PERCONA

Percona XtraDB Cluster in Practice

Jay Janssen
Managing Consultant, Percona
Percona Live MCE 2015

Virtual Machine Setup

1. Grab an ORANGE Percona USB stick
   - Copy the OVA file onto your Machine
   - Grab OSX VirtualBox installer
2. The RED USB stick has Virtualbox for all OSes
3. Install and Open Virtualbox, Import OVA, Start nodes
4. SSH to servers:
   - login: root / password: vagrant
   - ssh -p 22201 root@127.0.0.1 (node1)
   - ssh -p 22202 root@127.0.0.1 (node2)
   - ssh -p 22203 root@127.0.0.1 (node3)

Raise your hand if you are having any problems!
32-bit not supported!

Resources

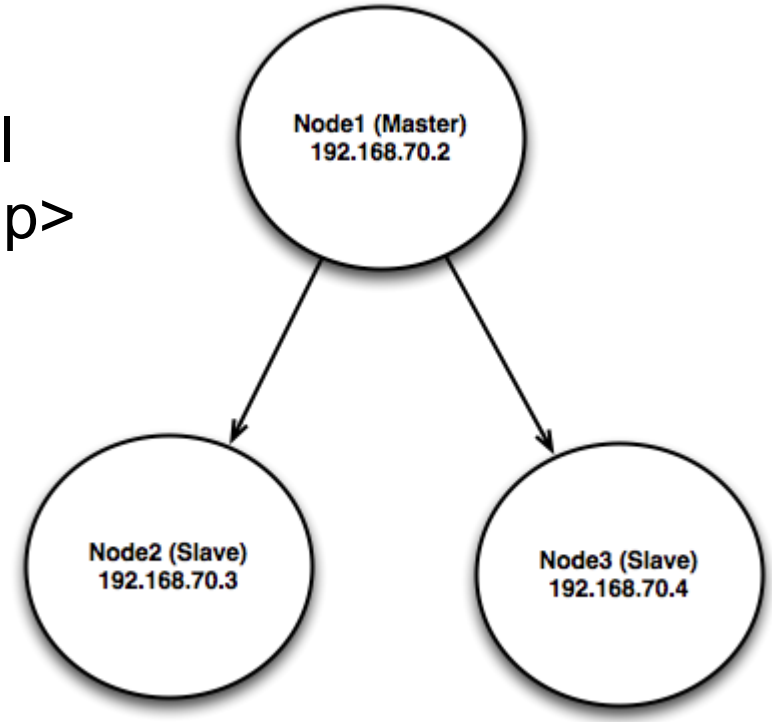
- These slides: http://bit.ly/pxc-tutorial-slides
- http://www.percona.com/doc/percona-xtradb-cluster/5.6/
- http://galeracluster.com/documentation-webpages/reference.html
- https://github.com/percona/xtradb-cluster-tutorial
- http://www.mysqlperformanceblog.com/category/percona-xtradb-cluster/

PERCONA

Migrating a Master/Slave to PXC

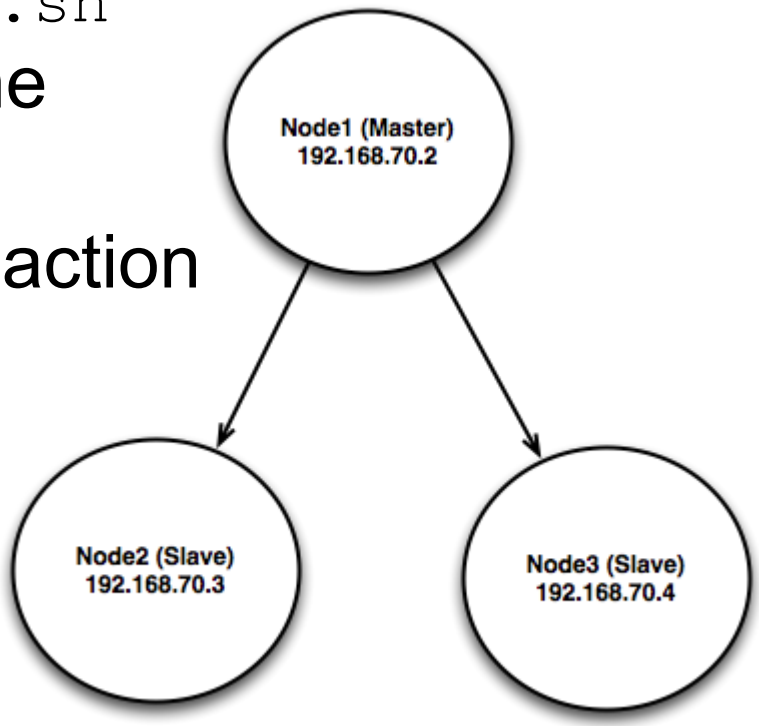
Verify your Master/Slave setup

- Connect to all 3 nodes
  - separate terminals are helpful
  - ssh -p2220<node #> root@<ip>
  - password: vagrant
- Verify MySQL is running
  - root# mysql
- Verify replication is running
  - SHOW SLAVE STATUS\G



Start the application and verify it is working

- `node1# run_sysbench_oltp.sh`
- Is replication flowing to all the nodes?
- Observe the sysbench transaction rate



Checking Consistency

```
1.  # Run a pt-table-checksum from node1
2.  [root@node1 ~]# pt-table-checksum
3.
4.  # Check for different checksums
5.  node3> SELECT db, tbl, SUM(this_cnt) AS total_rows, COUNT(*) AS chunks
    FROM percona.checksums WHERE ( master_cnt <> this_cnt OR master_crc <>
    this_crc OR ISNULL(master_crc) <> ISNULL(this_crc)) GROUP BY db, tbl;
6.
7.  # See all the results
8.  node3> select * from percona.checksums;
```

Are there any differences between node1 and its slaves? If so, why?

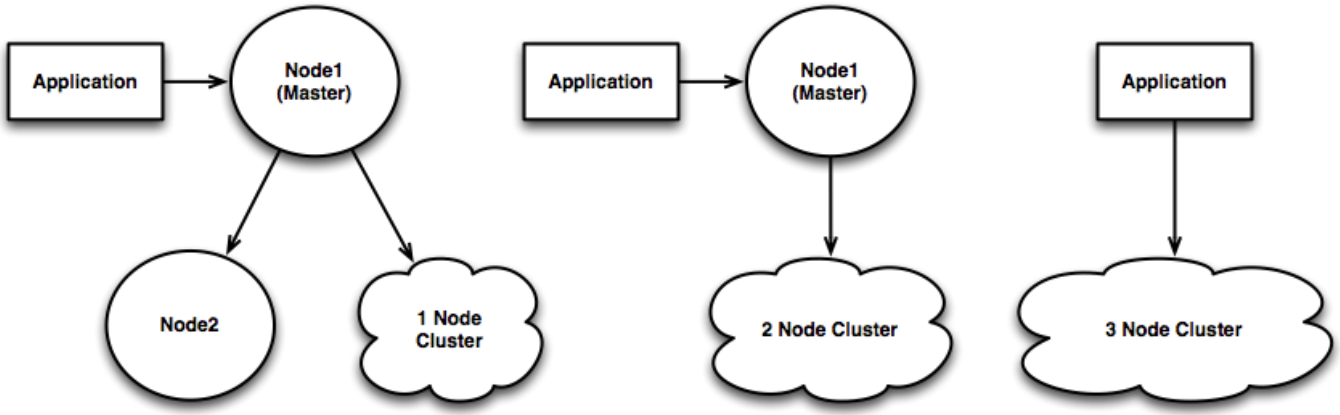
Monitoring Replication Lag

```
1.  # Start a heartbeat on node1
2.  [root@node1 ~]# pt-heartbeat --update --database percona --create-
    table --daemonize
3.
4.  # Monitor the heartbeat on node2
5.  [root@node2 ~]# pt-heartbeat --monitor --database percona --master-
    server-id=1
```

Stop the heartbeat on node1 and see how that affects the monitor.
STOP SLAVE on node2 for a while and see how that affects the monitor

What is our plan?

