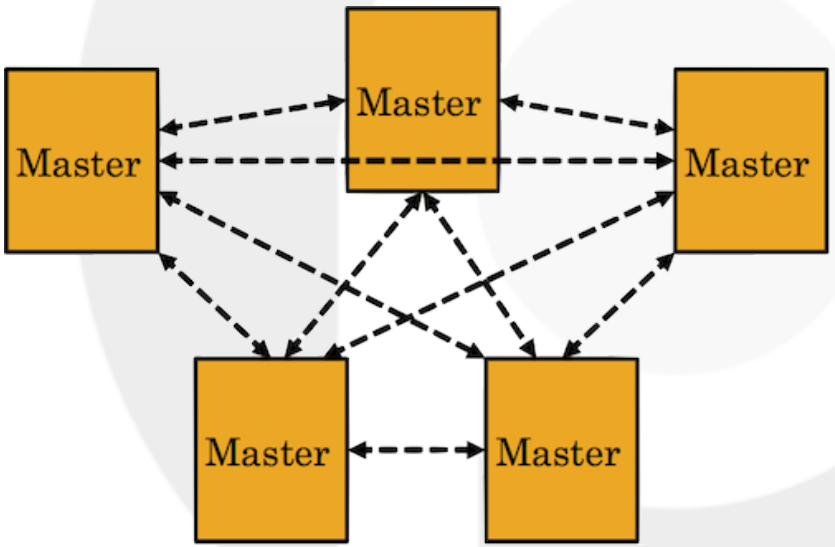
Standard Replication

- Replication is a mechanism for recording a series of changes on one MySQL server and applying the same changes to one, or more, other MySQL server.
- The source is the "master" and its replica is a "slave."



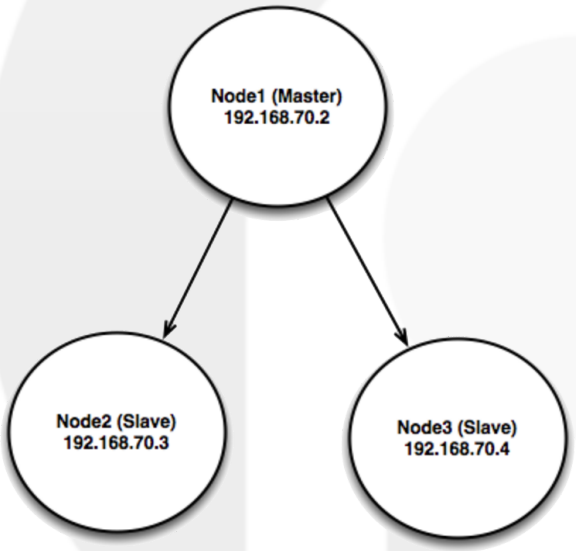
Galera Replication

- Developed by Codership
- Synchronous multi-master database cluster
- Write to any node; every node stays in sync.



Tutorial Setup

- Single master node; Two slave nodes
- node1 will also run `sysbench`
  - Sample application, simulating activity



(*) Your node IP addresses will probably be different.

Verify Connectivity and Setup

- Connect to all 3 nodes
  - Use separate terminal windows; we will be switching between them frequently.
  - Username and password is **vagrant**
  - DO NOT USE VIRTUAL BOX CONSOLE WINDOWS!

```
ssh -p 22201 vagrant@localhost
ssh -p 22202 vagrant@localhost
ssh -p 22203 vagrant@localhost
```

- Stop mysql on all 3 nodes (make sure you are root)

```
node1# systemctl stop mysql
node2# systemctl stop mysql
node3# systemctl stop mysql
```

Configure Node3 for PXC

```
[mysqld]
# Leave existing settings and add these

binlog_format                   = ROW

# galera settings
wsrep_provider                  = /usr/lib64/libgalera_smm.so
wsrep_cluster_name              = mycluster
wsrep_cluster_address           = gcomm://node1,node2,node3
wsrep_node_name                 = node3
wsrep_node_address              = 192.168.70.4
wsrep_sst_auth                  = sst:secret

innodb_autoinc_lock_mode        = 2
```

- Start node3 with:

```
node3# systemctl start mysql@bootstrap
```

Checking Cluster State

- Can you tell:
  - How many nodes are in the cluster?
  - Is the cluster Primary?

Checking Cluster State

- Use 'myq_status' to check the state of Galera on node3:

```
node3# /usr/local/bin/myq_status wsrep
```

node2's Config

```
[mysqld]
# Leave existing settings and add these

binlog_format                   = ROW
log-slave-updates

# galera settings
wsrep_provider                  = /usr/lib64/libgalera_smm.so
wsrep_cluster_name              = mycluster
wsrep_cluster_address           = gcomm://node1,node2,node3
wsrep_node_name                 = node2         # <-- !!! CHANGE !!!
wsrep_node_address              = 192.168.70.3  # <-- !!! CHANGE !!!
wsrep_sst_auth                  = sst:secret

innodb_autoinc_lock_mode        = 2
```