- We have an application working on node1
- We want to migrate our master/slave pool to a PXC cluster with minimal downtime
- We will build a cluster from one slave and then add the other servers to it one at a time



Upgrade node3 to PXC

```
1.  [root@node3 ~]# systemctl stop mysql
2.  [root@node3 ~]# yum swap -- remove Percona-Server-shared-56 Percona-
    Server-server-56 -- install Percona-XtraDB-Cluster-shared-56 Percona-
    XtraDB-Cluster-server-56
3.  [root@node3 ~]# systemctl start mysql
4.  [root@node3 ~]# mysql -e "show slave status\G"
```

The Percona XtraDB Cluster Server and Client packages are drop-in replacements for Percona Server and even community MySQL. It is the configuration that enables the cluster features.

Configure Node3 for the Cluster

```
[mysqld]
# Leave existing settings and add these
server-id                       = 3
binlog_format                   = ROW

# galera settings
wsrep_provider                  = /usr/lib64/libgalera_smm.so

wsrep_cluster_name              = mycluster
wsrep_cluster_address           = gcomm://192.
168.70.2,192.168.70.3,192.168.70.4
wsrep_node_name                 = node3
wsrep_node_address              = 192.168.70.4

wsrep_sst_auth                = sst:secret

innodb_autoinc_lock_mode        = 2
innodb_locks_unsafe_for_binlog  = ON
```

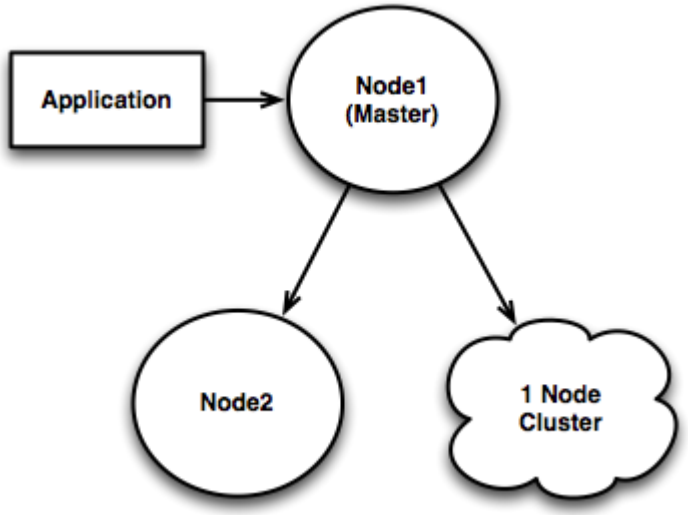
Start node3 with:
systemctl start mysql@bootstrap

Checking Cluster State

- Use ‘myq_status’ to check the state of Galera on node3:
  - [root@node3 ~]# myq_status wsrep
- Can you tell:
  - How many nodes are in the cluster?
  - Is the cluster Primary?
  - Are cluster replication events being generated?

No cluster replication events!

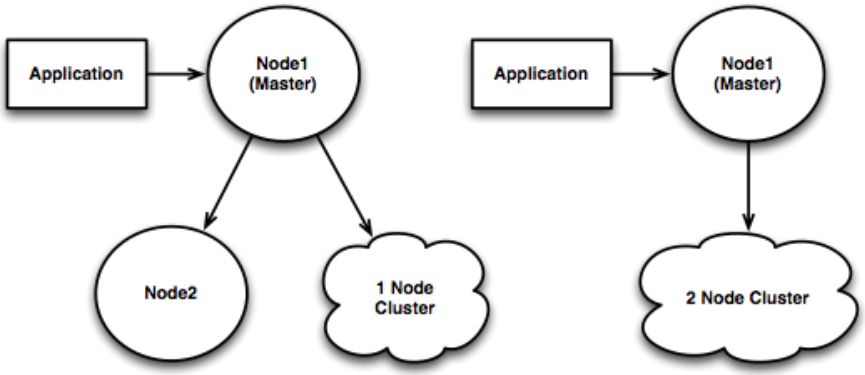
- What would the result be if we added node2 to the cluster?
- What can be done to be sure writes from replication into node3 are replicated to the cluster?



Hint: If you've ever configured relay slaves in MySQL replication, you know the fix!

Adding node2 to the cluster

- We only need a single node replicating from our production master
  - node2> stop slave; reset slave;
- Stop mysql and update the packages as before
- Edit node2's my.cnf
  - What needs to be different from node3's config?

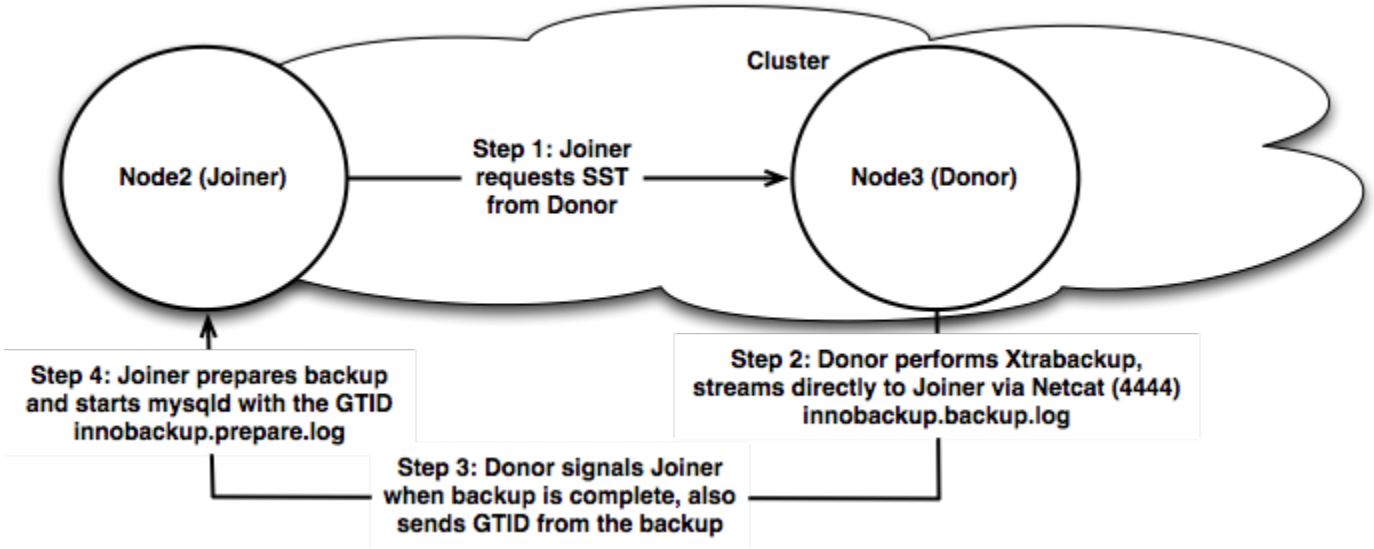


Node2's my.cnf

```
[mysqld]
# Leave existing settings and add these
server-id                       = 3
binlog_format                   = ROW
log-slave-updates

# galera settings
wsrep_provider                  = /usr/lib64/libgalera_smm.so

wsrep_cluster_name              = mycluster
wsrep_cluster_address           = gcomm://192.
168.70.2,192.168.70.3,192.168.70.4
wsrep_node_name                 = node2
wsrep_node_address              = 192.168.70.3

wsrep_sst_auth              = sst:secret

innodb_autoinc_lock_mode        = 2
innodb_locks_unsafe_for_binlog  = ON
```

State Snapshot Transfer (SST)

- 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’.



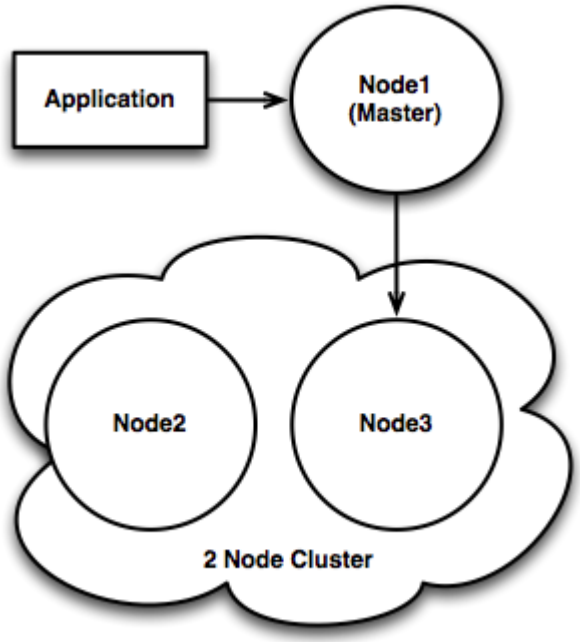
Additional Xtrabackup SST setup

```
1.  # Add an explicit 'datadir' to your my.cnf
2.  [mysqld]
3.  datadir         = /var/lib/mysql
4.
5.  # Grants need to be added for Xtrabackup
6.  node3> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT ON *.* TO
    'sst'@'localhost' IDENTIFIED BY 'secret';
7.
8.  # The user/pass should be added to your my.cnf
9.  [mysqld]
10. …
11. wsrep_sst_auth=sst:secret
12. ...
```

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

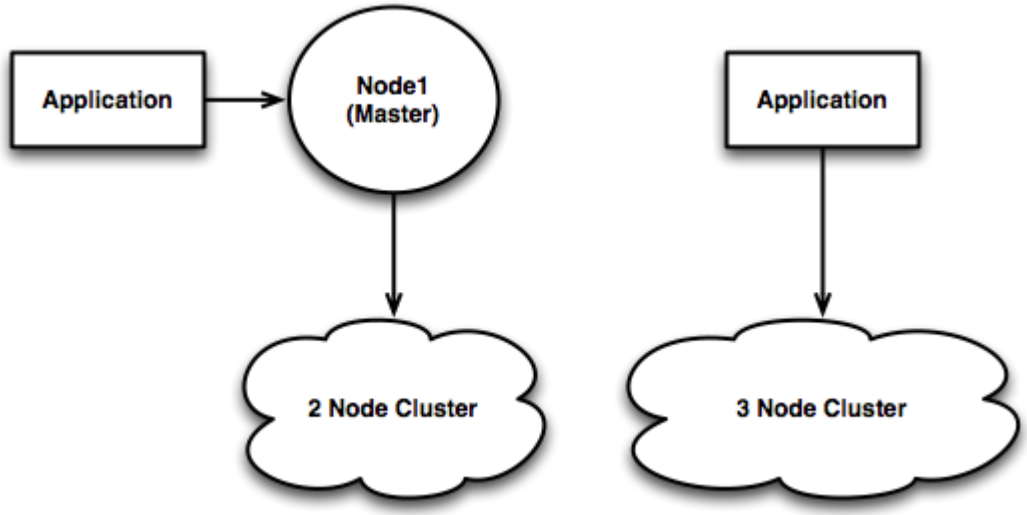
Taking Stock of our Achievements

- We have a two node cluster
- Our production master replicates into the cluster via node3
- What might we do now in a real migration before proceeding?



Finishing the Migration

- How should we finish the migration?
- How do we minimize downtime?
- What do we need to do to ensure there are no data inconsistencies?
- How might we rollback?



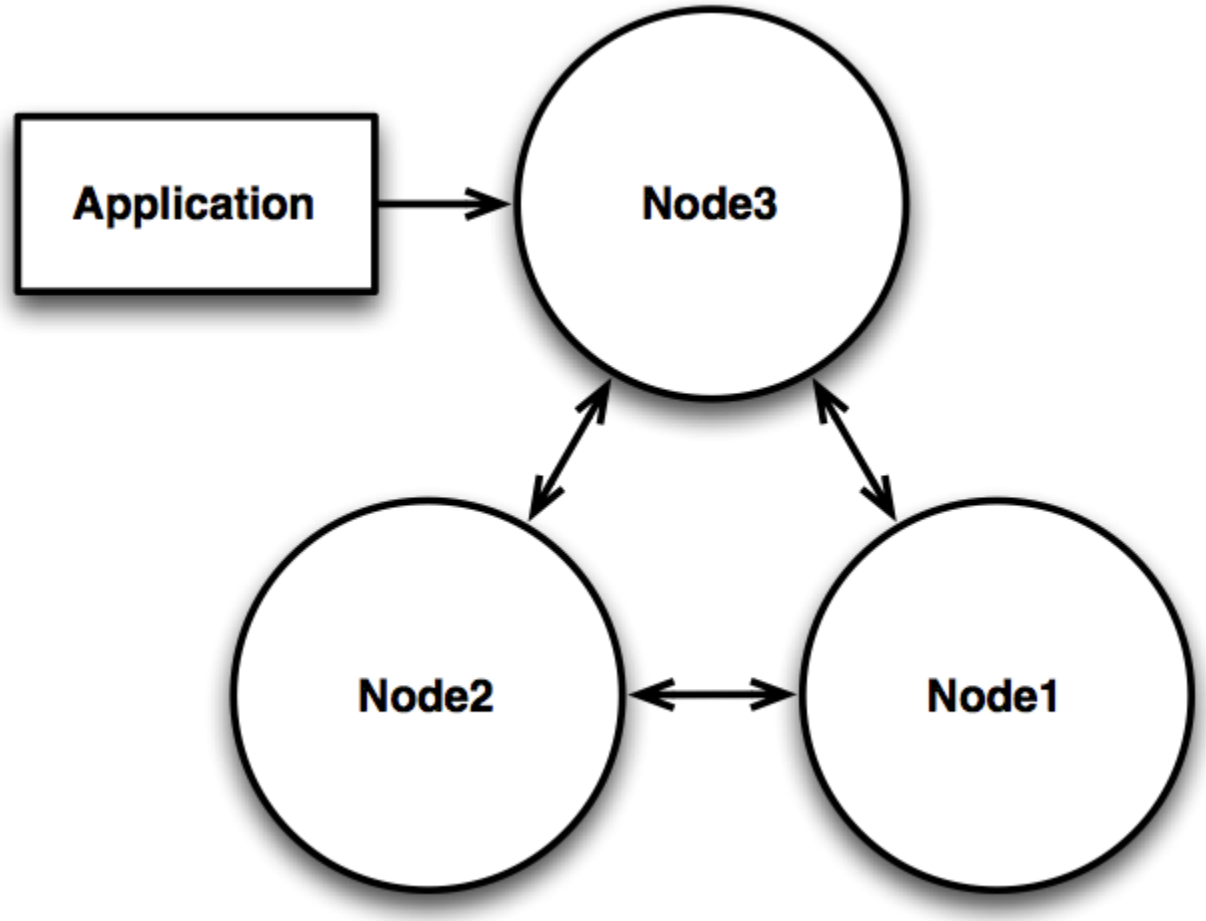
Possible Migration Steps

- Shutdown the application pointing to node1
- Shutdown (and RESET) replication on node3 from node1
- Startup the application pointing to node3
- Rebuild node1 as another member of the cluster

Migrate your application to run against node3 and minimize downtime.

Migrate node1 to the cluster with the knowledge you've gained so far.

Where are we now?