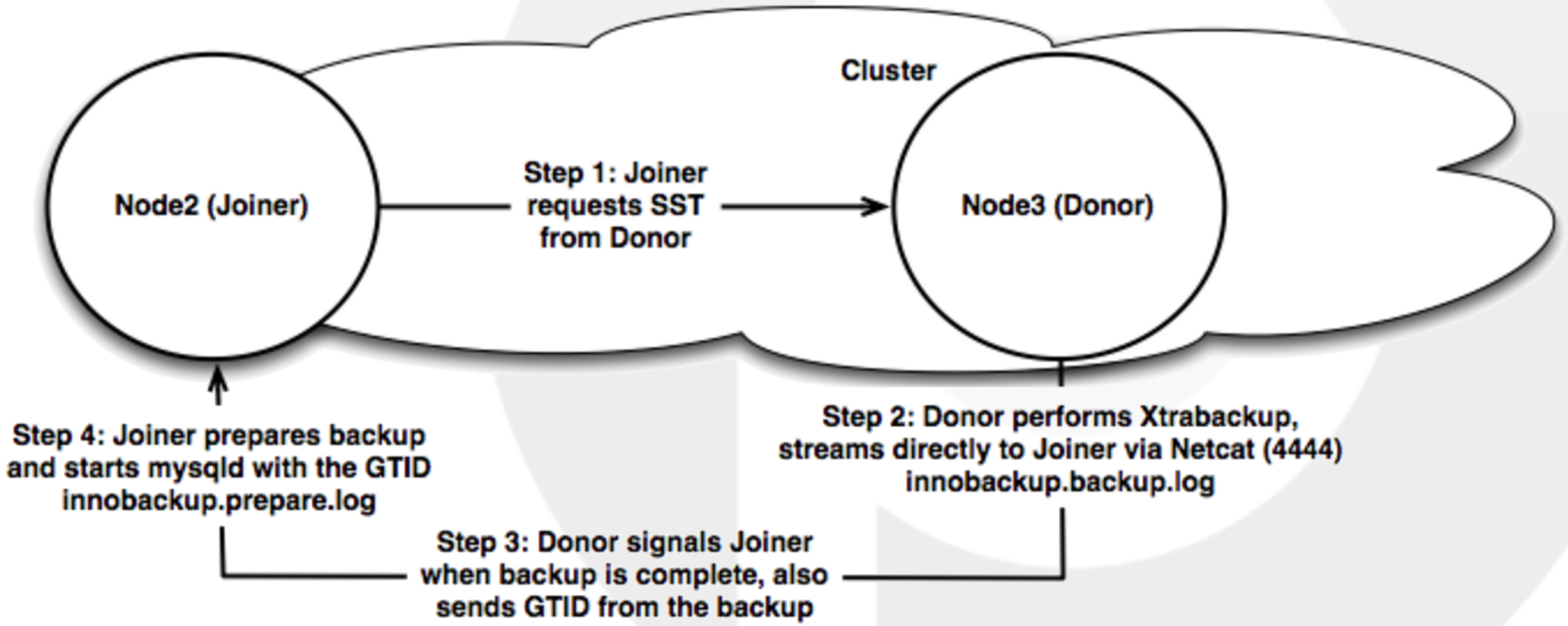
- Don't start MySQL yet!

What's going to happen?

- Don't start MySQL yet!
- When node2 is started, what's going to happen?
- How does node2 get a copy of the dataset?
  - Can it use the existing data it already has?
- Do we have to bootstrap node2?

State Snapshot Transfer (SST)

- Transfer a full backup of an existing cluster member (donor) to a new node entering the cluster (joiner).
- We configured our SST method to use ‘xtrabackup-v2’.



Additional Xtrabackup SST Setup

- What about that sst user in your my.cnf?

```
[mysqld]
...
wsrep_sst_auth = sst:secret
...
```

- User/Grants need to be added for Xtrabackup

```
node3> CREATE USER 'sst'@'localhost' IDENTIFIED BY 'secret';
node3> GRANT PROCESS, RELOAD, LOCK TABLES, REPLICATION CLIENT
ON *.* TO 'sst'@'localhost';
```

- The good news is that once you get things figured out the first time, it's typically very easy to get an SST the first time on subsequent nodes.

Up and Running node2

- Now you can start node2

```
node2# systemctl start mysql
```

- Watch node2's error log for progress

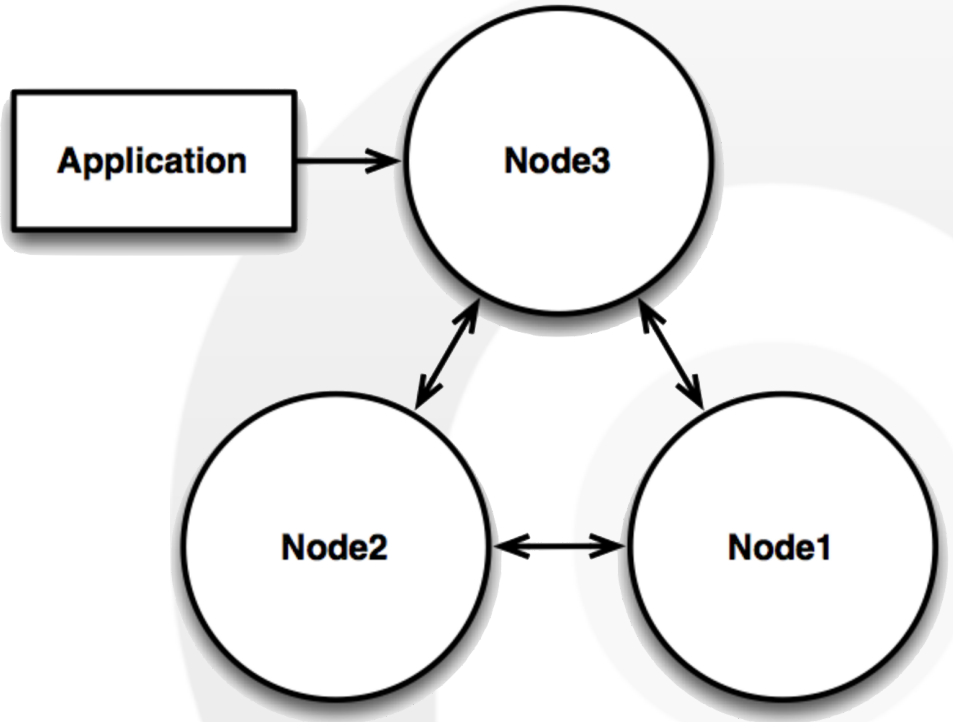
```
node2# tail -f /var/lib/mysql/error.log
```

node1's Config

```
[mysqld]
# Leave existing settings and add these

binlog_format                   = ROW
log-slave-updates

# galera settings
wsrep_provider                  = /usr/lib64/libgalera_smm.so
wsrep_cluster_name              = mycluster
wsrep_cluster_address           = gcomm://node1,node2,node3
wsrep_node_name                 = node1         # <-- !!! CHANGE !!!
wsrep_node_address              = 192.168.70.2  # <-- !!! CHANGE !!!
wsrep_sst_auth                  = sst:secret

innodb_autoinc_lock_mode        = 2
```

```
node1# systemctl start mysql
```