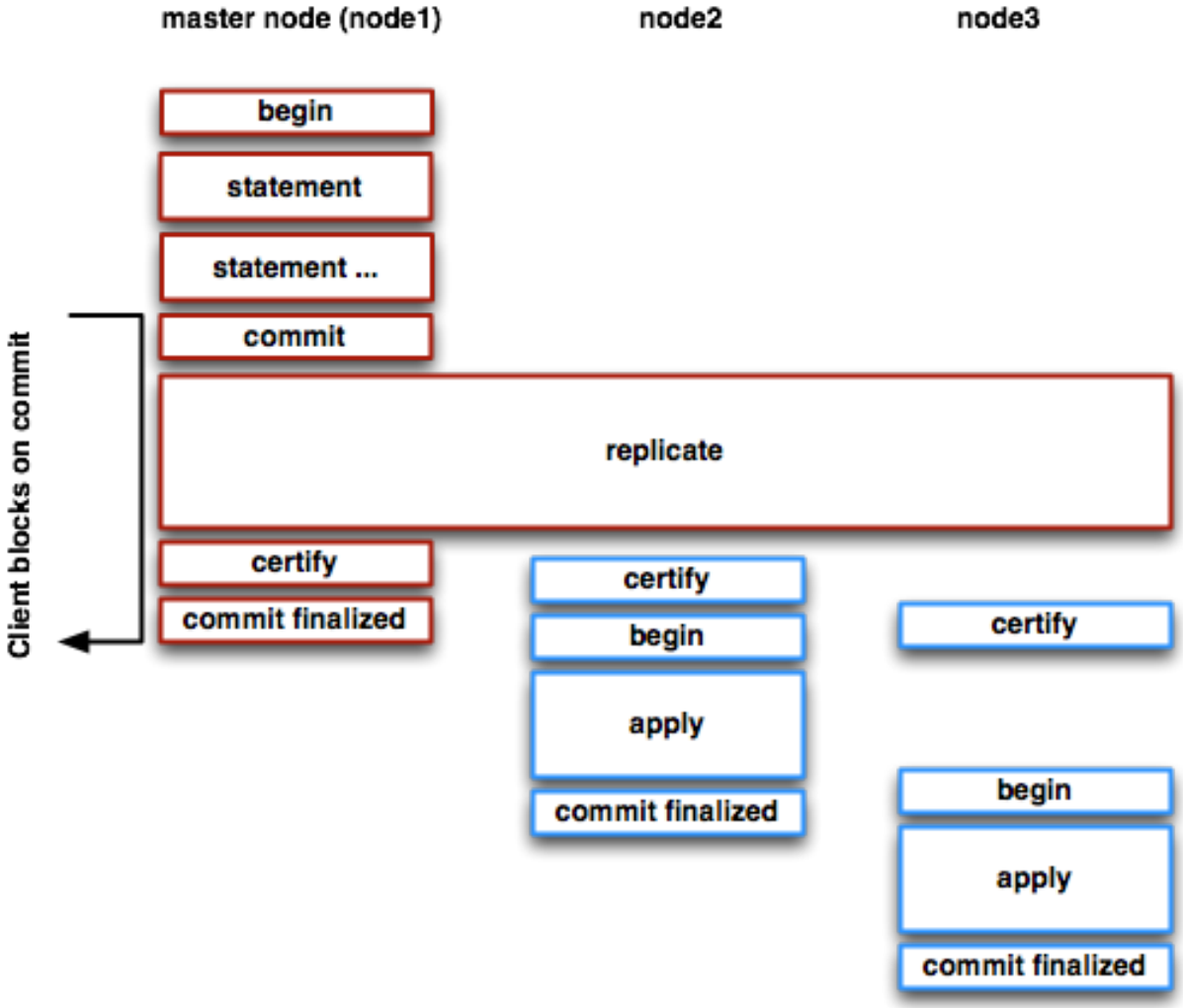
PERCONA

Application Considerations

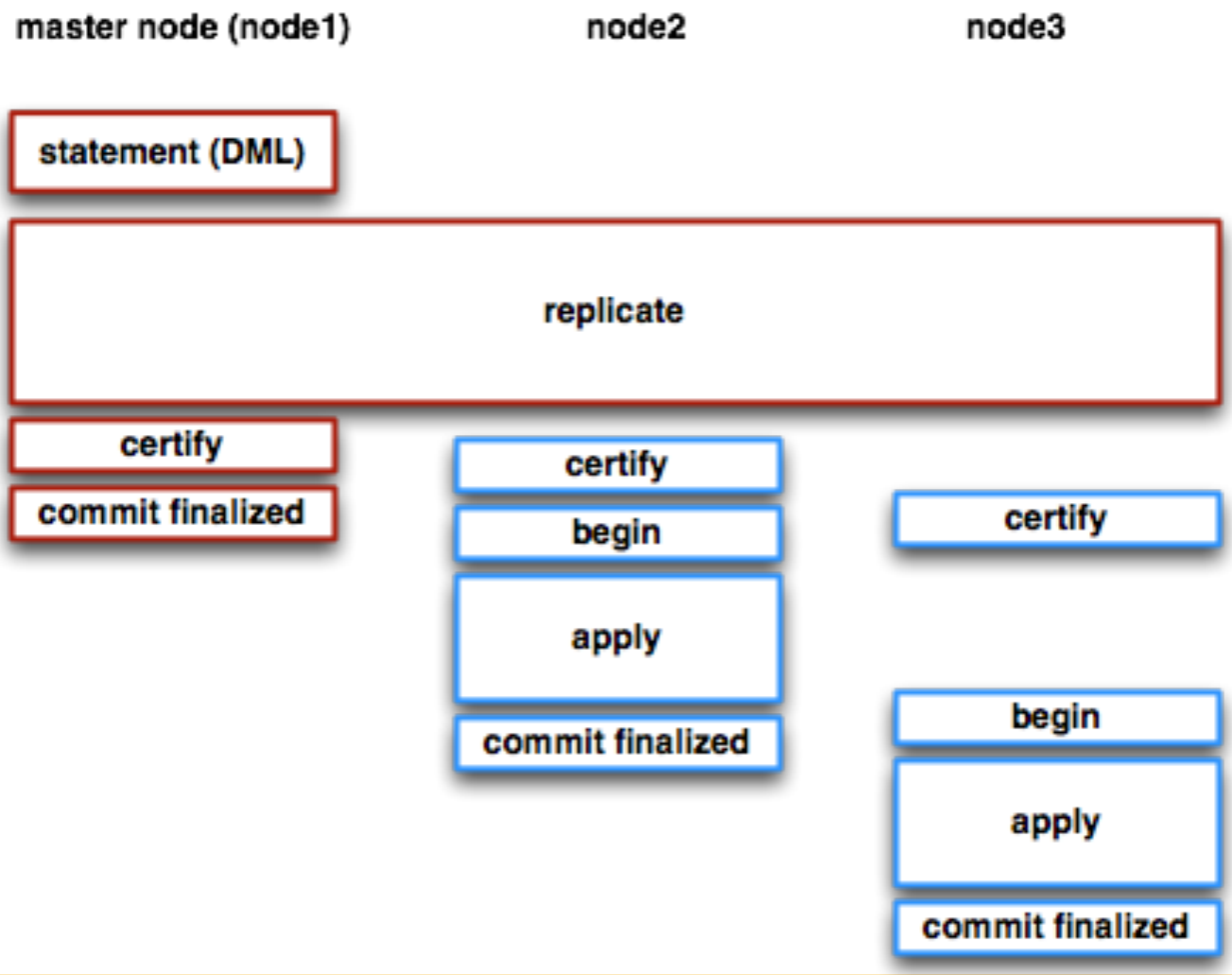
PERCONA

Galera Replication and Application Behavior

Galera replication (full transaction)



Galera Replication (autocommit)



Galera Replication Facts

- “Replication” is **Synchronous**, slave apply is not
- A successful commit reply to client means transaction WILL get applied across the entire cluster.
- A slave 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

```
1.  # Create a test table
2.  node1 mysql> create table test.deadlocks( i int unsigned not null
    primary key, j varchar(32) );
3.  node1 mysql> insert into test.deadlocks values ( 1, NULL );

4.  node1 mysql> begin; update test.deadlocks set j='node1' where i=1;
5.
6.  # Before commit, go to node3 in a separate window:
7.  node3 mysql> begin; update test.deadlocks set j='node3' where i=1;
8.  node3 mysql> commit;
9.
10. node1 mysql> commit;
11. node1 mysql> select * from test.deadlocks;
```

Things to watch during this:
myq_status (especially on node1)
SHOW ENGINE INNODB STATUS

Which commit succeeds?
Will only commit generate the deadlock error?
Is this a lcf or bfa?
How would you diagnose this error?