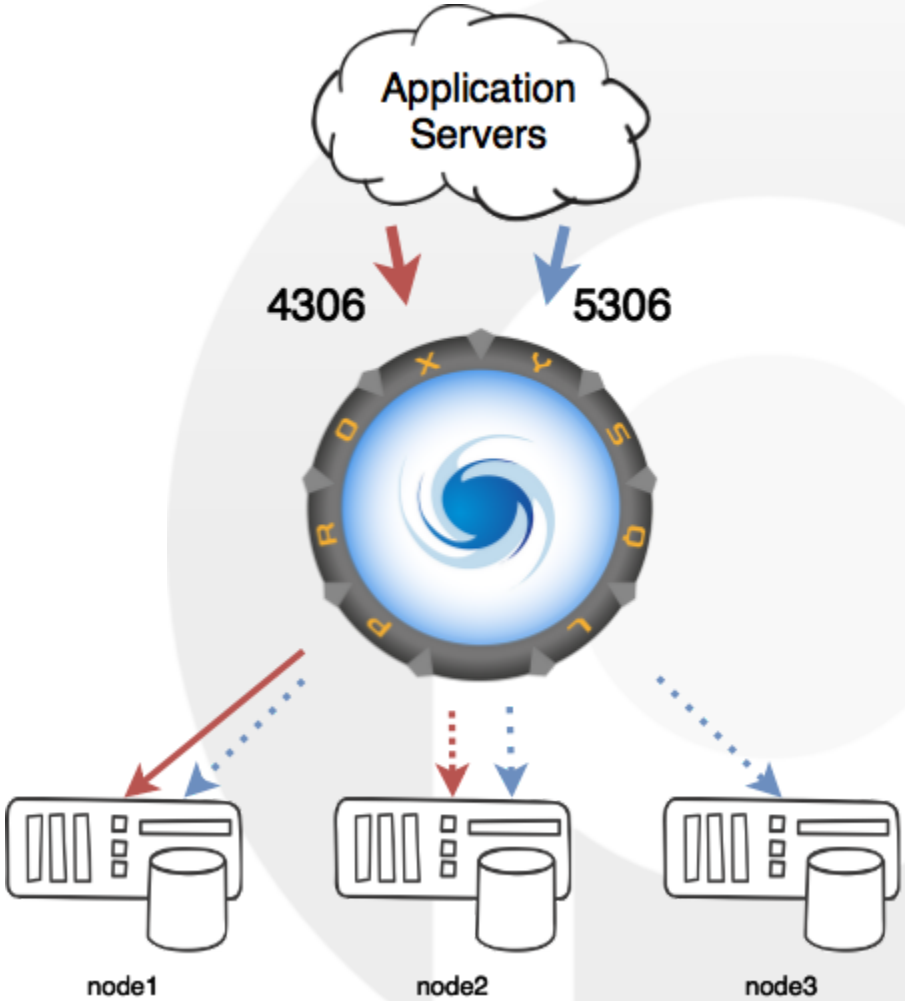
Where are We Now?



XtraDB Cluster Tutorial

APPLICATION HIGH AVAILABILITY

ProxySQL Architecture



Setting up ProxySQL

- Create monitor user

```
node1 mysql> CREATE USER 'monitor'@'%' IDENTIFIED BY 'monit0r';
```

- Create Cluster admin user

```
node3> CREATE USER 'cuser'@'%' IDENTIFIED BY 'cpass';
node3> GRANT ALL ON *.* TO 'cuser'@'%';
```

Configuring of ProxySQL

- Create and edit the file

```
node1# nano /etc/proxysql-admin.cnf

export PROXYSQL_USERNAME="admin"
export PROXYSQL_PASSWORD="admin"
export PROXYSQL_HOSTNAME="localhost"
export PROXYSQL_PORT="6032"
export CLUSTER_USERNAME="cuser"
export CLUSTER_PASSWORD="cpass"
export CLUSTER_HOSTNAME="localhost"
export CLUSTER_PORT="3306"
export MONITOR_USERNAME="monitor"
export MONITOR_PASSWORD="monit0r"
export CLUSTER_APP_USERNAME="sbuser"
export CLUSTER_APP_PASSWORD="sbpass"
export WRITE_HOSTGROUP_ID="10"
export READ_HOSTGROUP_ID="20"
export MODE="singlewrite"
export HOST_PRIORITY_FILE="/var/lib/proxysql/host_priority.conf"
```

Start ProxySQL

- Star ProxySQL with the following command:

```
node1# systemctl restart proxysql
node1# proxysql-admin --config-file=/etc/proxysql-admin.cnf \
--write-node=192.168.70.4:3306 --enable
```

- Node3 is the writer node
- Grant all privileges to app user

```
node3> GRANT ALL ON *.* TO 'sbuser'@'192.%';
```

- Check ProxySQL Stats

```
node1# mysql -P6032 -uadmin -padmin -h 127.0.0.1

proxysql> select * from stats_mysql_connection_pool \
order by hostgroup,srv_host desc;

+-----------+--------------+----------+--------+...------------+
| hostgroup | srv_host     | srv_port | status |... Latency_us |
+-----------+--------------+----------+--------+...------------+
| 10        | 192.168.70.4 | 3306     | ONLINE |... 461        |
| 20        | 192.168.70.3 | 3306     | ONLINE |... 820        |
| 20        | 192.168.70.2 | 3306     | ONLINE |... 101        |
+-----------+--------------+----------+--------+...------------+
```

Start Sysbench

- Sending Reads and Writes
  - Writes goes to Hostgroup 10 (node3)
  - Reads goes to Hostgroup 20 (node1,node2)

```
node1# /usr/local/bin/run_sysbench_oltp_proxy.sh

[   1s] threads: 1, tps: 9.99, reads: 139.89, writes: 39.97, response tir
[   2s] threads: 1, tps: 7.00, reads: 98.04, writes: 28.01, response time
[   3s] threads: 1, tps: 7.00, reads: 98.03, writes: 28.01, response time
[   4s] threads: 1, tps: 14.00, reads: 195.93, writes: 55.98, response ti
```

- Check ProxySQL Stats

```
node1# mysql -P6032 -uadmin -padmin -h 127.0.0.1

proxysql> select * from stats_mysql_connection_pool \
order by hostgroup,srv_host desc;
```

Test node2 Shutdown

- Start myq_status on node3
  - `/usr/local/bin/myq_status wsrep`
- Shut down node2
  - `systemctl stop mysql`
  - Check ProxySQL Stats

```
node2# systemctl stop mysql
proxysql> select * from stats_mysql_connection_pool \
order by hostgroup,srv_host desc;
+-----------+--------------+----------+--------..
| hostgroup | srv_host     | srv_port | status ..
+-----------+--------------+----------+--------..
| 10        | 192.168.70.4 | 3306     | ONLINE ..
| 20        | 192.168.70.2 | 3306     | ONLINE ..
+-----------+--------------+----------+--------..
```

- Start node2

```
node2# systemctl restart mysql
```

Crash Test

- Stop `node1` destructively: `kill -9`
  - Make sure to kill `mysqld_safe` first
  - Check Sysbech and myq_status
  - Check ProxySQL status
- Restart node2

```
node2# systemctl restart mysql
```

Stop writer node

- Stop Mysql on node3
- Check Sysbech and myq_status
- Check ProxySQL status

```
node3# systemctl stop mysql@bootstrap
proxysql> select * from stats_mysql_connection_pool \
order by hostgroup,srv_host desc;

+-----------+--------------+----------+--------...
| hostgroup | srv_host     | srv_port | status ...
+-----------+--------------+----------+--------...
| 10        | 192.168.70.2 | 3306     | ONLINE ...
| 20        | 192.168.70.3 | 3306     | ONLINE ...
+-----------+--------------+----------+--------...
```

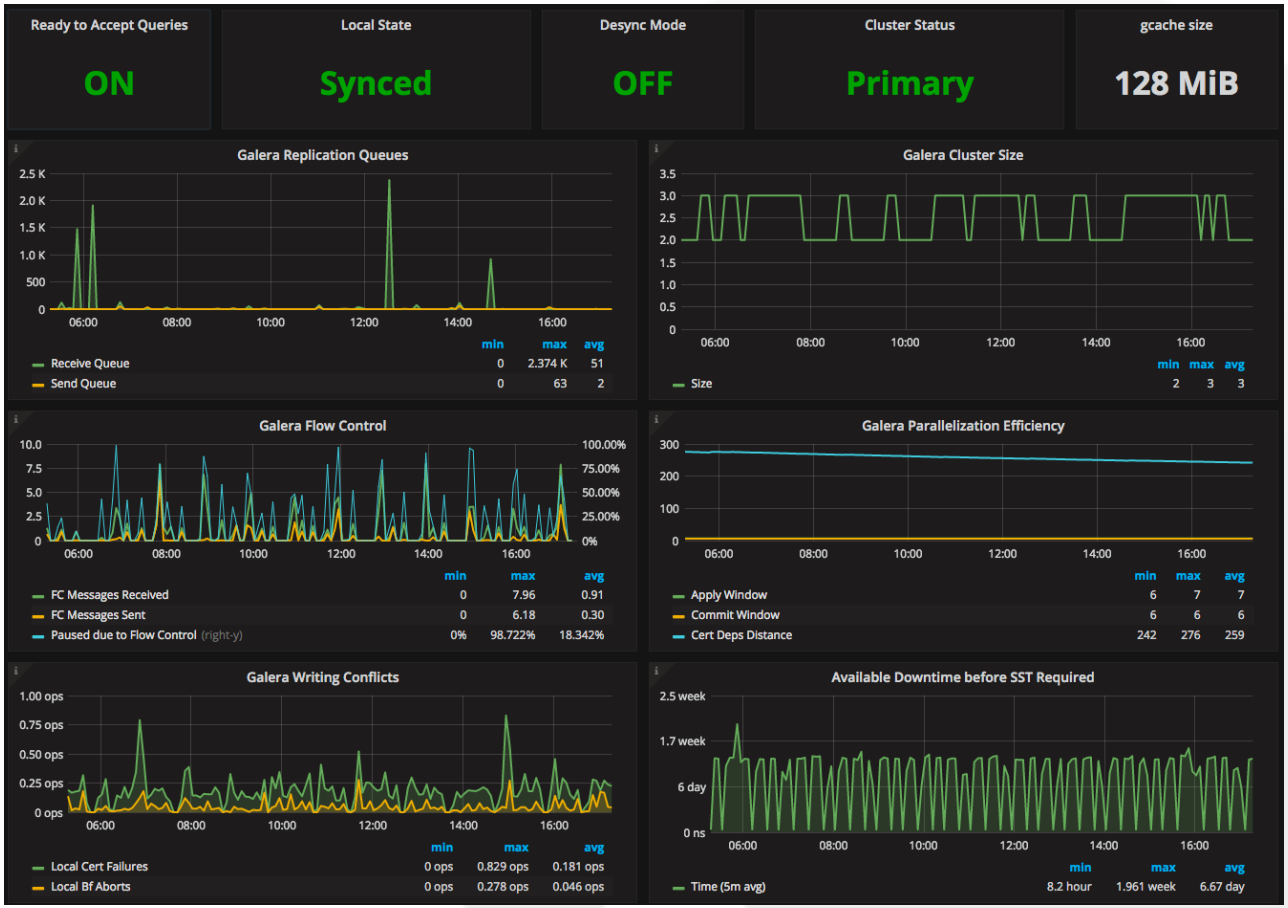
- Node1 is in the write hostgroup now.
- Restart node3

```
node3# systemctl restart mysql
```

XtraDB Cluster Tutorial

MONITORING GALERA with PMM

Percona Monitoring and Management



(*) https://pmmdemo.percona.com/

Start PMM container

- Start Docker
- Create PMM Container

```
node1# systemctl restart docker
node1# docker pull percona/pmm-server:latest
node1# docker create \
   -v /opt/prometheus/data \
   -v /opt/consul-data \
   -v /var/lib/mysql \
   -v /var/lib/grafana \
   --name pmm-data \
   percona/pmm-server:latest /bin/true
```