Application Hotspots

1. # Start sysbench on all the nodes at the same time
   $ cat /usr/local/bin/run_sysbench_update_index.sh
   sysbench --db-driver=mysql --
   test=/usr/share/doc/sysbench/tests/db/update_index.lua --mysql-
   user=test --mysql-password=test --mysql-db=test --mysql-host=localhost
   --mysql-ignore-errors=all --oltp-tables-count=1 --oltp-table-
   size=250000 --oltp-auto-inc=off --num-threads=1 --report-interval=1 --
   max-requests=0 --tx-rate=10 run | grep tps

2. # Watch myq_status 'Conflct' columns for activity
   $ myq_status wsrep

Adjust the following until you see some conflicts:
--oltp-table-size smaller
--tx-rate higher

When should you multi-node write?

- When your application can handle the deadlocks gracefully
  - wsrep_retry_autocommit can help somewhat
- When the successful commit to deadlock ratio is not too small
- NOT usually when you just don’t want to:
  - think about it
  - make changes to your application

Autoincrement Handling

```
1.  # pick a node to work on
2.  node2 mysql> create table test.autoinc ( i int unsigned not null
    auto_increment primary key, j varchar(32) );
3.
4.  node2 mysql> insert into test.autoinc (j) values ('node2' );
5.  node2 mysql> insert into test.autoinc (j) values ('node2' );
6.  node2 mysql> insert into test.autoinc (j) values ('node2' );
7.
8.  # Now select all the data in the table
9.  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

Those not present during the first session

- Find orange USB stick
- Open these slides at:
  - http://bit.ly/pxc-tutorial-slides
- Read Install instructions on slide #2
- We are starting on slide 32

The effect of large transactions

```
1.  # Sysbench on node1 as usual
2.  [root@node1 ~]# run_sysbench_oltp.sh

3.  # Insert 100k rows into a new table on node2
4.  node2 mysql> create table test.large like test.sbtest1;
5.  node2 mysql> insert into test.large select * from test.sbtest1 limit
    100000;
6.
7.  # Use an alternative method to accomplish the same thing
8.  node2 mysql> create table test.ptarchiver like test.sbtest1;
9.  [root@node2 ~]# pt-archiver --where 1=1 --limit 1000 --commit-each --
    progress=1000 --no-delete --source=D=test,t=sbtest1 --dest=D=test,
    t=ptarchiver
```

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
  - 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

PERCONA

Online Schema Changes

Direct ALTER TABLE (TOI)

Total Order Isolation (default)

```
1.  # Ensure sysbench is running against node1 as usual
2.  # Try these ALTER statements on the nodes listed and note the effect
    on sysbench's throughput.
3.  node1 mysql> alter table test.sbtest1 add column `m` varchar(32);
4.
5.  node2 mysql> alter table test.sbtest1 add column `n` varchar(32);
6.
7.  # Create a different table not being touched by sysbench
8.  node2 mysql> create table test.foo like test.sbtest1;
9.  node2 mysql> insert into test.foo select * from test.sbtest1;
10. 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?

Rolling Schema Upgrades (RSU)

```
1.  # Be sure this is set before each ALTER
2.  node2 mysql> set global wsrep_OSU_method='RSU';
3.
4.  node2 mysql> alter table test.foo add column `p` varchar(32);
5.
6.  node2 mysql> alter table test.sbtest1 add column `q` varchar(32);
7.
8.  # Watch myq_status and your error log on node2 here
9.  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

```
1.  # Be sure all your nodes are set back to TOI
2.  node* mysql> set global wsrep_OSU_method='TOI';
3.
4.  # Now let's add one final column:
5.  [root@node2 ~]# 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?

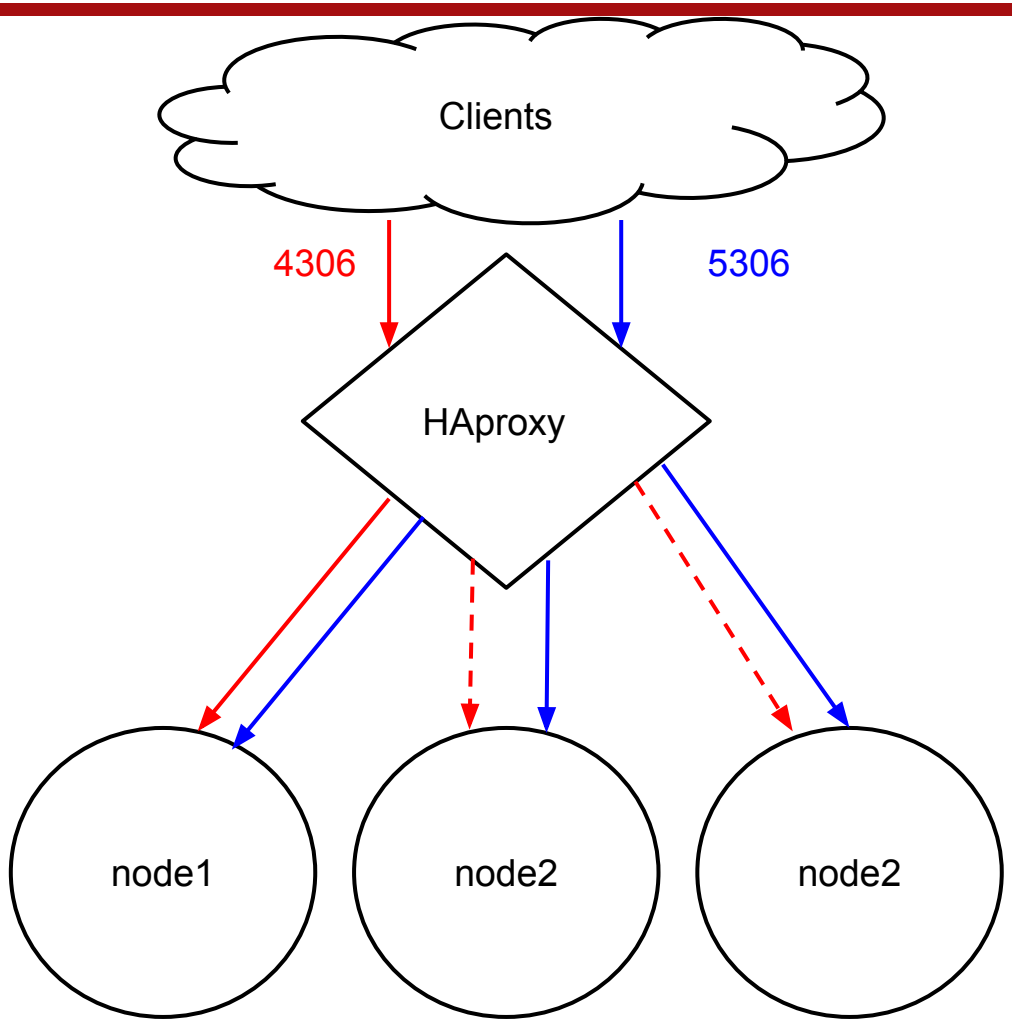
PERCONA

Application to Cluster High Availability

Clustercheck

- Get /usr/bin/clustercheck to work, should output “Node is Synced”
  - Check script contents for what it needs to work
- Check /etc/xinetd.d/mysqlchk and modify to get that to work
- Curl on port 9200 should return clustercheck results
  - curl http://127.0.0.1:9200
- What do you need to do to get it to work?
- Get it working on all 3 nodes!