- Start the container

```
node1# docker run -d \
   -p 80:80 \
   --volumes-from pmm-data \
   --name pmm-server \
   --restart always \
   percona/pmm-server:latest
```

- Check the Dashboard
  - open http://192.168.70.2/graph/

Configure PMM Clients on All nodes

```
node1-3# pmm-admin config --server 192.168.70.2
node1-3# pmm-admin add mysql
```

- Check the Dashboard again
  - open http://192.168.70.2/graph/

XtraDB Cluster Tutorial

GALERA REPLICATION INTERNALS

Group Replication

- Based on Totem Single-ring Ordering
- All nodes have to ACK message
- Uses binlog row events
  - But does not require binary logging
- Writes events to Gcache (configurable size)

Replication Roles

- Within the cluster, all nodes are equal
- master/donor node
  - The source of the transaction.
- slave/joiner node
  - Receiver of the transaction.
- Writeset: A transaction; one or more row changes

State Transfer Roles

- New nodes joining an existing cluster get provisioned automatically
  - Joiner = New node
  - Donor = Node giving a copy of the data
- State Snapshot transfer
  - Full backup of Donor to Joiner
- Incremental Snapshot transfer
  - Only changes since node left cluster

What's inside a write-set?

- The payload; Generated from the RBR event
- Keys; Generated by the source node
  - Primary Keys
  - Unique Keys
  - Foreign Keys
- Keys are how certification happes
  - A `PRIMARY KEY` on every table is required.

Replication Steps

- As Master/Donor
  - Apply
  - Replicate
  - Certify
  - Commit
- As Slave/Joiner
  - Replicate (receive)
  - Certify (includes reply to master/donor)
  - Apply
  - Commit

What is Certification?

- In a nutshell: "Can this transaction be applied?"
- Happens on every node for all write-sets
- Deterministic
- Results of certification are not relayed back to master/donor
  - In event of failure, drop transaction locally; Possibly remove self from cluster.
  - On success, write-set enters apply queue
- Serialized via group communication protocol

(*) https://www.percona.com/blog/2016/04/17/how-percona-xtradb-cluster-certification-works/

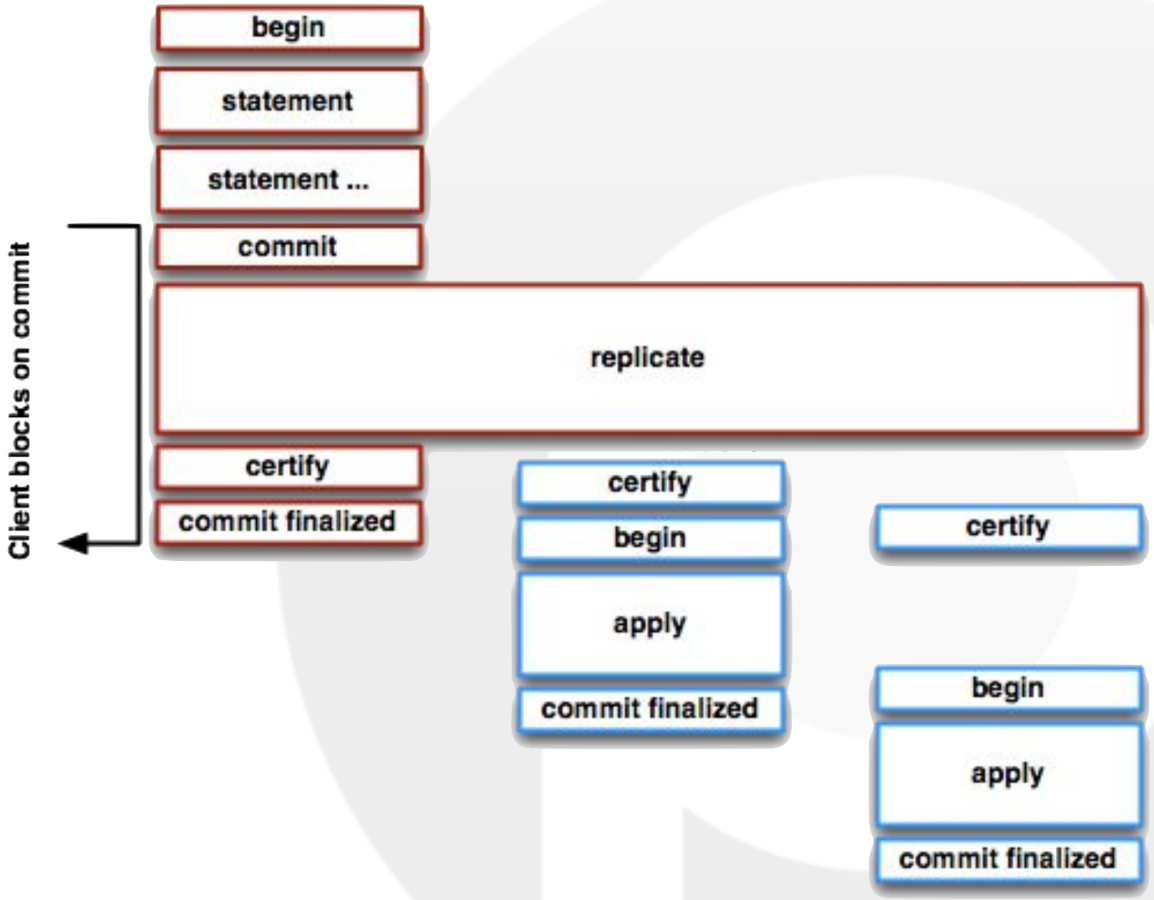
The Apply

- Apply is done locally, asynchronously, on each node, after certification.
- Can be parallelized like standard async replication (more on this later).
- If there is a conflict, will create **brute force aborts** (bfa) on local node.

Last but not least, Commit Stage

- Local, on each node, InnoDB commit.
- Applier thread executes on slave/joiner nodes.
- Regular connection thread for master/donor node.