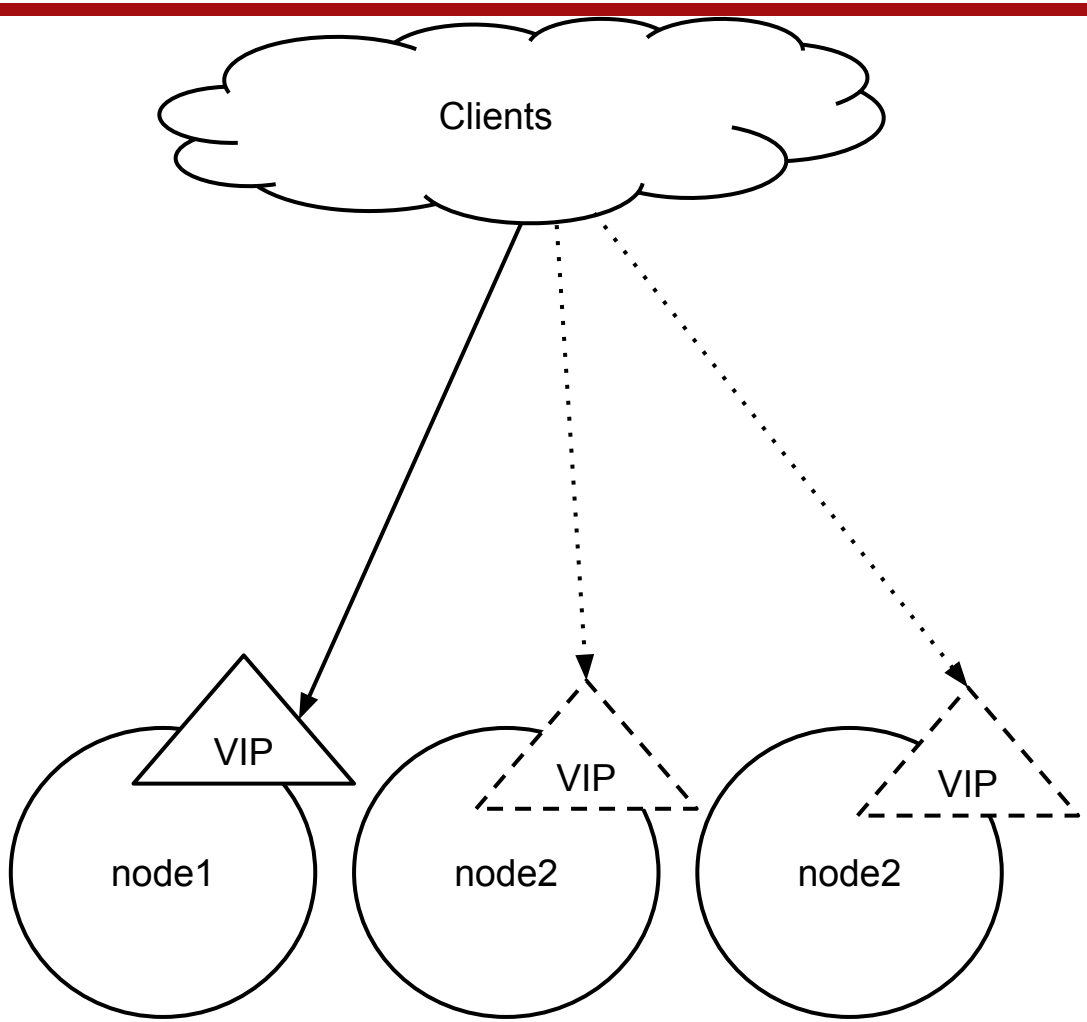
HAProxy Architecture



Setting up HAproxy

- Install HAproxy
  - just on node1 is fine
- Create /etc/haproxy/haproxy.cfg
  - https://gist.github.com/jayjanssen/4944530
  - Restart haproxy
- Test connection to proxy ports
  - mysql -u test -ptest -h 192.168.70.2 -P 4306 -e “show variables like 'wsrep_node_name';”
  - mysql -u test -ptest -h 192.168.70.2 -P 5306 -e “show variables like 'wsrep_node_name';
- Test shutting down nodes
  - Watch http://127.0.0.1:9991

Keepalived Architecture



Keepalived for simple VIP to PXC

- Install keepalived (all nodes)
- Create /etc/keepalived/keepalived.conf
  - https://gist.github.com/jayjanssen/9646544
  - Start on all nodes
- Check for which host has the VIP
  - ip addr list | grep 192.168.70.100
- Experiment with restarting nodes while checking host answering the vip
  - while(true;) do mysql -u test -ptest -h 192.168.70.100 -e "show variables like 'wsrep_node_name'"; sleep 1; done

Application HA essentials

- Nodes are properly health checked
- HA is mostly commonly handled at layer4 (TCP)
  - Therefore, expect node failures to mean app reconnects
- The cluster will prevent apps from doing anything bad
  - But, the cluster may not necessarily force apps to reconnect by itself
- Watch out for Donor nodes

PERCONA

Operational Aspects

PERCONA

Monitoring Galera

Low level

```
1.  mysql> SHOW GLOBAL VARIABLES LIKE 'wsrep%';
2.  mysql> SHOW GLOBAL STATUS LIKE 'wsrep%';
3.
4.  # Refer to:
5.  http://www.codership.com/wiki/doku.php?id=mysql_options_0.8
6.  http://www.codership.com/wiki/doku.php?id=galera_status_0.8
7.  http://www.percona.com/doc/percona-xtradb-cluster/wsrep-status-index.
    html
8.  http://www.percona.com/doc/percona-xtradb-cluster/wsrep-system-index.
    html
```

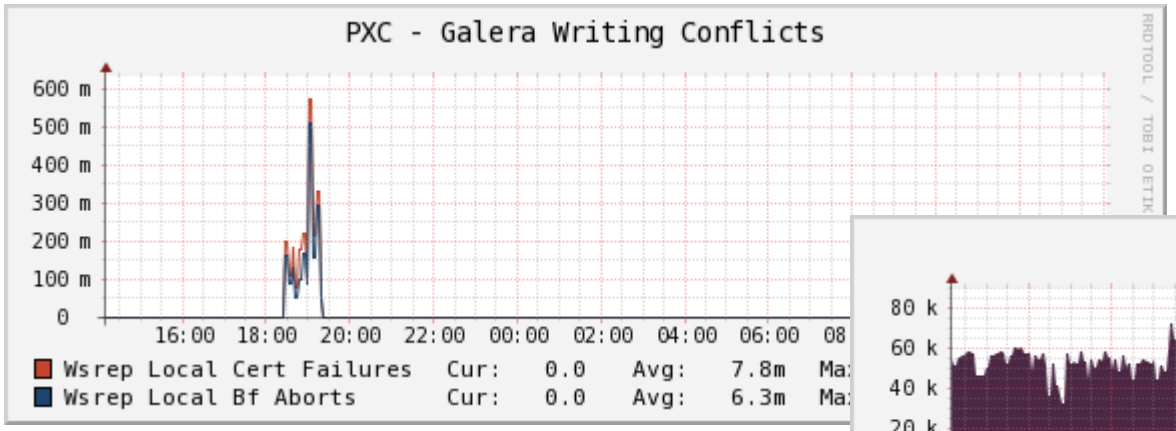
Alerting Best practices

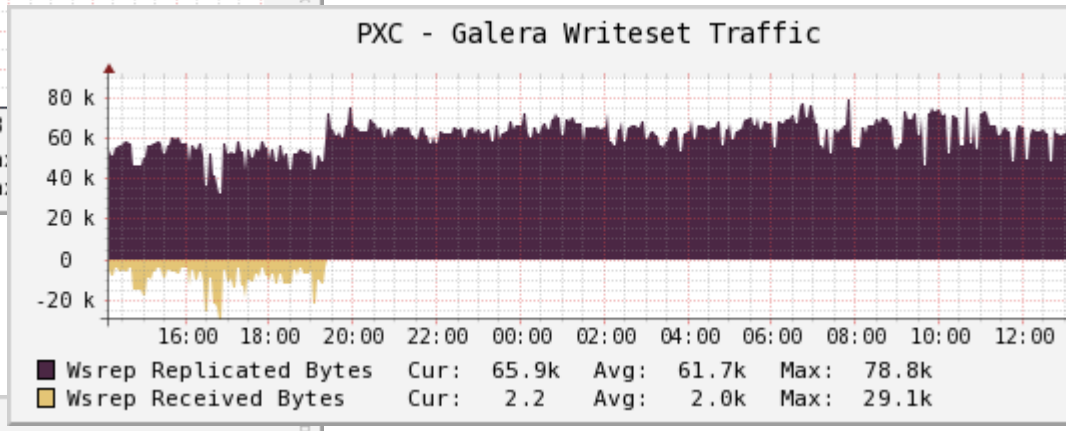
```
1.  $ yum install percona-nagios-plugins -y
2.
3.  # Verify we're in a Primary cluster
4.  $ /usr/lib64/nagios/plugins/pmp-check-mysql-status -x
    wsrep_cluster_status -C == -T str -c non-Primary
5.
6.  # Verify the node is Synced
7.  $ /usr/lib64/nagios/plugins/pmp-check-mysql-status -x
    wsrep_local_state_comment -C '!=' -T str -w Synced
8.
9.  # Verify the size of the cluster is normal
10. $ /usr/lib64/nagios/plugins/pmp-check-mysql-status -x
    wsrep_cluster_size -C '<=' -w 2 -c 1
11.
12. # Watch for flow control
13. $ /usr/lib64/nagios/plugins/pmp-check-mysql-status -x
    wsrep_flow_control_paused -w 0.1 -c 0.9
```

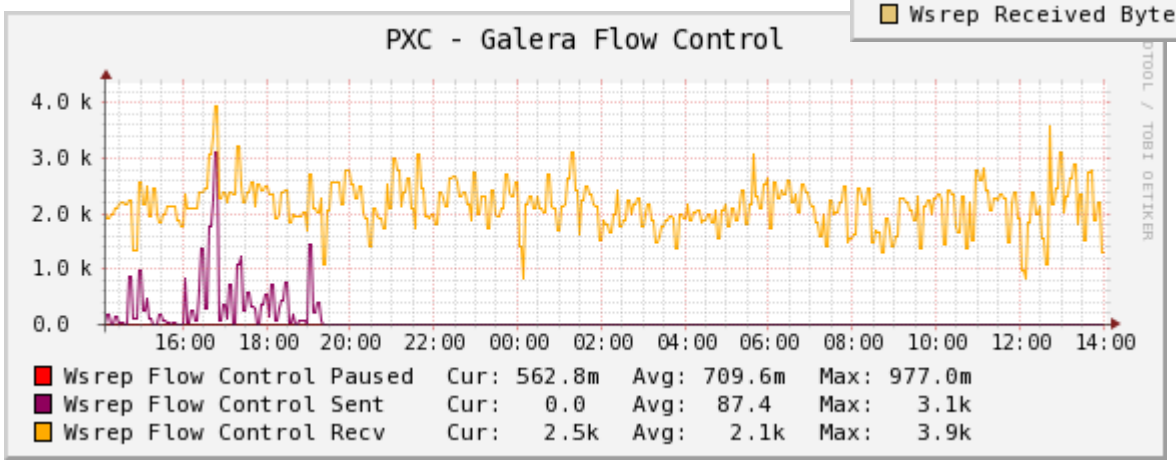
You may want to refer back to this page and run these checks during some of the next experiments

Alerting should be *actionable*, not noise. Any other ideas of types of checks to do?

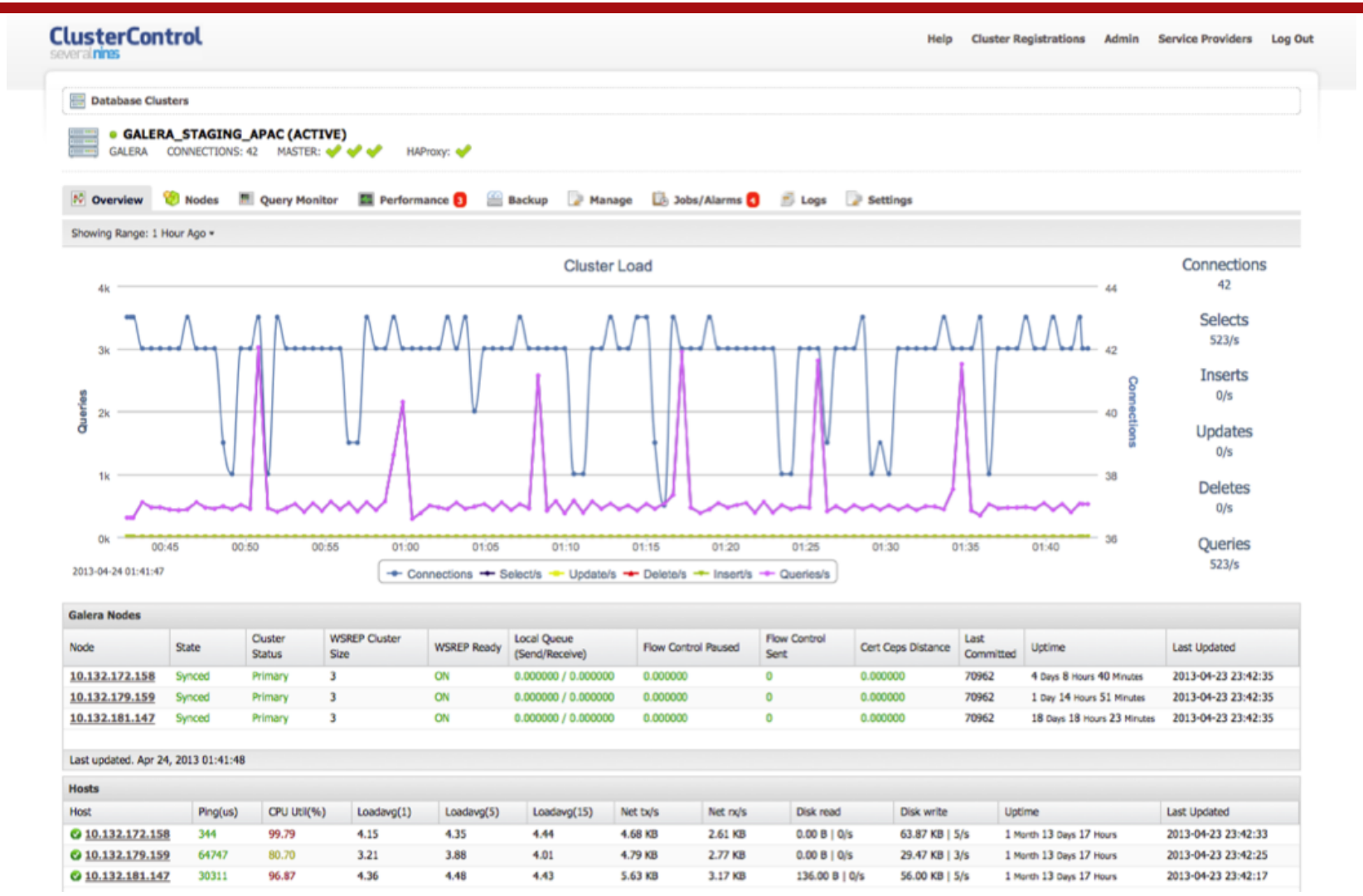
Percona Monitoring Plugins w/ Cacti Galera templates







ClusterControl by Severalnines



PERCONA

Incremental State Transfer

Incremental State Transfer (IST)

- Avoid full SST when possible
- All nodes contain a “Gcache” of recent writesets
- wsrep_provider_options=”gcache.size=2G;”
- IST still pulls from a Donor, like SST

IST uses TCP port 4568 by default

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?

Finding a viable IST donor

```
 1.  # Stop both node2 and node3 in order
 2.  [root@node2 ~]# systemctl stop mysql
 3.  [root@node3 ~]# systemctl stop mysql
 4.
 5.  # Restart node2 and check 'wsrep_local_cached_downto' on each node:
 6.  [root@node2 ~]# systemctl start mysql
 7.  node1 mysql> show global status like 'wsrep_local_cached_downto';
 8.  node2 mysql> show global status like 'wsrep_local_cached_downto';
 9.
10.  # Check node3's GTID.  Which node is the preferable donor?
11.  [root@node3 ~]# cat /var/lib/mysql/grastate.dat
12.
13.  # Start node3 with the proper donor name
14.  # THIS NO LONGER WORKS IN CENTOS 7!
15.  [root@node3 ~]# service mysql start --wsrep-sst-donor=node1
```

What happens if node2 is the donor?