Galera Replication



Galera Replication Facts

- "Replication" is synchronous; slave apply is not
- A successful commit reply to client means a transaction **WILL** get applied across the entire cluster.
- A node that tries to commit a conflicting transaction before will get a deadlock error.
  - This is called a *local certification failure* or 'lcf'
- When our transaction is applied, any competing transactions that haven't committed will get a deadlock error.
  - This is called a *brute force abort* or 'bfa'

Unexpected Deadlocks

- Create a test table

```
node1 mysql> CREATE TABLE test.deadlocks (i INT UNSIGNED NOT NULL
PRIMARY KEY, j VARCHAR(32) );

node1 mysql> INSERT INTO test.deadlocks VALUES (1, NULL);
node1 mysql> BEGIN;  -- Start explicit transaction
node1 mysql> UPDATE test.deadlocks SET j = 'node1' WHERE i = 1;
```

- Before we commit on node1, go to node3 in a separate window:

```
node3 mysql> BEGIN; -- Start transaction
node3 mysql> UPDATE test.deadlocks SET j = 'node3' WHERE i = 1;

node3 mysql> COMMIT;

node1 mysql> COMMIT;

node1 mysql> SELECT * FROM test.deadlocks;
```

Unexpected Deadlocks (cont.)

- Which commit succeeds?
- Will only a `COMMIT` generate the deadlock error?
- Is this a *lcf* or *bfa*?
- How would you diagnose this error?
- Things to watch during this exercise:
  - `myq_status` (especially on node1)
  - `SHOW ENGINE INNODB STATUS`

Application Hotspots

- Start sysbench on all the nodes at the same time

```
$ /usr/local/bin/run_sysbench_update_index.sh
```

- Watch myq_status 'Conflct' columns for activity

```
$ myq_status wsrep
```

- Edit the script and adjust the following until you see some conflicts:
  - *--oltp-table-size* smaller (250)
  - *--tx-rate* higher (75)

When could you multi-node write?

- When your application can handle the deadlocks gracefully
  - *wsrep_retry_autocommit* can help
- When the successful commit to deadlock ratio is not too small

AUTO_INCREMENT Handling

- Pick one node to work on:

```
node2 mysql> create table test.autoinc (
i int not null auto_increment primary key, j varchar(32) );

node2 mysql> insert into test.autoinc (j) values ('node2' );
node2 mysql> insert into test.autoinc (j) values ('node2' );
node2 mysql> insert into test.autoinc (j) values ('node2' );
```

AUTO_INCREMENT Handling (cont.)

- Now select all the data in the table

```
node2 mysql> select * from test.autoinc;
```

- What is odd about the values for 'i'?
- What happens if you do the inserts on each node in order?
- Check *wsrep_auto_increment_control* and *auto_increment%* global variables

The Effects of Large Transactions

- Sysbench on node1 as usual (Be sure to edit script and reset connection to `localhost`)

```
node1# run_sysbench_oltp.sh
```

- Insert 100K rows into a new table on node2

```
node2 mysql> CREATE TABLE test.large LIKE test.sbtest1;
node2 mysql> INSERT INTO test.large SELECT * FROM test.sbtest1;
```

- Pay close attention to how the application (sysbench) is affected by the large transactions

Serial and Parallel Replication

- Every node replicates, certifies, and commits **EVERY** transaction in the **SAME** order
  - This is serialization
- Replication allows interspersing of smaller transactions simultaneously
  - Big transactions tend to force serialization on this
- The parallel slave threads can **APPLY** transactions with no interdependencies in parallel on **SLAVE** nodes (i.e., nodes where the trx originated from)
  - Infinite *wsrep_slave_threads* != infinite scalability

XtraDB Cluster Tutorial

ONLINE SCHEMA CHANGES

How do we `ALTER TABLE`

- Directly - `ALTER TABLE`
  - "Total Order Isolation" - TOI (default)
  - "Rolling Schema Upgrades" - RSU
- `pt-online-schema-change`

Direct ALTER TABLE (TOI)

- Ensure sysbench is running against node1 as usual
- Try these ALTER statements on the nodes listed and note the effect on sysbench throughput.

```
node1 mysql> ALTER TABLE test.sbtest1 ADD COLUMN `m` varchar(32);
node2 mysql> ALTER TABLE test.sbtest1 ADD COLUMN `n` varchar(32);
```

- Create a different table not being touched by sysbench

```
node2 mysql> CREATE TABLE test.foo LIKE test.sbtest1;
node2 mysql> INSERT INTO test.foo SELECT * FROM test.sbtest1;
node2 mysql> ALTER TABLE test.foo ADD COLUMN `o` varchar(32);
```

- Does running the command from another node make a difference?
- What difference is there altering a table that sysbench is not using?

Direct ALTER TABLE (RSU)

- Change to RSU:

```
node2 mysql> SET wsrep_osu_method = 'RSU';
node2 mysql> alter table test.foo add column `p` varchar(32);
node2 mysql> alter table test.sbtest1 add column `q` varchar(32);
```

- Watch `myq_status` and the error log on node2:

```
node2 mysql> alter table test.sbtest1 drop column `c`;
```

- What are the effects on sysbench after adding the columns p and q?
- What happens after dropping 'c'?
  - Why?
  - How do you recover?
- When is RSU safe and unsafe to use?

pt-online-schema-change

- Be sure all your nodes are set back to TOI

```
node* mysql> SET GLOBAL wsrep_osu_method = 'TOI';
```

- Let's add one final column:
  - Separate terminal to node1

```
node1# pt-online-schema-change --alter \
"ADD COLUMN z varchar(32)" D=test,t=sbtest1 --execute
```

- Was `pt-online-schema-change` any better at doing DDL?
- Was the application blocked at all?
- What are the caveats to using pt-osc?

XtraDB Cluster Tutorial

INCREMENTAL STATE TRANSFER

Incremental State Transfer (IST)

- Helps to avoid full SST
- When a node starts, it knowns the UUID of the cluster to which it belongs and the last sequence number it applied.
- Upon connecting to other cluster nodes, broadcast this information.
- If another node has cached write-sets, trigger IST
  - If no other node has, trigger SST
- Firewall Alert: Uses port 4568 by default

Galera Cache - "gcache"

- Write-sets are contained in the "gcache" on each node.
- Preallocated file with a specific size.
  - Default size is 128M
- Represents a single-file circular buffer.
- Can be increase via provider option
  - `wsrep_provider_options = "gcache.size=2G";`
- Cache is mmaped (I/O buffered to memory)
- `wsrep_local_cached_downto` - The first seqno present in the cache for that node

Simple IST Test

- Make sure your application is running
- Watch `myq_status` on all nodes
- On another node
  - watch mysql error log
  - stop mysql
  - start mysql
- How did the cluster react when the node was stopped and restarted?
- Was the messaging from the init script and in the log clear?
- Is it easy to tell when a node is a Donor for SST vs IST?