PERCONA

Node Failures, Arbitration, and Bootstrapping

Partial and Total Single Node Failure

```
1.  # Run sysbench on node1 as usual, also watch myq_status
2.
3.  # Experiment 1: clean node restart
4.  [root@node3 ~]# systemctl restart mysql
5.
6.  # Experiment 2: partial network outage
7.  [root@node3 ~]# iptables -A INPUT -s node1 -j DROP; iptables -A OUTPUT
    -s node1 -j DROP
8.
9.  # Experiment 3: node3 stops responding to the cluster
10. [root@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
11.
12. # Remove iptables rules (after experiments 2 and 3)
13. [root@node3 ~]# iptables -F
```

Perform each experiment and note the effect it has on the cluster and the app.
What happens when you recover from each experiment?

A Non-Primary Cluster

```
1.  # Experiment 1: clean stop of 2/3 nodes (normal sysbench load)
2.  [root@node2 ~]# systemctl stop mysql
3.  [root@node3 ~]# systemctl stop mysql
4.
5.  # Experiment 2: failure of 50% cluster (leave node2 down)
6.  [root@node3 ~]# systemctl start mysql
7.  [root@node3 ~]# iptables -A INPUT -s node1 -j DROP; iptables -A OUTPUT
    -s node1 -j DROP
8.
9.  # Allow node1 to become primary like this:
10. node1 mysql> set global wsrep_provider_options="pc.bootstrap=true";
11.
12. [root@node3 ~]# iptables -F # When done
```

Does stopping 2 of the 3 nodes cleanly affect anything?
What eventually happens in Experiment 2? How long? Why?
What happens if you try a query on node1 or node3?
What would the result be if your application were allowed to write to either node in this state?

Using garbd as a third node

```
1.  # Node2's mysql is still stopped!
2.  [root@node2 ~]# yum install Percona-XtraDB-Cluster-garbd-3.x86_64
3.  [root@node2 ~]# garbd -a gcomm://node1:4567,node3:4567 -g mycluster
4.
5.  # Experiment 1: Reproduce node3 failure
6.  [root@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
7.
8.  # Experiment 2: Partial node3 network failure
  [root@node3 ~]# iptables -A INPUT -s node1 -j DROP; iptables -A OUTPUT -
    s node1 -j DROP
9.
10. # Remove iptables rules (after experiments 1 and 2)
11. [root@node3 ~]# iptables -F
```

How does the cluster handle node3's outage?
How does the cluster react if node3 is only connected to the garbd?

Recovering from a clean all-down state

```
1.  # Stop mysql on all 3 nodes
2.  $ systemctl stop mysql   ($ systemctl stop mysql@bootstrap on one)
3.
4.  # Use the grastate.dat to find the node with the highest GTID
5.  $ cat /var/lib/mysql/grastate.dat
6.
7.  # Bootstrap that node and then start the others normally
8.  $ systemctl start mysql@bootstrap
9.  $ systemctl start mysql
10. $ 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

```
1.  # Kill -9 mysqld on all 3 nodes
2.  $ killall -9 mysqld
3.
4.  # Check the grastate.dat again, is it any help?
5.  $ cat /var/lib/mysql/grastate.dat
6.
7.  # Instead try wsrep_recover on each node to get the GTID
8.  $ mysqld_safe --wsrep_recover
9.
10. # Bootstrap the highest node and then start the others normally
11. $ systemctl start mysql@bootstrap
12. $ systemctl start mysql
13. $ systemctl start mysql
14.
15. # On older systems
16. $ service mysql bootstrap-pxc
17. $ service mysql start
18. $ service mysql start
```

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?

PERCONA

Avoiding SST

Manual State Transfer with Xtrabackup

```
1.  # Take a backup on node3 (sysbench on node1 as usual)
2.  [root@node3 ~]# innobackupex --galera-info --no-timestamp backup
3.
4.  # Stop node3 and prepare the backup
5.  [root@node3 ~]# systemctl stop mysql
6.  [root@node3 ~]# innobackupex --apply-log backup
7.
8.  # Get GTID from backup and update backup/grastate.dat
9.  [root@node3 ~]# cp backup/xtrabackup_galera_info backup/grastate.dat
10. [root@node3 ~]# cat backup/grastate.dat
11. # GALERA saved state
12. version: 2.1
13. uuid:    8797f811-7f73-11e2-0800-8b513b3819c1
14. seqno:   22809
15. cert_index:
16.
17. # Copy back the backup and restart
18. [root@node3 ~]# rm -rf /var/lib/mysql/*
19. [root@node3 ~]# innobackupex --copy-back backup
20. [root@node3 ~]# chown -R mysql.mysql /var/lib/mysql
```

Cases where the state is zeroed

```
1.  # Add a bad config item to section [mysqld] in /etc/my.cnf
2.  [mysqld]
3.  fooey
4.
5.  # Check what happens to the grastate:
6.  [root@node3 ~]# systemctl stop mysql
7.  [root@node3 ~]# cat /var/lib/mysql/grastate.dat
8.  [root@node3 ~]# systemctl start mysql
9.  [root@node3 ~]# cat /var/lib/mysql/grastate.dat
10.
11. # Remove the bad config and use wsrep_recover to avoid SST
12. [root@node3 ~]# mysqld_safe --wsrep_recover
13. # 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

```
1.  [root@node3 ~]# cat /var/lib/mysql/grastate.dat
2.
3.  # Turn off replication for the session:
4.  node3 mysql> set wsrep_on=OFF;
5.
6.  # repeat until node3 crashes (sysbench should be running on node1)
7.  node3 mysql> delete from test.sbtest1 limit 10000;
8.
9.  [root@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?

PERCONA

Tuning Replication

Observing Flow Control

```
1.  # Run sysbench on node1 as usual
2.
3.  # Take a read lock on node3 and observe its effect on the cluster
4.  node3 mysql> flush tables with read lock;
5.
6.  # Make observations.  When you are done, release the lock:
7.  node3 mysql> unlock tables;
```

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

Tuning Flow Control

```
1.  # Run sysbench on node1 as usual
2.
3.  # Increase the fc_limit on node3
4.  node3 mysql> set global wsrep_provider_options="gcs.fc_limit=500"
5.
6.  # Take a read lock on node3 and observe its effect on the cluster
7.  node3 mysql> flush tables with read lock;
8.
9.  # Make observations.  When you are done, release the lock:
10. node3 mysql> unlock tables;
```

Now how big does the recv queue on node3 get before FC kicks in?
Try also setting gcs.fc_master_slave=NO and note how that affects the recv queue size for FC.

This setting (like all Galera settings) is tuned separately on each node. The cluster allows this (within reason).

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;
- Such nodes must not be written to!

Testing Node Apply throughput

- Using wsrep_desync and FTWRL 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 and release it, measuring the top speed it applies at
- This gives is an idea of how fast the node is capable of handling the bulk of replication.

Multiple Applier Threads

- Using the wsrep_desync trick, you can measure the throughput of a slave with any setting of wsrep_slave_threads you wish.
- wsrep_cert_deps_distance is the maximum “theoretical” parallel threads
  - Practically, it’s likely you should probably have much fewer slave threads (500 is a bit much)
- Parallel apply *has* been a source of node inconsistencies in the past

Wsrep Sync Wait

- Apply is asynchronous, so reads after writes on different nodes may see an older view of the database.
- Practically 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 <session|global> wsrep_sync_wait=[0-7];

PERCONA

Conclusion

Resources

- **Galera BOF Tuesday 6-7PM in Ballroom B**
- http://www.percona.com/doc/percona-xtradb-cluster/5.6/
- http://galeracluster.com/documentation-webpages/reference.html
- https://github.com/percona/xtradb-cluster-tutorial
- http://www.mysqlperformanceblog.com/category/percona-xtradb-cluster/