Tracking IST

- Percona XtraDB Cluster 5.7+ adds IST monitoring

```
mysql> SHOW STATUS LIKE 'wsrep_ist_receive_status'\G
*************************** 1. row ***************************
Variable_name: wsrep_ist_receive_status
        Value: 52% complete, received seqno 1506799 of 1415410-1589676
1 row in set (0.01 sec)
```

XtraDB Cluster Tutorial

NODE FAILURE SCENARIOS

Node Failure Exercises

- Run sysbench on node1 as usual, also watch myq_status
- Experiment 1: Clean node restart

```
node3# systemctl restart mysql
```

Node Failure Exercises (cont.)

- Experiment 2: Partial network outage
- Simulate a network outage by using `iptables` to drop traffic

```
node3# iptables -A INPUT -s node1 \
  -j DROP; iptables -A OUTPUT -s node1 -j DROP
```

- After observation, clear rules

```
node3# iptables -F
```

Node Failure Exercises (cont.)

- Experiment 3: node3 stops responding to the cluster
- Another network outage affecting 1 node

```
-- Pay Attention to the command continuation backslashes!
-- This should all be 1 line!
node3# iptables -A INPUT -s node1 -j DROP; \
iptables -A INPUT -s node2 -j DROP; \
iptables -A OUTPUT -s node1 -j DROP; \
iptables -A OUTPUT -s node2 -j DROP
```

- Remove iptables rules (after experiments 2 and 3)

```
node3# iptables -F
```

A Non-Primary Cluster

- Experiment 1: Clean stop of 2/3 nodes (normal sysbench load)

```
node2# systemctl stop mysql
node3# systemctl stop mysql
```

A Non-Primary Cluster (cont.)

- Experiment 2: Failure of 50% cluster (leave node2 down)

```
node3# systemctl start mysql

-- Wait a few minutes for IST to finish and for the
-- cluster to become stable. Watch the error logs and
-- SHOW GLOBAL STATUS LIKE 'wsrep%'

node3# iptables -A INPUT -s node1 -j DROP; \
       iptables -A OUTPUT -s node1 -j DROP
```

- Force node1 to become primary like this:

```
node1 mysql> set global wsrep_provider_options="pc.bootstrap=true";
```

- Clear iptables when done
  - `node3# iptables -F`

Using garbd as a third node

- Node2's mysql is still stopped!
- Install the arbitrator daemon on to node2

```
node2# yum install Percona-XtraDB-Cluster-garbd-3.x86_64
node2# garbd -a gcomm://node1:4567,node3:4567 -g mycluster
```

- Experiment 1: Reproduce node3 failure

```
node3# iptables -A INPUT -s node1 -j DROP; \
    iptables -A INPUT -s node2 -j DROP; \
    iptables -A OUTPUT -s node1 -j DROP; \
    iptables -A OUTPUT -s node2 -j DROP
```

Using garbd as a third node (cont.)

- Experiment 2: Partial node3 network failure

```
node3# iptables -A INPUT -s node1 -j DROP; \
 iptables -A OUTPUT -s node1 -j DROP
```

- Remember to clear iptables when done
  - `node3# iptables -F`

Recovering from a **clean** all-down state

- Stop mysql on all 3 nodes

```
$ systemctl stop mysql
```

- Use the grastate.dat to find the node with the highest GTID

```
$ cat /var/lib/mysql/grastate.dat
```

- Bootstrap that node and then start the others normally

```
$ systemctl start mysql@bootstrap
$ systemctl start mysql
$ systemctl start mysql
```

- Why do we need to bootstrap the node with the highest GTID?
- Why can’t the cluster just “figure it out”?

Recovering from a **dirty** all-down state

- Kill -9 mysqld on all 3 nodes

```
$ killall -9 mysqld_safe; killall -9 mysqld
```

- Check the grastate.dat again. Is it useful?

```
$ cat /var/lib/mysql/grastate.dat
```

- Try `wsrep_recover` on each node to get the GTID

```
$ mysqld_safe --wsrep_recover
```

- Bootstrap the highest node and then start the others normally
- What would happen if we started the wrong node?
- Wsrep Recovery pulls GTID data from the Innodb transaction logs. In what cases might this be inaccurate?

Weighted Quorum

- All nodes have a default weight of 1.
- Primary state is determined by strictly greater than 50% of last stable weighted total.
- Change weight of individual nodes to affect calculation:
  - `SET GLOBAL wsrep_provider_options = 'pc.weight = 2';`

```
node1: pc.weight = 2
node2: pc.weight = 1
node3: pc.weight = 0
```

- Killing node2 and node3 simultaneously preserves the Primary state on node1.
- Killing node1 causes node2 and node3 to become non-primary components.

XtraDB Cluster Tutorial

AVOIDING SST

Manual State Transfer with Xtrabackup

- Take a backup on node3 (sysbench on node1 as usual)

```
node3# xtrabackup --backup --galera-info --target-dir=/var/backup
```

- Stop node3 and prepare the backup

```
node3# systemctl stop mysql
node3# xtrabackup --prepare --target-dir=/var/backup
```

Manual State Transfer with Xtrabackup

- Copy back the backup

```
node3# rm -rf /var/lib/mysql/*
node3# xtrabackup --copy-back --target-dir=/var/backup
```

- Get GTID from backup and update grastate.dat

```
node3# cd /var/lib/mysql
node3# mv xtrabackup_galera_info grastate.dat
node3# nano grastate.dat

# GALERA saved state
version: 2.1
uuid:    8797f811-7f73-11e2-0800-8b513b3819c1
seqno:   22809
safe_to_bootstrap: 0

node3# chown -R mysql:mysql /var/lib/mysql
node3# systemctl start mysql
```

Cases where the state is zeroed

- Add a bad config item to section [mysqld] in /etc/my.cnf

```
[mysqld]
fooey
```

- Check what happens to the grastate:

```
node3# systemctl stop mysql
node3# cat /var/lib/mysql/grastate.dat
node3# systemctl start mysql
node3# cat /var/lib/mysql/grastate.dat
```

- Remove the bad config and use wsrep_recover to avoid SST

```
node3# mysqld_safe --wsrep_recover
```

- Update the grastate.dat with that position and restart
- What would have happened if you restarted with a zeroed grastate.dat?

When you **WANT** an SST

```
node3# cat /var/lib/mysql/grastate.dat
```

- Turn off replication for the session:

```
node3 mysql> SET wsrep_on = OFF;
```

- Repeat until node3 crashes (sysbench should be running on node1)

```
node3 mysql> delete from test.sbtest1 limit 10000;
..after crash..
node3# cat /var/lib/mysql/grastate.dat
```

- What is in the error log after line the crash?
- Should we try to avoid SST in this case?

XtraDB Cluster Tutorial

TUNING REPLICATION

What is Flow Control?

- Ability for any node in the cluster to tell the other nodes to pause writes while it catches up
  - Cry for help!
- Feedback mechanism for the replication process
- ONLY caused by `wsrep_local_recv_queue` exceeding a node's `fc_limit`
- This can pause the entire cluster and look like a cluster stall!

Observing Flow Control

- Run sysbench on node1 as usual
- Take a read lock on node3 and observe its effect on the cluster

```
node3 mysql> SET GLOBAL pxc_strict_mode = 0;
node3 mysql> LOCK TABLE test.sbtest1 WRITE;
```

- Make observations. When you are done, release the lock:

```
node3 mysql> UNLOCK TABLES;
```

- What happens to sysbench throughput?
- What are the symptoms of flow control?
- How can you detect it?

Tuning Flow Control

- Run sysbench on node1 as usual
- Increase the fc_limit on node3

```
node3 mysql> SET GLOBAL wsrep_provider_options="gcs.fc_limit=500";
```

- Take a read lock on node3 and observe its effect on the cluster

```
node3 mysql> LOCK TABLE test.sbtest1 WRITE;
```

- Make observations. When you are done, release the lock:

```
node3 mysql> UNLOCK TABLES;
```

- Now, how big does the recv queue on node3 get before FC kicks in?

Flow Control Parameters

- `gcs.fc_limit`
  - Pause replication if recv queue exceeds that many writesets.
  - Default: 100. For master-slave setups this number can be increased considerably.
- `gcs.fc_master_slave`
  - When this is *NO*, then the effective `gcs.fc_limit` is multipled by sqrt(number of cluster members). Default: NO.
  - Ex: `gcs.fc_limit = 500` * sqrt(3) = 866

Avoiding Flow Control during Backups

- Most backups use `FLUSH TABLES WITH READ LOCK`
- By default, this causes flow control almost immediately
- We can permit nodes to fall behind:

```
node3 mysql> SET GLOBAL wsrep_desync=ON;
node3 mysql> FLUSH TABLES WITH READ LOCK;
```

- Do not write to this node!
- No longer the case with 5.7! Yay!

More Notes on Flow Control

- If you set it too low:
  - Too many cries for help from random nodes in the cluster
- If you set it too high:
  - Increases in replication conflicts in multi-node writings
  - Increases in apply/commit lag on nodes
- That one node with FC issues:
  - Deal with that node: bad hardware? not same specs?

Testing Node Apply throughput

- Using `wsrep_desync` and `LOCK TABLE`, we can stop replication on a node for an arbitrary amount of time (until it runs out of disk-space).
- We can stop a node for a while, release it, and measure the top speed at which it applies.
- This gives is an idea of how fast the node is capable of handling the bulk of replication.
- Performance here is greatly improved in 5.7(*)

(*) https://www.percona.com/blog/2017/04/19/how-we-made-percona-xtradb-cluster-scale/

Wsrep Sync Wait

- Apply is asynchronous, so reads after writes on different nodes may see an older view of the database.
- In reality, flow control prevents this from becoming like regular Async replication
- Tighter consistency can be enforced by ensuring the replication queue is flushed before a read
  - `SET [GLOBAL] wsrep_sync_wait=[0-7];`
- Setting it ensures synchronous read view before executing an operation:
  - 0 - Disabled / 1 - READ (SELECT, BEGIN)
  - 2 - UPDATE and DELETE
  - 3 - READ, UPDATE and DELETE
  - 4 - INSERT and REPLACE

XtraDB Cluster Tutorial

CONCLUSION

Limitations

- Does not work as expected:
  - InnoDB/XtraDB Only
  - `tx_isolation = SERIALIZABLE`
  - `GET_LOCK()`
  - `LOCK TABLES`
  - `SELECT ... FOR UPDATE`
  - Careful with `ALTER TABLE ... IMPORT/EXPORT`.
  - Capped maximum transaction size.
  - XA transactions.

Common Configuration Parameters

wsrep_sst_auth = un:pw	Set SST Authentication
wsrep_osu_method = 'TOI'	Set DDL Method
wsrep_desync = ON;	Enable "Replication"
wsrep_slave_threads = 4;	Number of applier threads
wsrep_retry_autocommit = 4;	Number of commit retries
wsrep_provider_options:	
* gcs.fc_limit = 500	Increase flow control threshold
* pc.bootstrap = true	Put node into single mode
* gcache.size = 2G	Increase Galera Cache file
* pc.weight = 2	Increase voting of a node
wsrep_sst_donor = node1	Specify donor node

Common Status Counters

wsrep_cluster_status	Primary / Non-Primary
wsrep_local_state_comment	Synced / Donor / Desync
wsrep_cluster_size	Number of online nodes
wsrep_flow_control_paused	Number of seconds cluster has paused
wsrep_local_recv_queue	Current number of write-sets waiting to be applied

PERCONA

TRAINING

Questions?

(*) Slide content authors: Jay Janssen, Kenny Gryp, Fred Descamps, Matthew Boehm